HTML文法解析から機械学習まで

JS＋Node.jsによる Webクローラー／ネットエージェント［開発テクニック］

クジラ飛行机 著

まだ、Webデータを手で集めていますか？

- サーバーサイド JavaScript でクローラーが簡単に作れる！
- WebAPI 経由でデータをゲット！ レポートや PDF/Excel 文書を自動作成！
- 豊富なスクリプトがすべてダウンロード可能！

本書で使用している技術

機械学習、画像認識、形態素解析、マルコフ連鎖、ベイジアンフィルタ、GoogleCharts、各種SNSのWebAPIアクセス、HTML解析、リンク抽出、画像抽出 他

ソシム

HTML文法解析から機械学習まで

JS＋Node.jsによる Webクローラー／ネットエージェント［開発テクニック］

クジラ飛行机 著

ソシム

HTML文法解析から機械学習まで

JS＋Node.jsによる Webクローラー／ネットエージェント［開発テクニック］

クジラ飛行机 著

ソシム

●プログラムのダウンロード方法

本書で使用しているプログラム（ソースプログラム）は、弊社の Web ページにリンク先が掲載されています。
また、以下の URL からプログラムの圧縮ファイルを直接ダウンロードすることもできます。

http://www.socym.co.jp/download/993/src.zip

圧縮ファイルには JavaScript のプログラムが含まれているため、Windows などから警告がでることがありますが、そのままダウンロードを継続して下さい。

はじめに

本書は、さまざまな「エージェント」を作ることで、身の回りの作業を自動化することを目的にしています。Web上には、さまざまな有益なデータが存在します。そうしたデータを系統的に収集し、徹底的に分析活用するなら、趣味や仕事に役立てることができるでしょう。本書では、Webにあるデータの収集方法、また、それらを整理し、分析し、活用する方法を紹介します。

Webサイトを巡回するクローラーを作ってデータを収集することで、自分だけのデータベースを構築することができるでしょう。とは言え、そうした集めたデータベースを活用しないで眠らせておくのはもったいないものです。本書が扱うのは、データを集めるだけではありません。自分の嗜好にあった記事を取り出したり、一定の規則で自動分類したり、別のデータと組み合わせたりと、一歩先行く活用法を提案していきます。トレンドを調べて、株やFXのトレードの参考にしたり、アフィリエイトのツールとして活用したり、レポートの自動作成や要約、文章のリライトなど、さまざまな応用が考えられます。

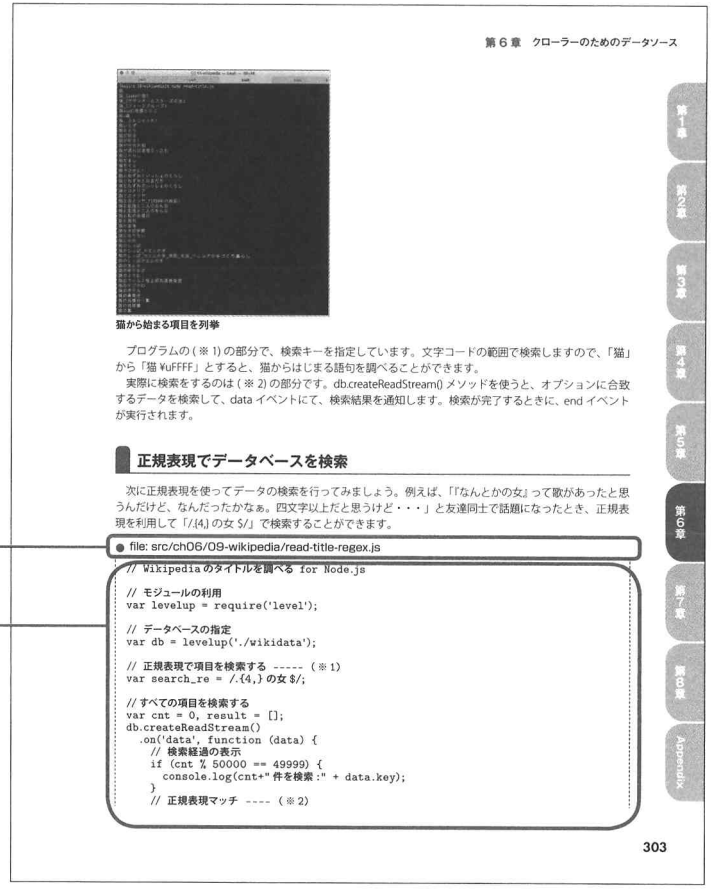
しかも、Webを代表する言語であるJavaScriptを利用して作りますので、プログラムの動作が理解しやすい上に改造も容易でしょう。本書で紹介する技術やプログラムは、そのままで使えるものも多いですが、ちょっと自分用に手を加えることで、より実践で使えるツールとなるはずです。本書のプログラムを叩き台にして、Webや業務データの中に埋もれている宝の山を発掘してみてください。

本書は、JavaScriptの基本が分かっている方、または、プログラミング中級者を対象にしています。JavaScriptが分からない方は、一通りの文法をマスターしておくと、よりよく内容を理解できるでしょう。

- **一歩進んだJavaScriptの活用法を知りたい方**
- **日本語情報の整理・分類・活用に興味のある方**
- **Webで公開されている価値の高い情報の活用方法**

本書の使い方

本書の紙面では、ソースコードを紹介していますが、紙面の都合上、一部を省略していることがあります。ソースコードは弊社のサイトからダウンロードすることができます。ダウンロードのURLは次ページを参考にして下さい。



第6章　クローラーのためのデータソース

猫から始まる項目を列挙

プログラムの（※1）の部分で、検索キーを指定しています。文字コードの範囲で検索しますので、「猫」から「猫 ¥uFFFF」とすると、猫からはじまる語句を調べることができます。
実際に検索をするのは（※2）の部分です。db.createReadStream() メソッドを使うと、オプションに合致するデータを検索して、data イベントにて、検索結果を通知します。検索が完了するときに、end イベントが実行されます。

正規表現でデータベースを検索

次に正規表現を使ってデータの検索を行ってみましょう。例えば、「『なんとかの女』って歌があったと思うんだけど、なんだったかなぁ。四文字以上だと思うけど・・・」と友達同士で話題になったとき、正規表現を利用して「/.{4,}の女 $/」で検索することができます。

● file: src/ch06/09-wikipedia/read-title-regex.js

```
// Wikipedia のタイトルを調べる for Node.js

// モジュールの利用
var levelup = require('level');

// データベースの指定
var db = levelup('./wikidata');

// 正規表現で項目を検索する ----- (※1)
var search_re = /.{4,}の女 $/;

// すべての項目を検索する
var cnt = 0, result = [];
db.createReadStream()
  .on('data', function (data) {
    // 検索経過の表示
    if (cnt % 50000 == 49999) {
      console.log(cnt+"件を検索:" + data.key);
    }
    // 正規表現マッチ ---- (※2)
```

303

第1章 第2章 第3章 第4章 第5章 第6章 第7章 第8章 Appendix

ソースリストのディレクトリ名とファイル名

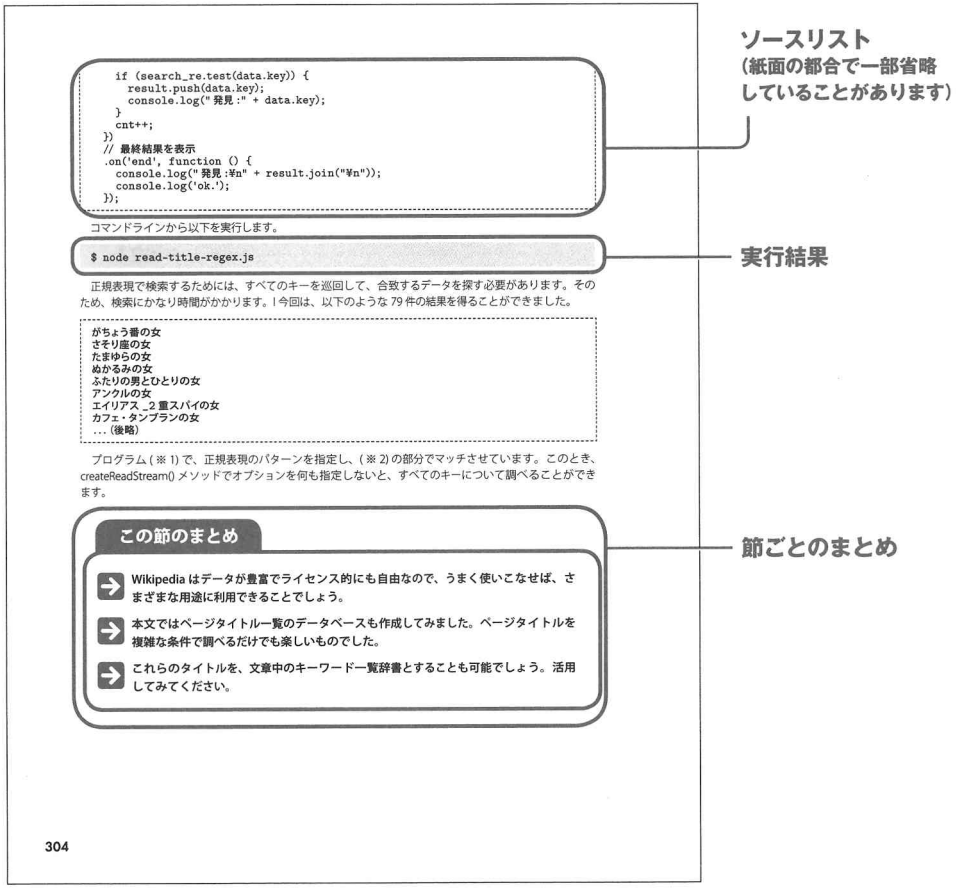
ソースリスト（紙面の都合で一部省略していることがあります）

プログラムのダウンロード方法

本書で使用しているプログラム（ソースプログラム）は、弊社の Web ページにリンク先が掲載されています。
また、以下の URL からプログラムの圧縮ファイルを直接ダウンロードすることもできます。

http://www.socym.co.jp/download/993 ／ src.zip

圧縮ファイルには JavaScript のプログラムが含まれているため、Windows などから警告がでることがありますが、そのままダウンロードを継続して下さい。



CONTENTS

第 1 章　開発環境の準備

第2章　Web データの収集

第3章　ログインの必要なWebサイトをクロールする

第4章　データの整形と保存

第8章　データの視覚化と応用

Appendix

第１章

開発環境の準備

１章では、開発環境を整えます。基本的にJavaScriptはOSを選ばず実行できるのが良い所ですが、本書では開発が便利になるように仮想マシンを利用します。また、JavaScriptの実行エンジンやJavaScriptの開発で役立つオススメのエディタなども紹介します。楽しい宝探しに出る前に、まず装備を調えましょう。

01

JavaScript実行エンジンいろいろ

かつて、JavaScript の互換性の低さは、アプリケーション開発者の悩みの種でした。しかし、現在ではほぼ標準化されており、Web ブラウザーだけでなく多くの環境で動作するようになっています。ここでは、JavaScript の実行エンジンについてまとめてみます。

ここで学ぶポイント

- JavaScript実行エンジンの特徴
- 開発環境のインストール方法

ツールやライブラリの一覧

- Node.js
- Rhino/Nashorn
- JScript

ECMAScript が実現した汎用 JavaScript の世界

ECMAScript（エクマスクリプト）は、Ecma International によって標準化されたスクリプト言語の仕様です。この仕様は、Web ブラウザーごとに異なっていた JavaScript の実装を標準化するために作られました。それ以降、JavaScript は標準化され、ISO/IEC JTC 1、ISO/IEC 16262 として標準化されました。日本でも JIS X 3060 として JIS 規格化されています。現在の最新バージョンは 5 で「The 5th Edition」と呼ばれています。

こうした標準化によって、JavaScript は開発言語としての地位をますます確実にしています。

現在では、Web ブラウザーという枠を飛び越え、さまざまなアプリケーションでマクロ言語的に使われるようになっていきます。

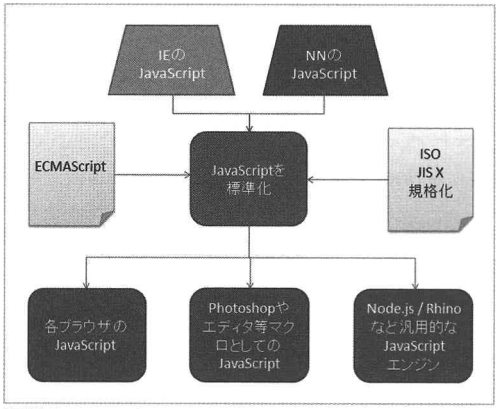
こうしたマクロ的な用途として有名なのは、Photoshop や Illustrator で有名な Adobe の CC（Creative Cloud）シリーズです。これらの製品では、JavaScript を利用して各種作業を自動化することができます。

また、Windows にははじめから JScript エンジンが組み込まれています。これを利用することで、Windows のさまざまな機能に手軽にアクセスすることができます。

さらに、HTML5 の普及により、スマートフォンのアプリさえも、JavaScript で開発できるようになっています。PhoneGap などのフレームワークを利用することで、HTML5 で作った Web アプリをネイティブアプリに変換できる仕組みも整っているので、パフォーマンスに関しても問題はなくなっています。

つまり、JavaScript を覚えたら、Web アプリの開発から、デスクトップアプリやモバイルアプリの開発、バッチ処理の記述まで、さまざまな分野のアプリを開発できる時代となっているのです。

習得が容易で、言語の柔軟性が高く、そしてあらゆる場面で使える JavaScript の評価は、ますます上がっています。



標準化されたことにより、いろいろなところで使われることになった

JavaScript 実行エンジンいろいろ

さて、本書では、JavaScript を利用して、データ収集から分類・分析など、幅広く実践的な情報処理の方法を紹介します。JavaScript だけで、本当にこれら高度な処理を行うことができるのでしょうか？　それを可能にするためには「実行エンジン」の力を借りる必要があります。

優れた絵師が複数の筆を使い分けて、美しくキャンバスに着色していくように、JavaScript の実行エンジンも多くの種類があり、仕事に応じて使い分けることができます。これによって、より短時間で高度な仕事を行うことができます。

本書では主に、Node.js という実行エンジンを利用したプログラムを紹介していきます。Node.js の実用度は、どの JavaScript エンジンも高いものとなっています。

Web ブラウザーで利用できる JavaScript エンジン

もともと、JavaScript は Web ブラウザーに搭載されていたスクリプト言語です。一般的に「JavaScript」と言えば、Microsoft Internet Explorer や Mozilla Firefox、Google Chrome、Apple Safari などの Web ブラウザーに搭載されているスクリプト言語を指します。

これらの JavaScript エンジンは速度も高速になり、さまざまな処理が可能になってきています。

しかし、Web ブラウザーに搭載されている JavaScript エンジンは、Web ブラウザーの中だけで動作するものですから制限が多いのも事実です。ブラウザーの枠を飛び出して、自由にローカル PC 内のファイルを操作することはできませんし、Web サーバーとして動かすこともできません。

そこで登場するのが、Web ブラウザーに依存せず、多用途に利用できる JavaScript エンジンです。

最速エンジンから生まれたエンジン「Node.js」

「Node.js」は、もともと Web サーバーなどのネットワークプログラムを記述するために開発された、JavaScript の実行環境です。その心臓部には、Google Chrome に搭載されている JavaScript 実行エンジン V8 が搭載されています。V8 の特徴は、とにかく高速に動作することです。

また、Web ブラウザーの JavaScript では、セキュリティに配慮されているため、ファイル処理などは実現できませんが、Node.js を使うなら、ファイル処理にはじまりネットワーク処理まで、さまざまな仕事をこ

なすことができます。これに加えて、パッケージマネージャーである「npm」を利用することで、さまざまな拡張機能を容易に導入することもできます。

Node.jsはサーバーで動くJavaScript実行エンジン（サーバーサイドスクリプト）として多く利用されています。そのため、サーバー関連の技術だけがピックアップされがちです。しかし、実際のところ、Node.jsをベースにして、数多くの実用アプリケーションが作成されています。冒頭で、HTML5のアプリをスマホアプリに変換するPhoneGapを紹介しましたが、PhoneGapもNode.jsを利用して開発されているのです。

node.jsのWebサイト

Javaの豊富な資産を利用できる「Rhino」と「Nashorn」

次に、「Rhino」と「Nashorn」を紹介します。

少々紛らわしいですが、これらのJavaScriptエンジンは、「Java」で実装されています。そのため、Javaの仮想マシン(JVM)上でJavaScriptを動かすことができます。

これらJavaScript実行エンジンの最大のメリットは、JavaScriptからJavaの各機能にアクセスできるという点です。つまり、Javaのために用意されている膨大なAPIやライブラリをJavaScriptから利用できます。本書でもJavaのライブラリを利用したプログラムを紹介しますが、これが可能なのは、Java仮想マシン上で動いているからです。

なお、「Rhino」はJava1.4以降、「Nashorn」はJava SE 8以降で利用できるJavaScriptエンジンです。基本的にできることは両者で同じなのですが、Nashornは新たに作り直されただけあって、JVMの新しい機能を利用しており、高速に実行できます。また、開発で重要な点として、エラーメッセージが親切になっており、デバッグも容易になりました。

このようにNashornは利点も多いのですが、本書では、ある程度古いPC上でも動くように、できるだけRhinoを利用したプログラムを紹介します。

Mozilla Rhinoのサイト

Nashorn のブログサイト

Windows に標準搭載されている「JScript」

「JScript」は、Windows でしか利用できないという点が欠点ですが、その分、Windows のさまざまな機能に容易にアクセスできるようになっているのが特徴です。

具体的には、JavaScript を用いて、Windows の ActiveX(COM) 技術に対応した各種アプリケーションの機能を利用することができます。例えば、Excel/Word を遠隔操作し、Excel で作った名簿を読み込んで、Word で作った手紙のひな形に宛名を差し込むといった実務で便利な機能が手軽に利用できます。それこそ「JScript Excel」などのキーワードで検索エンジンを検索すると、Excel を 100% 使いこなすためのさまざまな Tips が見つかることでしょう。

JScript の Microsoft のマニュアル

JavaScript がデータ収集に適している理由

なぜ、JavaScript がデータ収集に適しているのでしょうか。一言で答えるなら、「JavaScript なら手軽に記述できるから」です。今では、多くの Web ページで JavaScript が利用されており、どんな人でも JavaScript を記述できます。そして、JavaScript 用にさまざまなライブラリが用意されています。では、もう少し詳しく見ていきましょう。

習得が容易

JavaScript の基本的な文法はとても簡単です。

基本的な部分だけで言えば、容易に習得できるのではないでしょうか。本書では、JavaScript についてある程度知識があることを前提にしていますが、本書を読みながら、JavaScript の実際的な書き方を学ぶこともできるでしょう。

さまざまなライブラリが用意されている

前述したように、さまざまな JavaScript の処理系があります。そして、JavaScript からさまざまなライブラリを利用できる環境が整っています。本書の大半のプログラムでは、Node.js を利用していますが、Node.js のために書かれた多くの有益なライブラリが用意されています。加えて、それらをパッケージマネージャーの npm を利用して、手軽にインストールすることができるのです。

柔軟性が高く素早く処理を記述できる

JavaScript の柔軟性の高さには定評があります。JavaScript は、プロトタイプ型のオブジェクト指向言語を採用しています。オブジェクト指向を利用して整然と処理を記述していくことができます。また、Node.js などではモジュールの機構を備えているため、それを利用することで、機能をモジュール単位で管理することができるようになっています。

この節のまとめ

- この節では、JavaScript の各種実行エンジンについて紹介しました。
- ここで見たように JavaScript の実行環境にはいろいろなものがあります。
- なかでも Node.js が優勢で多くの開発者が支持しています。
- 本書でも、Node.js を利用したプログラムを多く紹介します。
- さらに、Rhino/Nashorn を利用すれば、Java 用に用意された数限りないライブラリを JavaScript から手軽に活用できます。

02 エージェントとは何か？

本書では JavaScript を使ってデータ収集やデータ活用を行う方法を紹介しています。ここでは、本書が目指す内容を簡単に説明しつつ、これらを実行するプログラム＝「エージェント」を作成するために、どんな分野の技術が必要となるのか紹介していきます。

ここで学ぶポイント	ツールやライブラリの一覧
●エージェントについて	●なし

エージェントの意味

「エージェント (Agent)」とは、他の個人や組織から依頼され、その代わりに行動する個人や組織を指します。つまり、代理人、代理業者、仲介業者と言い換えることもできます。スポーツ選手の代わりにチームと交渉するのもエージェントです。また、芸能界でタレントに代わってさまざまな交渉を行うのもエージェントです。身近なところでいえば、旅行者の代わりに旅行の手配をするのは「旅行エージェント」です。土地や建物について、売り主と買い主の間に入って折衝業務を行うのが「不動産エージェント」です。

こうした作業、処理、業務を IT の分野に適用したのが「ソフトウェア・エージェント」です。つまり、ユーザーあるいは他のソフトウェアに代わって仲介するようなプログラムのことをいいます。本書では、簡単に「エージェント」と呼ぶことにします。

本書では、Web サイトからさまざまなデータを収集し、そのデータに基づいて、何かしらの処理を自動化するエージェントの作り方を紹介します。そのために、データ収集の方法、データの整形と保存、日本語処理の方法やデータ分析手法など、さまざまな技術を広く紹介していきます。最初から全部読んでも良いですし、必要に応じて特定の知識を増強することもできるようにもなっています。

知的エージェント (IA) について考えてみる

ところで、ソフトウェアで作るエージェントにはどのようなものがあるでしょうか。いくつか具体的なエージェントについて考えてみましょう。

この分野で突出しているのは「知的エージェント (Intelligent Agent、IA)」です。人工知能の機能を有し、ユーザーを補助し、繰り返し行うタスクをユーザーに代わって行います。知的エージェントは次のような特徴を持っています。

- **環境との相互作用によって学習し動作が改善される**
- **オンラインでリアルタイムに適応する**
- **大量のデータから高速に学習する**
- **新たな問題解決規則に適応する**
- **自身の動作に関して、成功と失敗を自己分析する**

特徴を流し読みすると、SF に出てくる高度な知能を持ったロボットを想像してしまうかもしれません。しかし、すでに私たちの身近で IA は活躍しています。いくつかその実例を見ていきましょう。

バイヤーエージェント

「買い物ボット」とも言われています。ユーザーがインターネット上で商品やサービスを見つけるのを助けるエージェントです。例えば、Amazon などで買い物をする時に、ページの下に、類似商品が表示されます。これは、そのページを見た他のユーザーが買った本や、ユーザーが他に見た商品の傾向などの情報を利用したものです。専門店の店員さんであれば、お客さんの声に耳を傾け、お客さんにぴったり合う商品を選んでくれます。薬剤師であれば、お客さんの症状を聞いて、ぴったりの薬を薦めてくれるでしょう。これと同じで、バイヤーエージェントは、ユーザーの嗜好を考慮して、同様の商品を判断します。

ユーザーエージェント

インターネットの世界では、ユーザーエージェントが大活躍しています。例えば、Web ブラウザーや電子メールのクライアントがそれで、ユーザーのために自動的にタスクを実行します。電子メールであれば、新着メールを受信し、ユーザーの指示に従ってメールを分類したり、重要なメールがあればユーザーに知らせたりします。同様に、Web ブラウザーは、ユーザーと Web サーバーの間に立って情報をやりとりするエージェントです。ユーザーが見たいページを伝えると、Web ブラウザーは、サーバー上にあるデータを取得してきて、ユーザーに見やすい形で表示します。もう少し高度なエージェントになると、ユーザーの嗜好に応じて、Web 上にあるニュースを集めることもします。また、ゲームの相手をしたり、ユーザーと対話したりもします。

監視エージェント

そして、地味ながらなくてはならない働きをするのが「監視エージェント」です。このエージェントは、コンピューター機器やシステムが正常に動作しているかどうかを監視して報告します。機器が故障したり、システムがダウンしたりすると、管理者に通知します。また、小売店などの在庫状況を監視して品切れの発生を予測して報告します。もちろん、可能であれば、自動で復旧を試みます。

データマイニング・エージェント

データマイニングとは、さまざまな情報源に対して、そのデータの中から有用なデータを探し出すことを指します。この作業は、鉱山に眠る鉱石を「採掘 (mining)」する作業になぞらえて、「データマイニング (data mining)」と呼ばれます。このエージェントは、まず情報を集めることから始めて、集めたデータを分類・分析します。そこから、トレンドの転換を検出したり新しい情報を検出したりします。

たとえば、有名なデータマイニングの例として、スーパーの販売データを分析することで、以下のような意外な情報が判明しました。

- **ビールを買う客は一緒にオムツを買うことが多い**
- **雨の日は肉の売上が良い**

こうした発見をすることができれば、ビールとオムツの売り場を近くする、雨の日に仕入れる肉の種類を増やすなど、簡単な施策で売り上げを伸ばすことができるでしょう。

このように、エージェントは既に私たちの身の回りで利用されています。残念ながら、本書では、すべての内容を網羅できているわけではありません。しかし、本書で紹介した技術を入り口として、深いところへ潜っていくことができるはずです。

この節のまとめ

- **エージェントとは、その人の代わりに何か仕事をする人や物のことです。**
- **私たちの周りには、既にいろいろなエージェントがあり、通販サイトの中や、ブラウザやメールソフトといったアプリの形で提供されています。**
- **本書はコンピューターを使った「エージェント」の作り方を解説していきます。**

03 開発環境を構築しよう

仮想マシンを用意して、そこに開発環境を構築しましょう。仮想マシンを用意して、その中でプログラムを動かすなら、安心してプログラム実行できます。そこで、ここでは仮想マシンで手軽に開発環境を構築する方法を紹介します。

ここで学ぶポイント

- 仮想マシンに開発環境を構築する

ツールやライブラリの一覧

- Vagrant / VirtualBox
- CentOS
- Node.js
- Poderosa
- nvm
- git

仮想マシンに実行環境を構築しよう

本書で紹介しているプログラムは、Windows/Mac OS X/Linux のいずれでも動かすことができます。しかし、プログラムを開発する際には仮想環境を構築しておくと便利です。本書では、仮想マシン上に CentOS をインストールして利用することにします。これにより、細かい OS の差異を気にせずにプログラムの解説に集中することができます。

また、仮想環境を作っておけば、いろいろな実験を行うことができますし、環境が壊れても、すぐに元に戻すことができます。どんなに無理なプログラムを動かしても、ホストマシンの環境に影響が及ばないからです。

無償で多くの仮想化のためのツールが提供されていますので、これを使わない手はありません。最近の PC ならば、仮想マシンを起動しっぱなしにしておいてもホストマシンがそれほど重くなることもありません。

本書ではコマンドライン上で、プログラムを実行したり、追加ライブラリをインストールする場面が多くあります。もちろん、Windows 上でもこうした操作は可能なのですが、やはり、Windows のコマンドラインよりも、Linux 系のターミナルの方が使い勝手が良いという事情があります。また、標準でパッケージマネージャーが備わっていますので、ちょっとしたツールのインストールも簡単です。

仮想マシンを作成しよう

では、仮想マシンを準備する方法を紹介していきます。

ここで作成する仮想マシンは、VirtualBox 上に CentOS のインストールされた仮想マシンを作成するというものです。OS のインストールは非常に面倒で時間のかかる作業です。そこで、CentOS

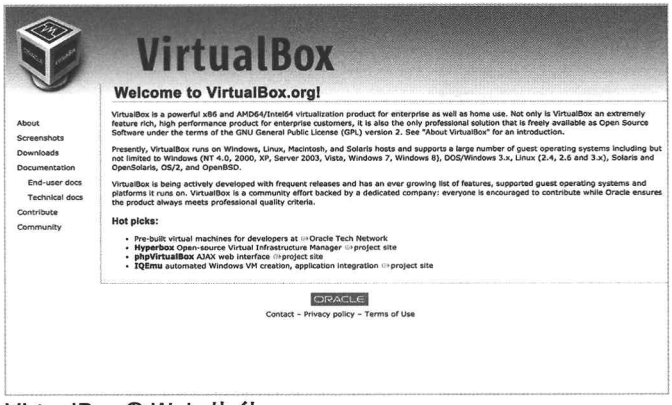
のインストールされたマシンイメージをコマンド一発でダウンロードできる、Vagrant というツールを利用します。

VirtualBox と Vagrant のインストール

それでは、VirtualBox と Vagrant をインストールしましょう。

VirtualBox は以下の URL から入手できます。Windows ならば「VirtualBox x.x.x for Windows hosts」を、Mac OS X ならば「VirtualBox x.x.x for OS X hosts」を選んでクリックします。インストーラーが用意されているので、「次へ」「次へ」とクリックしていけばインストールが完了します。

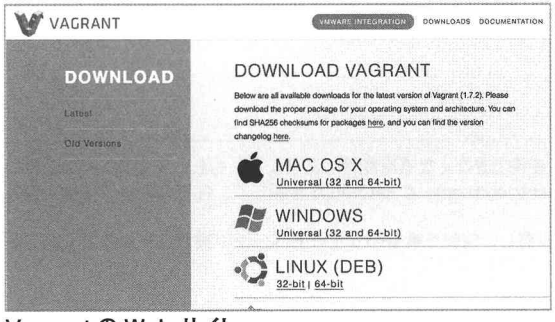
```
VirtualBox > Downloads
https://www.virtualbox.org/wiki/Downloads
```



VirtualBox の Web サイト

次に、Vagrant をインストールしましょう。以下の URL から入手できます。こちらもインストーラーが用意されているので、ウィザードの指示にしたがってインストールを行います。

```
Vagrant > Download
https://www.vagrantup.com/downloads.html
```



Vagrant の Web サイト

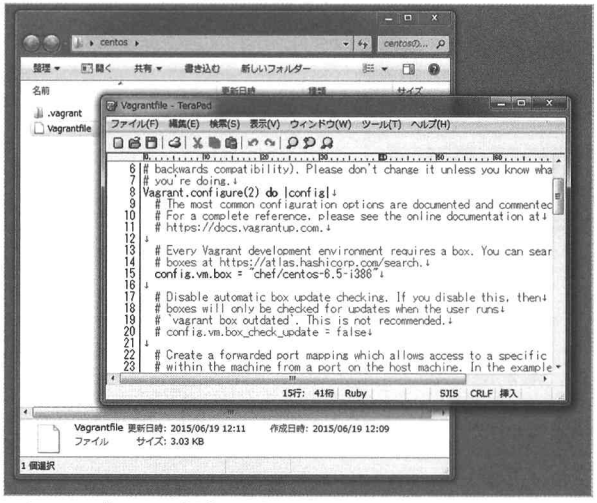
仮想マシンを追加しよう

Windows ならばコマンドプロンプト、Mac OS X ならばターミナルを開いて、仮想マシンを作成したいディレクトリに移動しましょう。その上で、「vagrant init」コマンドを実行します。すると、Vagrantfile が自動的に作成されます。

```
$ vagrant init
```

テキストエディターを使って、このファイルを編集します。そして、このファイルの中の「config.vm.box = "base"」と書かれている部分を探して、以下のように書き換えましょう。これは、「Vagrant Cloud」から自動で仮想マシンファイル (Box ファイル) をダウンロードするように指示するものです。*

```
# config.vm.box = "base"                    ← コメントアウト
config.vm.box = "puphpet/centos65-x64"      ← このように書き換える
```



テキストエディターで編集します

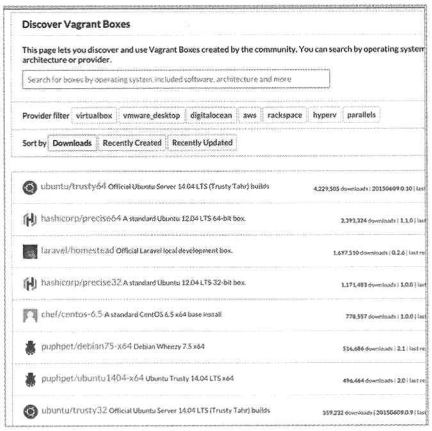
ここでは、CentOS 6.5 の仮想マシンを作成する設定ですが「Vagrant Cloud」には、他にもいろいろな仮想マシンの Box が登録されているので、好きなものを探すことができます。

```
Vagrant Cloud の Web サイト
[URL] https://vagrantcloud.com/
```

---

* この Box ファイルは有志によって作られているので、使用できなくなる可能性もあります。もし、Box ファイルが見つからなければ、本文で述べているとおり、「http://vagrantcloud.com」から CentOS を探して、任意の Box ファイルの名前を記述してみてください。
また、Vagrant の環境がうまく構築できない場合、より詳しい情報を筆者のサポートページに掲載しますので、以下を参考にしてみてください。

● Vagrant で CentOS をインストール
http://kujirahand.com/blog/index.php?Vagrant で CentOS をインストール



Box ファイルの一覧

仮想マシンの起動

これで準備は完了です。以下のコマンドを実行しましょう。すると、自動で Box ファイルがダウンロードされて、仮想マシンが起動します。

```
$ vagrant up
```

初回起動では Box ファイルをダウンロードする

初回の起動では、Box ファイルがダウンロードされるので、時間がかかりますが、２回目以降の起動は、それほど時間がかかりません。

仮想マシンの操作

マシンが起動しているか確認するには、以下のコマンドを実行すると状態を調べることができます。正しく起動していれば「running(動作中)」と状態が表示されます。

```
$ vagrant status
```

起動した仮想マシンを終了するには「vagrant halt」と入力します。

他にも、次のようなコマンドで仮想マシンを操作することができます。

コマンド	操作の説明
vagrant up	仮想マシンを起動する
vagrant halt	仮想マシンを停止
vagrant suspend	仮想マシンをスリープさせる
vagrant resume	仮想マシンをスリープから復帰させる
vagrant reload	仮想マシンの再起動
vagrant status	仮想マシンの状態を確認
vagrant destroy	仮想マシンを破棄
vagrant ssh	仮想マシンへログインする

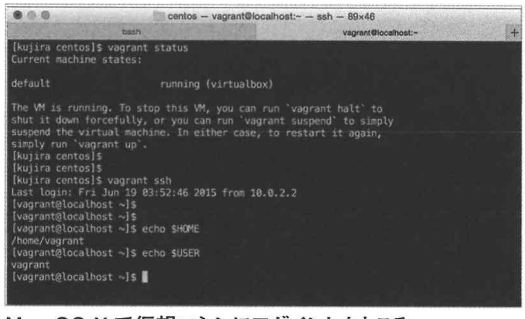
Vagrant を踏査するコマンド一覧

仮想マシンにログインしよう

先ほど仮想マシンにインストールした CentOS のイメージは、最小限の構成なので、デスクトップ画面などを持たないものです。そのため、ホストマシンから仮想マシンを利用するためには、仮想マシンにログインします。

Mac OS X であれば、ターミナルを開いて、以下のようにコマンドを実行すれば、SSH を利用して仮想マシンにログインできます。

```
$ vagrant ssh
```



Mac OS X で仮想マシンにログインしたところ

Windows で仮想マシンにログインする方法

Windows の場合には、標準で SSH クライアントがインストールされていないために、以下のように、SSH 接続に必要な情報が表示されます。

```
> vagrant ssh

（以下表示例）
Host: 127.0.0.1
Port: 2222
Username: vagrant
Private key: C:/Users/kujira/Desktop/centos/.vagrant/machines/default/
virtualbox
/private_key
```

SSH クライアントをインストールして、この情報を設定する必要があります。Windows の有名な SSH クライアントとしては、Putty、TeraTerm や、Poderosa などがあります。ここでは、Poderosa を利用する方法を紹介します。

以下より、Poderosa をダウンロードします。ZIP ファイルを解凍すると、「Poderosa.exe」があり、これがメインの実行ファイルです。ダブルクリックで実行しましょう。

```
Poderosa
[URL] http://sourceforge.net/projects/poderosa/
```

Poderosa を起動したら、メニューから [ファイル > 新規 Telnet/SSH 接続] をクリックします。そして、先ほど「vagrant ssh」で得た接続情報を指定します。ホストに「127.0.0.1」、プロトコルに「SSH2」、ポートにポート番号 (Port)、アカウントに「vagrant」、認証法を「公開鍵」にして、鍵ファイル (Private key) を指定します。このとき、ポート番号が標準の 22 ではなく、2222 など異なる番号になっているので注意しましょう。「OK」を押すと、仮想マシンに接続します。

Poderosa で接続するところ

Poderosa に限らず、SSH クライアントは画面カラーを自由に変更できます。メニューの「ツール > オプション」から「表示」タブを開くと画面カラーを設定することができます。

Poderosa で実行したところ

はじめからインストールされているデフォルトアカウントは「vagrant」で、パスワードは「vagrant」となっています。

デフォルト	
アカウント	vagrant
パスワード	vagrant

デフォルトアカウント

Node.js をインストール

それでは、本書で主に利用する JavaScript エンジンの「Node.js」をインストールしてみましょう。Vagrant で仮想マシンにログインします。

Node.js はバージョンによって動作が異なります。そこで、nvm というツールを使って任意のバージョンの Node.js をインストールすることにしましょう。以下のコマンドを実行すると、nvm がインストールされます。

```
$ curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.25.3/install.
sh | bash
```

シェルを再起動するように促されますので、一度、仮想マシンからログアウトして、再度ログインしましょう。続いて、本書が推奨する Node.js の 0.12.4 をインストールしてみましょう。

```
$ nvm install  v0.12.4
$ nvm alias default v0.12.4
```

それでは、正しくインストールできたか確かめてみましょう。「node -v」コマンドで、Node.js のバージョン情報を調べることができます。バージョンは異なるかもしれませんが、以下のように表示されれば成功です。

```
$ node -v
v0.12.4
```

Node.js では、REPL と呼ばれる対話環境が用意されています。「node」コマンドを実行すると、REPL が起動します。そこで、3 + 5 と入力して Enter キーを押してみてください。「8」と結果が表示されます。

```
$ node
> 3 +5
8
```

ただし、nvm を利用して Node.js をインストールした場合、sudo コマンドを利用した時に、npm や node コマンドが利用できません。以下のコマンドを実行して、sudo コマンド実行時のパスを普段のユーザーのパスと同等にします。まずは、下記のように visudo コマンドを実行します。

```
$ sudo visudo
```

たくさんの設定項目がありますが、env_keep を設定している以下の行を探して、コメントの「#」を外します。ただし、visudo を実行した場合、「vi」エディターが起動することになっています。「vi」エディターでは、カーソルの移動などを行う「コマンドモード」と文字の入力を行う「入力モード」を切り替えて編集を行います。起動したデフォルトでは「コマンドモード」になっています。「i」キーを押すと「入力モード」に切り替わります。編集が終わったら [ESC] キーを押します。すると「コマンドモード」に切り替わります。

```
### 一つ目の変更点（env_reset を無効に）
Defaults    env_reset
      ↓
Defaults    !env_reset

### 二つ目の変更点（HOME を追加）
# Defaults   env_keep += "HOME"
      ↓
Defaults   env_keep += "HOME"

### 三つ目の変更点（パスを書き換えない）
Defaults    secure_path = /sbin:/bin:/usr/sbin:/usr/bin
      ↓
# Defaults    secure_path = /sbin:/bin:/usr/sbin:/usr/bin
```

そこで「:wq」キーを押すと、保存して編集を終了します。一度ログアウトして、再度、ログインすると、設定が反映されます。

git のインストール

また、さまざまなオープンソースのプロジェクトのソースコードをローカル環境に取得することができる「git」もインストールしておきましょう。

```
$ sudo yum install git
```

仮想マシンの Web サーバーを利用できるようにする

さて、仮想マシンは言わば箱庭です。そのため、仮想マシンで Web サーバーを作成した場合、ホストマシンからは、仮想マシンの Web サーバーにアクセスすることができません。アクセスできるようにするには、Vagrant の設定ファイル「Vagrantfile」に以下の設定を加えます。

```
config.vm.network "forwarded_port", guest: 80, host: 8080
config.vm.network "private_network", ip: "192.168.33.10"
```

設定ファイルの行末にある「end」より上に記述してください。設定を記述したら、仮想マシンを再起動しましょう。コマンドラインで「vagrant reload」を実行してください。

一行目の設定を行うことで、仮想マシンの Web サーバーのポート 80 が、ホストマシンのポート 8080 に割り当てられます。二行目で、仮想マシンの IP アドレスが、192.168.33.10 に割り当てられます。

ホストマシンと仮想マシンでフォルダを共有する

ホストマシンと仮想マシンでフォルダを共有することも可能です。デフォルトでは、仮想マシンの「/vagrant」フォルダが、ホストマシンの設定フォルダ (Vagrantfile のあるフォルダ) と共有されています。

もちろん、ホストマシンの任意のフォルダを、仮想マシンの任意のフォルダと共有することができます。そのためには、設定ファイル「Vagrantfile」に以下のような設定を記述します。

```
[書式]
config.vm.synced_folder "host_path", "guest_path"
```

設定を変更したら、やはり、「vagrant reload」で再起動しましょう。

仮想マシンのフォルダ共有でエラーが出る場合

もし、上記の設定でうまく共有できない場合は、vbox をリビルドすることで問題を解決できることがあります。仮想マシンにログインしたら、以下のコマンドを実行します。

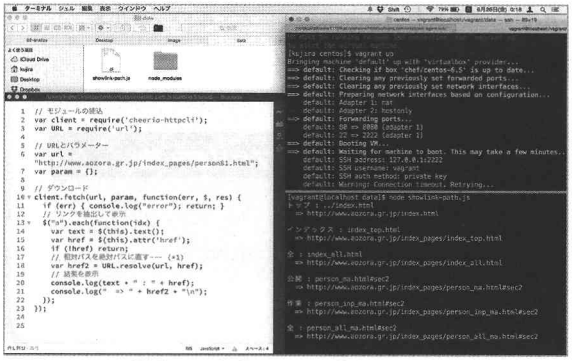
```
$ sudo /etc/init.d/vboxadd setup
```

あるいは、ホストマシンで、プラグインをインストールしておくと便利です。仮想端末の Virtualbox-guest-addtion のインストール状況を確認し、必要があれば自動的にインストールしてくれます。

```
$ vagrant plugin install vagrant-vbguest
```

ホストマシンで開発し、仮想マシンで実行する

ホストマシンと仮想マシンでフォルダが共有できると便利なことはたくさんあります。例えば、ホストマシンの Windows（あるいは Mac OS X）でプログラムを作成して、仮想マシン上の CentOS でプログラムを実行するということが簡単になります。また、本書で紹介するプログラムはダウンロードすることができますが、プログラムのダウンロードや解凍するのはホストマシンで行って、実行は仮想マシン上で行うということができます。



Mac OS X で開発し、仮想マシンで実行テストしているところ

この節のまとめ

- この節では仮想マシンを利用して、開発環境を整える方法を紹介しました。
- Vagrant を利用することで、比較的短時間で開発環境を整えることができたのではないでしょうか。
- 本書では、これ以降、基本的にこの仮想マシン上で動作を確認していきます。

04

Node.jsモジュールのインストール

Node.js の利点は、たくさんのモジュールが公開されている点にあります。モジュールをインストールするにはパッケージマネージャーの「npm」を利用するのが便利です。ここでは、npm について理解を深めましょう。

ここで学ぶポイント	ツールやライブラリの一覧
●npmの使い方	●npm ●nano

「npm」とは何か？

npm とは、Node.js のモジュールを管理するためのパッケージマネージャーです。基本的に Node.js のモジュールをインストールすることを目的としています。しかし、それだけでなく、Node.js で作られた便利なツールをインストールすることもできます。

npm install モジュールのインストール

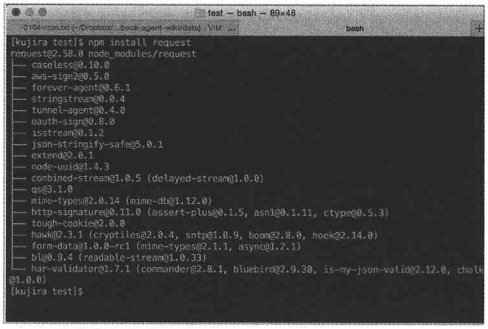
では、npm の基本的な使い方を紹介します。モジュールをインストールするには、「npm install」を利用します。コマンドラインで次のようなコマンドを実行します。

```
［書式］npmでモジュールをインストールする
$ npm install （モジュール名）
```

例えば、Web サイトからファイルやデータをダウンロードするのに「request」モジュールをよく使います。request モジュールをインストールするには、以下のようなコマンドを実行します。

```
$ npm install request
```

コマンドを実行すると以下のようになります。



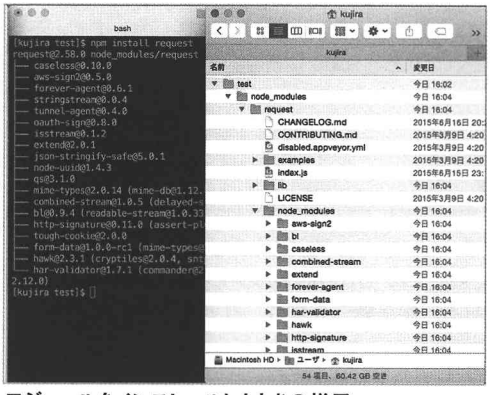
npm を実行したところ

上記を見るとわかりますが、「npm install」を実行すると、そのモジュールの中で利用している別のモジュール（つまり、依存しているモジュール）も一緒にインストールされるのです。

そのため、request モジュールが使いたいから、mime-types モジュールと form-data モジュールもインストールするという手間を省くことができます。これは、非常に便利な仕組みで、npm を利用するメリットであると言えるでしょう。

モジュールがインストールされる場所について

ただし、注意すべき点もあります。基本的に「npm install」で実行すると、このコマンドを実行したカレントディレクトリにモジュールがダウンロードされます。具体的には、カレントディレクトリに、<node_modules> というディレクトリが作成され、そこにモジュールがダウンロードされるようになっています。



モジュールをインストールしたときの様子

そのため、一度モジュールをインストールしたからと言って、別のディレクトリに配置したプロジェクトからは、このモジュールにアクセスできないということです。

少し整理してみましょう。以下のようなディレクトリ構成の二つのプログラムがあったとします。

```
+ <root>
|---+ <ProjectA>
|   |---+ <node_modules>
|   |--- program-a.js
|
|---+ <ProjectB>
|   |--- program-b.js
```

このとき、<ProjectA> の program-a.js で、request モジュールを使うので、<ProjetA> のディレクトリで「npm install request」を実行します。すると、program-a.js で request モジュールが利用できるようになります。しかし、<ProjectB> の program-b.js では、request モジュールを見つけることができないので、改めて、<ProjectB> のディレクトリで、再度、request モジュールをインストールする必要があるということです。

グローバルインストール「-g」

しかし、よく使うモジュールで、毎回インストールするのは面倒だという場合もあります。すべてのプロジェクトから利用したいという場合は、モジュールをインストールする際に「-g」オプションを付けます。これにより、グローバルなパスにインストールすることができます。

```
# モジュールをグローバルインストールする
$ npm install -g（モジュール名）
```

「-g」オプションを付けてインストールすることを「グローバルインストール」と言います。これに対して、「-g」を付けない方法を「ローカルインストール」と言います。

そして、グローバルインストールを行う場合の注意点ですが、大抵の環境では、実行に管理者権限が必要となります。CentOS や、Mac OS X では、コマンドの冒頭に管理者権限でコマンドを実行する「sudo」を付けて以下のように記述します。

```
# 管理者権限を付けてグローバルインストールする
$ sudo npm install -g（モジュール名）
```

「sudo」コマンドを実行した場合には、パスワードを尋ねられますので、パスワードを入力します。正しいパスワードが入力され、管理者権限があることが確認されたなら、sudo に続くコマンドが管理者権限で実行されます。

グローバルインストールのパスに注意

グローバルインストールを行うと、CentOS5 の場合では「/usr/lib/node_modules」のパスにインストールされます。Mac OS X に Homebrew でインストールした場合では「/usr/local/lib/node_modules」にインストールされます。どこにモジュールがインストールされるのかを調べるには、以下のコマンドを実行すると確かめることができます。

```
# グローバルインストールのインストール先を調べる
$ npm root -g
```

よく起きるトラブルとしては、npm でグローバルインストールしたモジュールを、Node.js が探せないということがあります。そんな時には、以下のようなメッセージが表示されます。

```
Error: Cannot find module '（モジュール名）'
```

これは、そもそもモジュールをインストールしていない時、あるいは、Node.js と npm がうまく連携出来ていない時に、パスが見つからないというエラーです。

ところで、Node.js はモジュールを探すとき、どのパスを検索するのでしょうか。以下のコマンドを実行することで、モジュールの検索パスを表示してくれます。

```
$ node -e "console.log(global.module.paths)"
```

例えば、CentOS で実行すると、以下のように表示されます。

```
[ '/home/vagrant/test/node_modules',
  '/home/vagrant/node_modules',
  '/home/node_modules',
  '/node_modules' ]
```

つまり、Node.js では、カレントディレクトリの node_modules、その親の node_modules、またその親の node_modules……のようにモジュールを順に検索していくのです。

それから、Node.js は環境変数の「NODE_PATH」も検索します。そこで、「npm root -g」コマンドを実行して、グローバルインストールされるパスを調べます。その上で、環境変数「NODE_PATH」にグローバルなパスを登録します。すると、Node.js でグローバルインストールしたモジュールのパスを正しく検索してくれるようになります。

以下は、仮想マシンの CentOS で、「npm root -g」を実行したところです。この値を覚えておきます。

```
$ npm root -g
/home/vagrant/.nvm/versions/node/v0.12.4/lib/node_modules
```

そして、Linux(CentOS) や Mac OS X で環境変数に値を登録するには、ユーザーのホームディレクトリにある「~/.bash_profile」（仮想マシンの環境では、/home/vagrant/.bash_profile」を編集します。以下のように、グローバルインストールされるパスを指定します。

```
# 環境変数「NODE_PATH」を設定
export NODE_PATH=/home/vagrant/.nvm/versions/node/v0.12.4/lib/node_modules
```

環境変数「NODE_PATH」を参照するのは、Windows の Node.js でも同じです。Windows で環境変数を編集するには、コントロールパネルの「システムのプロパティ＞詳細設定＞環境変数」から行います。

CentOS にわかりやすいエディターをインストールする

ところで、仮想マシン上の CentOS でファイルを編集するには、どうしたら良いでしょうか。一般的にテキストエディターの「vi」を利用します。しかし、「vi」の使い方は初心者には難しいものです。「vi」を起動したら最後、文字を打つことも、vi を終了することもできないと思うでしょう。もし、vi を起動してしまったら「:q!」と Enter キーをタイプして終了させます。

そこで、「vi」に変わってメモ帳ライクに使える「nano」エディターをインストールしておきましょう。CentOS では以下のコマンドを入力すると nano がインストールされます。

```
$ sudo yum install nano
```

インストールが完了したら「~/.bash_profile」を編集してみましょう。以下のように入力します。

```
$ nano ~/.bash_profile
```

編集が完了したら、[Control] キーを押しながら [x] キーを押します。すると編集したテキストを保存するか尋ねられるので「y」キーと Enter キーを押しましょう。するとファイル名を尋ねられますので、そのまま Enter キーを押すと上書きすることができます。

nano エディターを起動したところ

ところで、「nano」エディターの画面下部には、どのキーを押すと何が起きるのかが記されています。「^X」と書いてあるのは、Control キーを押しながら [X] キーを押すという意味になります。

モジュールのアンインストール

モジュールのインストールの仕方がわかったところで、アンインストールの方法も紹介しましょう。モジュールが不要になった時に「npm uninstall」で削除することができます。

```
# [ 書式 ] モジュールをアンインストール
$ npm uninstall （モジュール名）
```

この節のまとめ

- この節では、Node.js のパッケージマネージャーである「npm」について紹介しました。
- モジュールのインストールとアンインストールに関して、落とし穴となりそうな部分に関して書いてみました。
- グローバルインストールについてはよく理解しておくと良いでしょう。

05

開発効率を高めるモダンなエディターを紹介

この節では、JavaScript の開発を行う上で便利なプログラマー向けのテキストエディターを紹介します。この世にテキストエディターはたくさんありますが、特に、今注目の「モダンなエディター」を紹介します。本書のプログラムを実行したり、プログラムを改造したりする上で役立つでしょう。

ここで学ぶポイント

- モダンなJavaScript開発エディターの紹介

ツールやライブラリの一覧

- Atom
- Brackets
- NetBeans
- Sublime Text
- WebStorm

JavaScript を記述するのに適したエディターは？

結論から言ってしまうと、JavaScript を書くのに、特別なエディターは必要ありません。お気に入りのテキストエディターを使えば良いでしょう。本書の多くのプログラムは、コマンドラインからコマンドを入力して実行するものとなっています。テキストエディターで、JavaScript のプログラムを確認し、コマンドを実行して結果を確認してみてください。

ですから、テキストエディターに求められるのは、JavaScript の文法に応じたカラーリングが行われることと、必要に応じて簡単な入力補助があれば良いということになるでしょう。

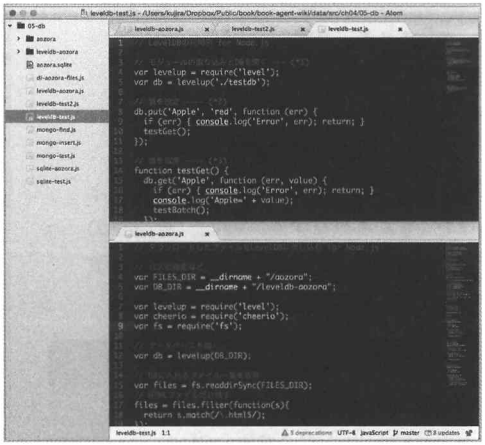
ここでは、JavaScript 開発を行う上で、生産性を向上させるのに役立つ機能を備えたモダンなエディターを紹介します。

Atom - 豊富なプラグインが魅力

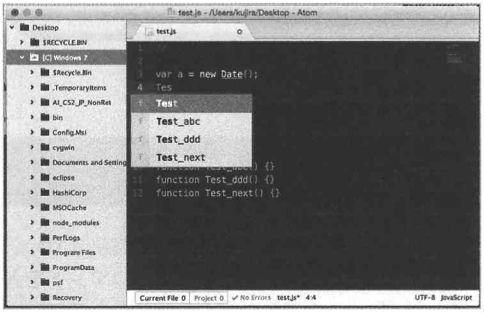
```
Atom エディター
[URL] https://atom.io/
[対応 OS] Windows / Mac OS X / Linux
[ライセンス] MIT License
```

「Atom」はオープンソースで開発されているテキストエディターです。主に GitHub により開発されています。Atom 最大のポイントは、Google Chrome の元となっている、Chromium をベースとしたエディター

であるということです。つまり、Web ブラウザーをベースにテキストエディターが作られているのです。さまざまなプログラミング言語のキーワード強調に対応しており、もちろん、JavaScript の構文もサポートしています。オートコンプリート機能もあるので入力も便利です。また、さまざまな拡張機能を追加することにより、多くの機能を加えることができます。



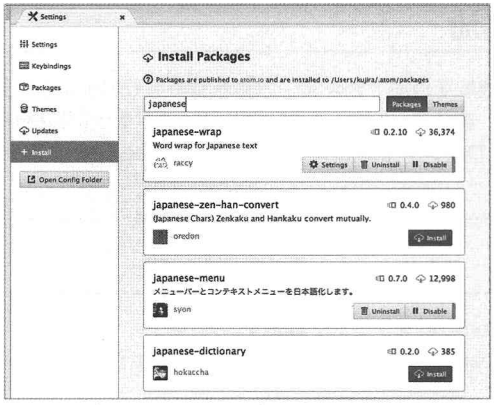
Atom エディター - 画面分割やタブ切り替え機能もある



Atom エディター - キーワード強調やコード補完機能もある

「Atom」にプラグインを追加

Atom のパッケージ（プラグイン）は非常に豊富で、任意のものを検索し、「Install」ボタンをクリックすることで手軽に追加できます。機能を追加するには、Windows ではメインメニューの「File > Settings > Install」をクリックします。Mac OS X では、Atom メニューの「Preference > Install」をクリックします。Atom は基本的に英語なのですが、「japanese-menu」をインストールすると、メニューを日本語化することができます。また、「file-icons」をインストールすると、ツリービュー内のアイコンをファイルの種類によって変えることができます。他にも、配色を変えるテーマもパッケージとして配布されており、気軽にテーマを変更することができます。



Atom エディターでパッケージを検索しているところ

Sublime Text - カスタマイズ性が高く恋に落ちると噂のエディター

```
Sublime Text
[URL] http://www.sublimetext.com/
[対応 OS] Windows / Mac OS X / Linux
[ライセンス]（有料）70 ドル
```

軽快でカスタマイズ性が高く人気があるのが「Sublime Text」です。JavaScript に限らず、多くのプログラミング言語をサポートしています。ブログの紹介記事を見ると、Web 制作や Ruby などの LL 言語 (軽快動作のスクリプト言語) 開発に多く使われているようです。オートコンプリートや各種プログラミング言語のキーワード強調の機能など、プログラミングにフォーカスしたテキスト編集機能が魅力です。任意のファイル、シンボル、行をすぐに開けるように工夫されています。70 ドルという有料のテキストエディターですが、保存のタイミングでライセンス購入のメッセージが表示される以外に機能制限がないので、心ゆくまで試すことができます。

Sublime Text の画面

「Sublime Text」にプラグインを追加

パッケージをインストールすることで、さまざまな機能を追加できます。原稿執筆時点(2015年6月)で3039ものパッケージが用意されています。とは言え、標準の「Sublime Text」には、パッケージの機能が用意されていません。以下のWebサイトを開いて、コマンドをコピーしてきて、Sublime Textのコンソールに打ち込む必要があります。

```
Package Control
https://packagecontrol.io/installation
```

コンソールを開くには、メニューから[View > Show Console]をクリックします。

例えば、Sublime Textを日本語化するパッケージ「Japanize」をインストールしてみましょう。パッケージのインストールは、次の手順で行います。

(1) **メニューから「Preferences > Package Control」をクリックします。**
(2) **「Package Control: Install Package」を入力し、Enterキーを押します。**
(3) **すると、パッケージのインストール画面がでるので、「Japanize」などパッケージ名を入力し、これをインストールします。**
(4) **必要に応じてエディターを起動し直します。**

Sublime Textでパッケージをインストール

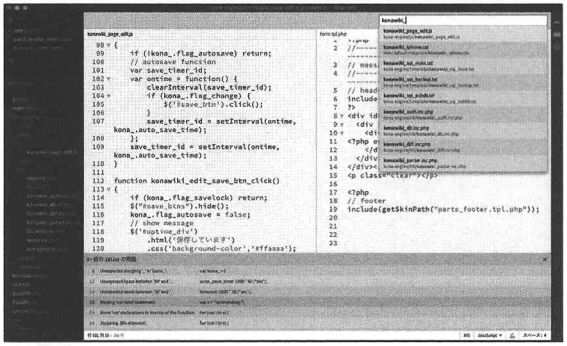
このように、Sublime Textには、多くのパッケージが用意されており、簡単なコマンドを入力することにより機能を拡張することができます。

Brackets - HTML編集に特化した光るエディター

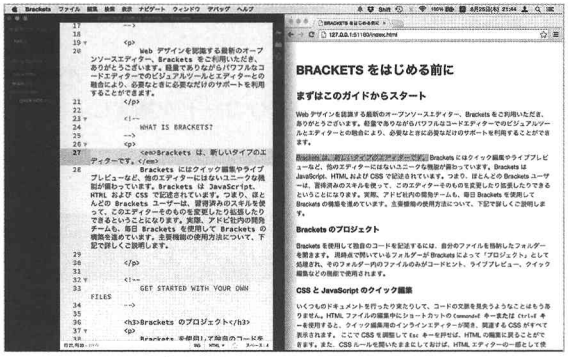
```
Brackets
[URL] http://brackets.io/
[対応OS] Windows / Mac OS X / Linux
[ライセンス] MIT License
```

BracketsはPhotoshopやIllustratorで有名なAdobeが主導で開発しているオープンソースのテキストエディターです。Web開発に特化したエディターとなっており、JavaScriptのキーワード表示、コード補完といった基本的な機能が備わっています。さらに、ライブプレビューの機能があります。これは、編集したHTMLファイルをすぐにブラウザーで確認できるだけなく、カーソルをHTMLのコードに移動させると、ブラウザー上でその部分が強調され、とても便利です。

また、Adobe Photoshop CC などで利用できる「Extract」機能を Brackets でも利用できるのも魅力の一つです。拡張機能マネージャーが備わっており、そこから配色テーマを変更したり、さまざまな機能を追加することができます。



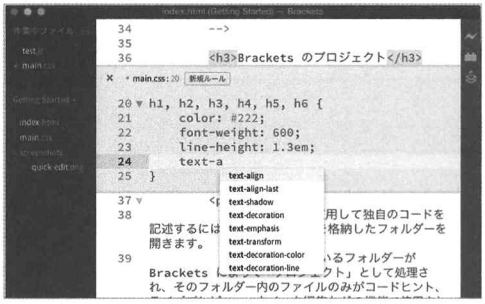
Brackets の画面



Brackets のライブプレビュー画面

クイック編集 - 便利なインラインエディター

本書では、JavaScript の開発に主眼を置いているので、本書を読む限りでは、それほど重宝しないものの、ここでは、Brackets の機能の中でもキラリと光る機能を紹介します。それが、インラインエディターです。<h3> などタグにカーソルを置いた状態で、メニューの [ナビゲート > クイック編集] をクリックしてみてください。すると、別ファイルで宣言している CSS の該当部分 (<h3> の定義) が別エディターとして表示されます。このポップアップした部分を編集すると、CSS ファイルの内容も変更されるというものです。ライブプレビューやクイック編集など、Brackets は、HTML 編集に特化したさまざまな機能が備わっているのが嬉しいところです。



Brackets のインラインエディターは便利

統合開発環境の JavaScript サポートに酔う

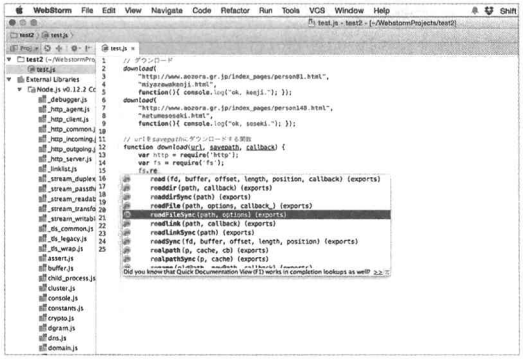
そのほかに、マシンパワーをガンガン使うものの、WebStorm や NetBeans、Eclipse などの統合開発環境を利用するという手もあります。実は、上記で紹介したテキストエディター以上に、コード補完機能が賢いのも統合開発環境の特徴です。

それというのも、JavaScript は動的型のプログラミング言語なので、なかなかコード補完が難しいのです。しかし、以下で紹介している統合開発環境の JavaScript サポートは、できるだけコードを補完しようと頑張ってくれます。

WebStorm - JavaScript 開発に特化した開発環境

```
WebStorm
[URL] https://www.jetbrains.com/Webstorm/
[対応 OS] Windows / Mac OS X / Linux
[ライセンス] 個人 49 米ドル / 商用 99 ドル（30 日間のお試し期間があり）
```

WebStorm は、統合開発環境の IntelliJ から Web 開発に特化した部分を取り出したもので、JavaScript の開発に注力した開発環境です。本書のプログラムでは、Node.js を使っていますが、Node.js を使った開発で役立つモジュールの補完機能を追加するような機能も用意されています。賢いコード補完やデバッガーなど便利な機能が用意されています。

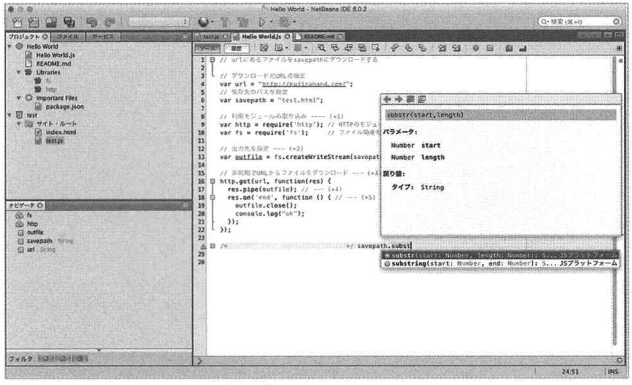


優れた補完機能を持つ WebStorm

NetBeans - Java で培ったノウハウを Web 開発でも使える

```
NetBeans
[URL] https://netbeans.org/
[ライセンス] CDDL,GPL2
```

統合開発環境「NetBeans」は、もともと Java の開発環境ですが、それ以外にも、C/C++ や PHP に特化した開発ツールとして利用できるようになっています。JavaScript の開発を行うには、NetBeans の PHP バージョンを選んでダウンロードすることになります。プラグインとして「Node.js」を追加すると、Node.js プロジェクトを作成することができます。



NetBeans で Node.js プロジェクトを作ったところ

この節のまとめ

- ここでは、JavaScript を開発する上で役立つモダンなエディターに焦点を当てて紹介しました。
- JavaScript の開発でこれらのエディターを選ぶ必要はまったくありません。
- しかし、最終的に普段から使っている便利なエディターに戻るとしても、普段から使っている自分のお気に入りの道具が一番便利だということが認識できればそれはそれでプラスでしょう。

第2章

Webデータの収集

いよいよ実際のプログラムを書いていきます。Webサイトには、たくさんの有益なデータが転がっています。ここでは、そうしたWebサイト上のデータをダウンロードする方法について紹介します。単にデータをダウンロードするだけでなく、HTMLファイルを解析する方法や、リンクを抽出する方法についても考えます。

01

Webページのダウンロード

この節では、Web ページを自分の PC などにダウンロードする方法を紹介します。単一ページをダウンロードするだけであれば、ブラウザーから保存すればいいので特別なツールは不要です。しかし、複数ページをダウンロードしたり、繰り返しダウンロードしたりする必要がある場合には、プログラムを作ると小回りが利きます。

ここで学ぶポイント	ツールやライブラリの一覧
●Webページのダウンロード方法	●Node.js ●Rhino/Nashorn

最も簡単なダウンロードの方法

単一の Web ページをダウンロードする一番簡単な方法は何でしょうか？　それは、Web ブラウザーで、今見ているそのページを自分の PC にダウンロードするという方法です。

具体的には Web ブラウザーのメニューから「名前を付けて保存」を選んでクリックするだけです。たいていの Web ブラウザーでは、そのページからリンクされている画像や CSS など、ページの構成要素を一気にダウンロードしてくれるでしょう（ブラウザーの仕様にもよります）。単一ページとはいっても、HTML ファイルだけを取得するのではなく、HTML ファイルからリンクされている画像や CSS などの情報を分析して逐次ダウンロードする必要があるからです。単一ページならば、あっという間に保存できます。しかし、これを自作プログラムでやろうとしたら結構大変です。

Node.js でダウンロードしてみよう

次に、プログラムを書いてファイルをダウンロードしてみましょう。

Node.js を利用してプログラムを作成します。Node.js に慣れてない方もいるかと思いますが、まずはプログラムのすべてを理解しようとするのではなく、その雰囲気を掴むことから始めてください。

それでは、ファイルをダウンロードするプログラムを見てみましょう。

● file: src/ch02/01-download/download-node.js

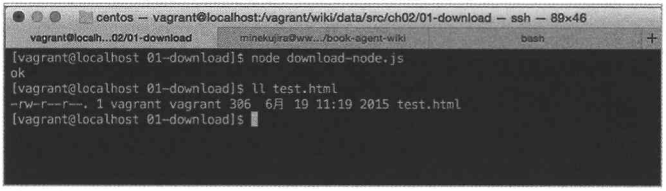
```
// urlにあるファイルをsavepathにダウンロードする

// ダウンロード元URLの指定
var url = "http://kujirahand.com/";
// 保存先のパスを指定
var savepath = "test.html";

// 利用モジュールの取り込み ---- (※1)
var http = require('http'); // HTTPのモジュール
var fs = require('fs');     // ファイル関連モジュール

// 出力先を指定 --- (※2)
var outfile = fs.createWriteStream(savepath);

// 非同期でURLからファイルをダウンロード --- (※3)
http.get(url, function(res) {
  res.pipe(outfile); // --- (※4)
  res.on('end', function () { // --- (※5)
    outfile.close();
    console.log("ok");
  });
});
```

1章では、仮想マシンにCentOSを準備して、Node.jsを実行する環境を整えました。SSHクライアントを開いて、CentOSにログインしたら、以下のコマンドを入力してプログラムを実行しましょう。WindowsにNode.jsをインストールした方は、コマンドプロンプト（Mac OS Xなら、ターミナル）を開きましょう。

cdコマンドを使ってプログラムを保存したディレクトリに移動します。その上で、以下のコマンドを実行すると、プログラムを実行できます。

```
$ node download-node.js
```

実行すると次のような結果になります。



プログラムを実行したところ

Node.jsのプログラムは非同期処理で書かれるため、少し読みづらいのですが、少しずつ慣れていきましょう。プログラムの各部分の処理を見ていきます。

まず、冒頭（※1）の部分ですが、プログラムの中で利用するモジュールをrequireを使って取り込んでいます。このようにNode.jsでは、requireを利用して、さまざまな処理を実現するモジュールを取り込んで利用することができます。このプログラムで利用しているのは、HTTPプロトコルに関わる機能を持つ「http」モジュールとファイル関連の機能を持つ「fs」モジュールです。

モジュールの取り込みを完了したら、fs.createWriteStream()メソッドで、出力先のファイルを指定します（※2）。さらに、http.get()メソッドで指定のURLにアクセスします（※3）。ただし、http.get()メソッドの返り値は取得した結果ではありません。

それというのも、Node.js では、時間のかかる処理を非同期で行うというスタイルを貫いています。つまり、その場で処理を完了するまで待つのではなく、処理が完了した際にコールバック関数で通知するようになっているのです。そのため、サーバーへリクエスト（要求）を出しただけで処理を戻します。返り値としては、リクエストに関する機能を持つ http.ClientRequest のインスタンスが返されます。

実行した結果は、http.get() メソッドの第 2 引数で指定したコールバック関数に返されます。そして、(※ 4) の部分で、ダウンロードしたデータをファイルに保存するよう指定を行います。とはいえ、ここでは保存の指定をするだけで、その時点で保存が完了するわけではありません。サーバーからのレスポンス（返答）を取得するのにも時間がかかるため、処理が完了したときに改めてコールバック関数で通知されます。それが、(※ 5) の部分で、ダウンロードが完了すると「end」に指定した関数が実行されます。

このように、少し大げさな感じもありますが、20 行程度のプログラムを書けば、ファイルのダウンロードを行うことができます。ダウンロードしたいファイルがたくさんあったり、定期的に繰り返しダウンロードしたりする際には、プログラムを書くと良いでしょう。

プログラムをブラッシュアップする

上記のプログラムは、このままだとちょっと使いづらいと思います。そこで、関数にして手軽にダウンロードできるようにしましょう。

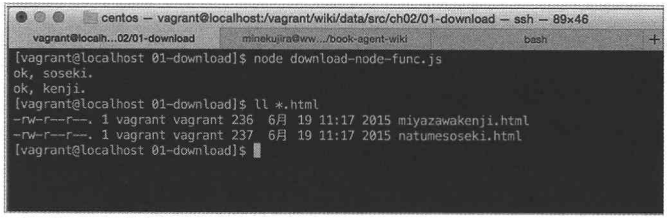
● file: src/ch02/01-download/download-node-func.js

```
// ダウンロード
download(
  "http://www.aozora.gr.jp/index_pages/person81.html",
  "miyazawakenji.html",
  function(){ console.log("ok, kenji."); });
download(
  "http://www.aozora.gr.jp/index_pages/person148.html",
  "natumesoseki.html",
  function(){ console.log("ok, soseki."); });

// url を savepath にダウンロードする関数
function download(url, savepath, callback) {
  var http = require('http');
  var fs = require('fs');
  var outfile = fs.createWriteStream(savepath);
  var req = http.get(url, function(res) {
    res.pipe(outfile);
    res.on('end', function () {
      outfile.close();
      callback();
    });
  });
}
```

プログラムを実行してみましょう。以下のコマンドを実行します。

```
$ node download-node-func.js
```

実行結果はこのようになります。



プログラムを実行したところ

ダウンロードを行っていた部分を関数で囲っただけなので、先ほどとそれほど変わるものではありません。Node.jsのコードはインデントが深くなりがちなので、格段に見やすいプログラムとまでは言えないかもしれません。

さて、このプログラムを実行して青空文庫からHTMLファイルをダウンロードしてみましょう。

すると、HTMLの本体ではなく「リダイレクトするように」との指示が書かれたHTMLファイルがダウンロードされてしまいました。

Webブラウザーであれば、自動でリダイレクトが行われるのですが、このNode.jsのプログラムでは行われません。

この点を改善するには、サーバーからのレスポンスコードを調べて、リダイレクトするようにプログラムを書き換える必要があります。ただし、ここでは余計なコードを足さず不完全なままにしておきます。

Rhino/Nashornでダウンロードしてみよう

それでは、参考までに、Javaで実装されたJavaScriptエンジンの「Rhino」(あるいは「Nashorn」)で動くプログラムを紹介します。同じJavaScriptのプログラムとはいえ、Node.jsのプログラムとはまた違ったものになっています(ちなみに、仮想マシンのCentOSでRhinoを動かすには、本家OracleのJavaランタイムをインストールした上で、yumでRhinoをインストールする必要があります)。

● file: src/ch02/01-download/download-rhino.js

```
// urlにあるファイルをsavepathにダウンロードする
var url = "http://kujirahand.com/";
var savepath = "test.html";

// ダウンロード
var aUrl = new java.net.URL(url);
var conn = aUrl.openConnection(); // URLに接続する      ┐--- (※1)
var ins = conn.getInputStream();  // 入力ストリームを得る ┘
var file = new java.io.File(savepath);            ┐ 出力ストリームを得る ---(※2)
var out = new java.io.FileOutputStream(file);     ┘
// 読み込んで書き込む ---- (※3)
var b;
while ((b = ins.read()) != -1) {
  out.write(b);
}
out.close(); // 出力ストリームを閉じる --- (※4)
ins.close(); // 入力ストリームを閉じる
```

プログラムを実行するには、以下のコマンドを実行します。変数urlの内容を、変数savepathに保存します。

```
> rhino download-rhino.js
```

Nashorn でも同じように実行できます。以下は、Nashorn を利用して実行する例です。

```
> jjs download-rhino.js
```

上記 Rhino(Nashorn) のプログラムのポイントですが、Java の API を利用しているという点です。Java の API をそのまま使えるのが Rhino や Nashorn の良い点です。

さて、Java のファイルやネットワークの API では、データをストリームとして扱います。ストリームとは、データを「流れるもの」として捉え、流れ込んでくるデータを入力、流れ出ていくデータを出力として扱います。つまり、今回のストリームの入り口は、Web サーバーから受信するデータです。ストリームの出口は、ローカル PC のファイルです。このように、ストリームの入り口と出口を用意したら、一気にデータを読み書きします。ここまでわかったらプログラムを確認してみましょう。

プログラム中 (※ 1) の部分では、Web サーバーに接続し、入力ストリームを得ます。つまり、サーバーから送信されたデータをストリームの入り口にセットします。(※ 2) の部分では、出力用のストリームを得ます。つまり、PC のローカルファイルにデータの出口をセットします。(※ 3) の部分では、入力したデータを出力ストリームに流し込みます。最後に (※ 4) の部分で各ストリームを閉じています。

この節のまとめ

- **JavaScript のプログラムを利用して、指定の URL をダウンロードする方法を紹介しました。**
- **はじめに、Node.js、つぎに Rhino/Nashorn の 2 つの JavaScript エンジンを利用してみました。**
- **エンジンによって、まったく異なる API が用意されており、プログラムの書き方も異なるということがわかりました。**
- **用途に応じて、どのエンジンを使うのか使い分けることになります。**

02 HTMLの解析(リンクと画像の抽出)

Web ページをダウンロードしたら、そのページを解析します。ダウンロードした HTML に記述されているさまざまな要素を調べます。特に、HTML の中に記述されているリンクや、画像ファイルなどは利用価値の高いものです。ここでは HTML のタグの解析方法を紹介します。

ここで学ぶポイント

- npmの使い方
- ダウンロードしたHTMLファイルの解析
- リンクや画像の抽出

ツールやライブラリの一覧

- Node.js と npm
- 「cheerio-httpcli」モジュール
- 「request」モジュール

スクレイピングしよう

Web の世界で俗にいう「スクレイピング」とは、Web サイトから Web ページの HTML データを収集して、特定のデータを抽出、整形し直すことです。単に Web サイトから HTML ファイルをダウンロードするだけでなく、その HTML ファイルの各内容を調べることを言います。

「cheerio-httpcli」モジュールをインストールしよう

Node.js を使って、スクレイピングをするのに便利なのが「cheerio-httpcli」というモジュールです。このモジュールを使うと、ファイルを手軽にダウンロードできるだけでなく、jQuery のように要素を取り出すことができるのです。また、Web ページの文字コードも自動判定して読み込んでくれます。ページ中のデータを取り出そうという時、手軽に指定した要素を取り出せる点は大きなメリットです。リンクや画像の抽出も簡単に実現できるからです。

どれほど簡単にできるかは、このあとじっくり紹介しますので、まずは、Node.js のパッケージマネージャーである「npm」でモジュールをインストールしてみましょう*。

```
$ npm install cheerio-httpcli
```

* モジュールをインストールしたのに、エラーが出てプログラムが正しく動かない場合には、1章の「Node.js モジュールのインストール」をご覧ください。

HTML ファイルをダウンロードしてみよう

それでは、「cheerio-httpcli」モジュールを使ってみましょう。まずは、「青空文庫にある著者『宮沢賢治』のページをダウンロードして、HTML 文書を画面に出力する」というプログラムを作ってみます。

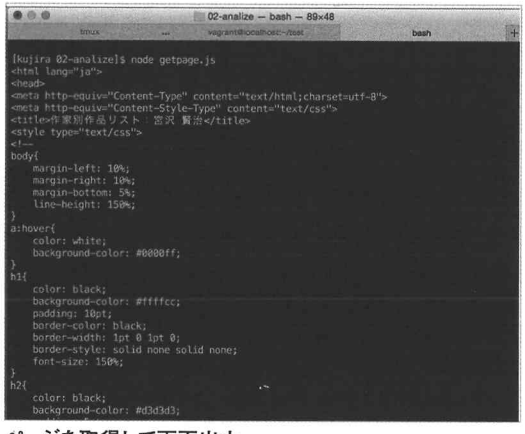
● file: src/ch02/02-analize/getpage.js

```
// モジュールの取り込み --- (※1)
var client = require('cheerio-httpcli');

// ダウンロード ---- (※2)
var url = "http://www.aozora.gr.jp/index_pages/person81.html";
var param = {};
client.fetch(url, param, function (err, $, res) {
  // エラーがないかチェック
  if (err) { console.log("Error:", err); return; }
  // ダウンロードした結果を画面に表示 ---- (※3)
  var body = $.html();
  console.log(body);
});
```

プログラムを実行するには、以下のコマンドを実行します。

```
$ node getpage.js
```

実行すると以下のような結果になります。



ページを取得して画面出力

前項の Node.js でファイルをダウンロードするプログラムよりも、ずいぶんわかりやすくなってると思います。fetch() メソッド一発で、ダウンロードからページの解析までが完了してしまうからです。

プログラムの (※1) に注目してみましょう。Node.js でモジュールを利用するためには、require() でモジュールを取り込む必要があります。また、取り込んだモジュールの名前には、どんな名前をつけてもよく、任意の変数にモジュールの持つ機能を与えることができます。ここでも、モジュール「cheerio-httpcli」の機能を取り込んでいますが、client という名前の変数にモジュールの機能を代入しています。

ダウンロード（正確には、Web サイトから HTML を取得）している部分が、プログラムの (※2) の部分です。fetch() メソッドを利用します。このメソッドに与える引数は、URL、パラメーター、コールバック関数の順で指定します。このコールバック関数は、Web サイトからデータを取得した後で実行されるもので

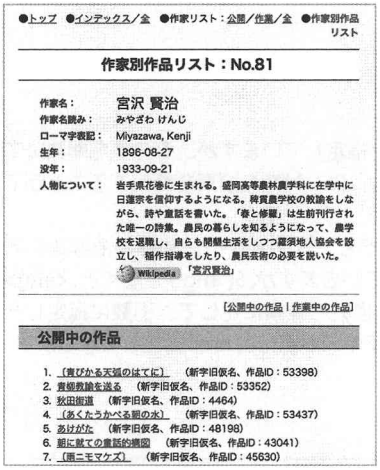
す。コールバック関数の引数では、エラー情報(err)、取得したデータ($)、サーバーからのレスポンス情報(res)という3つの情報が与えられます。

プログラムの(※3)の部分では、取得したデータが得られる$変数のhtml()メソッドを呼び出すことで、HTMLを取得して画面に表示しています。

HTMLファイルのリンクを抽出してみよう

「cheerio-httpcli」モジュールの特徴として、WebからHTMLを取得した後、内容を解析して、CSSのセレクタで任意の要素を検索できるという機能があります。そこで、「cheerio-httpcli」モジュールでHTMLからリンクされているURLの一覧を表示するという簡単なプログラムを作ってみます。

以下の画面は、青空文庫にある著者「宮沢賢治」のページです。このページから、宮沢賢治の作品が多数リンクされています。そこで、このページから、リンクされている作品のタイトルとURLを取り出してみたいと思います。



宮沢賢治の作品一覧ページ - ここからリンクを抽出します

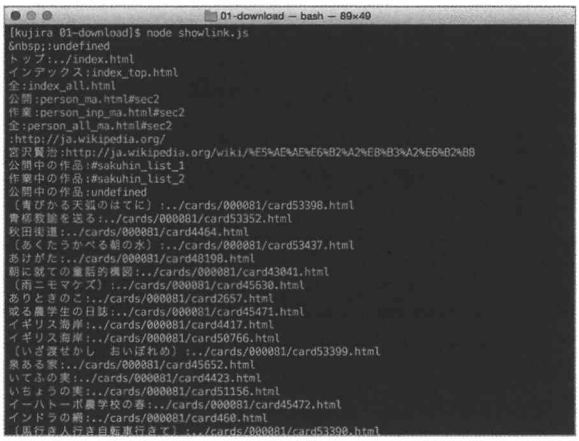
それでは、先のプログラムを書き換えて作品リスト取得のプログラムを作ってみましょう。

● file: src/ch02/02-analize/showlink.js

```
// モジュールの読込
var client = require('cheerio-httpcli');

// ダウンロード
var url = "http://www.aozora.gr.jp/index_pages/person81.html";
var param = {};
client.fetch(url, param, function(err, $, res) { //----(※1)
  if (err) { console.log("error"); return; }
  // リンクを抽出して表示 --- (※2)
  $("a").each(function(idx) {
    var text = $(this).text();
    var href = $(this).attr('href');
    console.log(text+":"+href);
  });
});
```

このプログラムを実行するには、コマンドラインから以下のコマンドを実行します。

```
> node showlink.js
```



プログラムを実行してリンク一覧を取得したところ

プログラム (※ 1) の部分で、fetch() メソッドでコールバック関数を指定していますが、その無名関数にて指定している第二引数を $ としています。これによって、jQuery のように、特定の要素を抽出することができるようになります。

プログラム (※ 2) の部分を見れば、jQuery を知らないという方でも、雰囲気がつかめるのではないでしょうか。$("a") と書くと、<a> タグを抽出してくれます。この後見ていきますが、$("img") と書くと、<img> タグを抽出することができます。続く each() メソッドは、抽出した複数の要素に対して、引数に指定したコールバック関数を実行します。つまり、抽出した各要素についてテキスト部分と href 属性を表示します。

相対 URL を絶対 URL に変換しよう

先のプログラムで宮沢賢治の著者ページからリンクを取り出しました。ここで取り出した URL (<a> タグの「href」属性) をよく見てみましょう。以下は、実行結果からの抜粋です。

```
...
〔雨ニモマケズ〕:../cards/000081/card45630.html
ありときのこ:../cards/000081/card2657.html
或る農学生の日誌:../cards/000081/card45471.html
イギリス海岸:../cards/000081/card4417.html
...
```

多くの href 属性は「./test.html」や「../hoge.PNG」のように、相対パスで URL が書かれていることに気がつくかと思います。相対パスは、そのページを起点とした場合の関係を表しており、Web ページを作る際には記述量が減って都合が良いものです。

しかし、プログラムでダウンロードを行う際には都合が悪く、相対パスの記述を、絶対パスに直す必要があります。そこで、Node.js で相対パスの URL を絶対パスの URL に変換する方法を紹介します。

Node.js で URL のパスを操作するには、「url」モジュールを利用します。このモジュールは標準モジュールです。相対パスを絶対パスに変換するには、url.resolve() メソッドを使います。引数に、基本 URL と相対 URL を指定することでパスを解決し、絶対 URL に変換できます。

```
[書式] 相対パスを絶対パスに変換する方法
var URL = require('url');
var res = URL.resolve(基本URL, 相対URL);
```

それでは、簡単な使い方を見てみましょう。

● file: src/ch02/02-analize/url-test.js

```
// urlモジュールを読み込む
var URL = require('url');

// 相対パスを絶対パスに変換
var base = "http://kujirahand.com/url/test/index.html";
var u1 = URL.resolve(base, 'a.html');
console.log("u1 = " + u1);
var u2 = URL.resolve(base, '../b.html');
console.log("u2 = " + u2);
var u3 = URL.resolve(base, '/c.html');
console.log("u3 = " + u3);
```

上記のプログラムを実行すると、次のような絶対パスに変換された URL が出力されます。

```
$ node url-test.js
u1 = http://kujirahand.com/url/test/a.html
u2 = http://kujirahand.com/url/b.html
u3 = http://kujirahand.com/c.html
```

これを踏まえて、resolve() メソッドを使って、先ほどの showlink.js を修正してみます。宮沢賢治の著者ページからリンクを取り出しますが、相対パスではなく、すべて絶対パスに変換した上で画面に出力します。

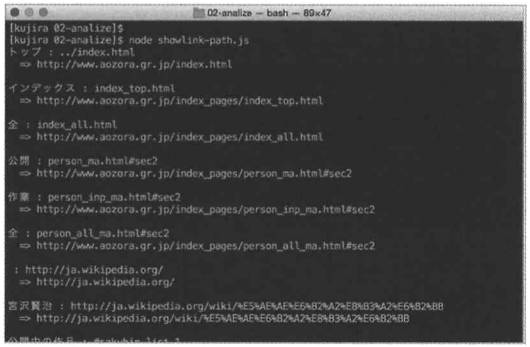
● file: src/ch02/02-analize/showlink-path.js

```
// モジュールの読込
var client = require('cheerio-httpcli');
var URL = require('url');

// URLとパラメーター
var url = "http://www.aozora.gr.jp/index_pages/person81.html";
var param = {};

// ダウンロード
client.fetch(url, param, function(err, $, res) {
  if (err) { console.log("error"); return; }
  // リンクを抽出して表示
  $("a").each(function(idx) {
    var text = $(this).text();
    var href = $(this).attr('href');
    if (!href) return;
    // 相対パスを絶対パスに直す--- (※1)
    var href2 = URL.resolve(url, href);
    // 結果を表示
    console.log(text + " : " + href);
    console.log("  => " + href2 + "\n");
  });
});
```

プログラムを実行するには、以下のコマンドを入力します。

```
$ node showlink-path.js
```

実行すると結果は以下のようになります。



プログラムを実行したところ

プログラムは、先のプログラムとほとんど同じです。異なるのは、プログラムの(※1)の部分です。resolve() メソッドを利用して、相対パスを絶対パスに変換しています。

画像ファイルを抽出してみよう

リンク一覧を抽出をしたところで、次に、画像ファイルの URL 一覧を抽出するプログラムを作ってみましょう。ここでは、Wikipedia の犬に関するページから画像をダウンロードするというプログラムを作ってみます。

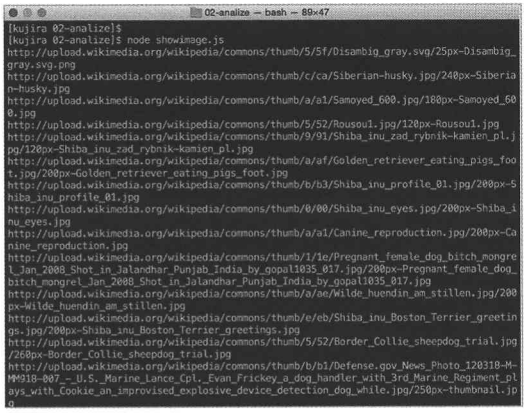
Wikipedia の犬のページ。画像が何枚も入っています

このように、Wikipedia の犬のページには複数の画像があります。このページに貼られている画像をダウンロードするプログラムを作ってみましょう。

まずは、<img> タグを取り出し、src 属性で指定している画像ファイルの一覧を表示するものを作ってみます。このプログラムも、先ほどのプログラムをちょっと書き換えれば作れます。

● file: src/ch02/02-analize/showimage.js

```
// モジュールの読込
var client = require('cheerio-httpcli');
var URL = require('url');

// ダウンロード
var url = "http://ja.wikipedia.org/wiki/イヌ";
var param = {};
client.fetch(url, param, function(err, $, res) {
  if (err) { console.log("error"); return; }
  // リンクを抽出して表示
  $("img").each(function(idx) {
    var src = $(this).attr('src');
    src = URL.resolve(url, src);
    console.log(src);
  });
});
```



img タグの一覧を抽出したところ

先ほどのプログラムの <a> タグを <img> タグに変えただけですので、わかりやすいですよね。ただし、画像ファイルの URL を取得しただけでは意味がありません。ダウンロードしてこそ役立つものでしょう。

では、次にプログラムを改造して、画像をダウンロードするようにしてみたいと思います。

request モジュールを使ってみよう

さて、ここで、画像のダウンロードを行うのに際して、便利なモジュールを紹介します。「request」モジュールです。これを使うと、より簡潔にダウンロード処理を記述することができます。また、リダイレクトの指示があれば、自動でリダイレクト先のファイルをダウンロードします。

npm を使って、request モジュールをインストールしてみましょう。

```
> npm install request
```

インストールができたら、requestモジュールの使い方を簡単に見てみましょう。以下のプログラムは、requestモジュールを利用して、指定URLのファイルをローカルにダウンロードするプログラムです。

● file: src/ch02/02-analize/download-node-request.js

```
// モジュールの読み込み
var request = require('request');
var fs = require('fs');

// URLの指定
var url = "http://kujirahand.com/";
var savepath = "test.html";

// ダウンロード
request(url).pipe(fs.createWriteStream(savepath));
```

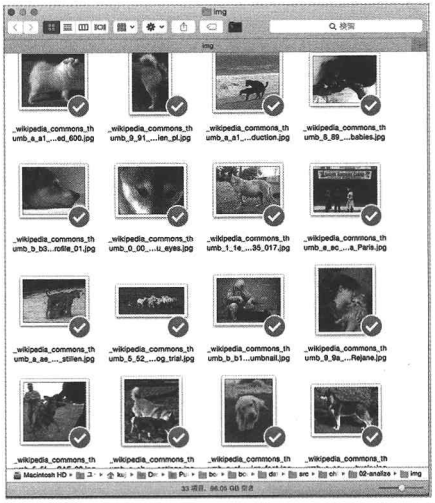
モジュールの読込やURLの指定がありますが、ダウンロードを実行しているのは、最終行のわずか一行です。このように、requestモジュールを使うと非常に手軽にダウンロードが実現できます。

リンクされている画像をすべてダウンロードする

次に、HTMLからリンクされている画像をすべてダウンロードするプログラムを作ってみましょう。以下のプログラムは、Wikipediaの犬に関するページから <img> タグを取り出し、画像ファイルの一覧をimgフォルダにダウンロードします。

● file: src/ch02/02-analize/dl-image.js

```
// モジュールの読込
var client = require('cheerio-httpcli');
var request = require('request');
var fs = require('fs');
var URL = require('url');

// ダウンロード先のディレクトリを作る
var savedir = __dirname + "/img"; // --- (※1)
if (!fs.existsSync(savedir)) {    // --- (※2)
  fs.mkdirSync(savedir);          // --- (※3)
}

// HTMLファイルの指定
var url = "http://ja.wikipedia.org/wiki/イヌ";
var param = {};
// HTMLファイルの取得 --- (※4)
client.fetch(url, param, function(err, $, res) {
  if (err) { console.log("error"); return; }
  // リンクを抽出して表示 --- (※5)
  $("img").each(function(idx) {
    var src = $(this).attr('src');
    // 相対パスを絶対パスに変更 --- (※6)
    src = URL.resolve(url, src);
    // 保存用のファイル名を作成 --- (※7)
    var fname = URL.parse(src).pathname;
    fname = savedir + "/" + fname.replace(/[^a-zA-Z0-9\.]+/g, '_');
    // ダウンロード --- (※8)
    request(src).pipe(fs.createWriteStream(fname));
  });
});
```



ページからリンクされている画像ファイルを全部ダウンロードしたところ

このプログラムにも、いくつかポイントがあります。まず、Wikipediaの犬のページには、たくさんの画像ファイルがあるので、imgディレクトリを作り、そこに画像をダウンロードすることにしました。そのために、fs.mkdirSync()メソッドを使ってディレクトリを作成しています。

プログラムの(※1)から(※3)では、imgディレクトリが存在しなければ作成するという処理になっています。Node.jsでは、スクリプトの実行ディレクトリが「__dirname」変数に設定されています(※1)。この変数を参照することで、手軽に、保存先のパスを指定することができます。そして、(※2)の部分のfs.existsSync()メソッドを使うことで、ファイルやディレクトリが存在するのかどうか調べることができます。ディレクトリが存在しないとき(※3)にはディレクトリを作成します。

ちなみに、fs.exists()やfs.mkdir()といったメソッドが存在するのですが、これは、非同期で実行されるものとなっています。つまり、fs.mkdir()メソッドでは、ファイルやディレクトリの作成が完了した時点で、第二引数に指定したコールバック関数が呼ばれるという仕組みです。fs.mkdirSync()メソッドは名前にあるとおり、ディレクトリの作成が完了するまでスクリプトの実行を待機します。

あとは、先ほどの画像の取得と同じです。(※4)の部分でHTMLファイルを取得し、(※5)の部分で<img>タグの一覧を取得します。そして、取り出した各要素に対して、ダウンロード処理を実行します。(※6)の部分では、相対パスを絶対パスに変換しています。(※7)の部分では、まず、ファイル保存用にURLからパス名を取り出します。それから、それをアルファベットと数字、ドット以外の文字をアンダーバー「_」に変換します。最後に(※8)の部分で、requestモジュールを利用してダウンロードを行います。

この節のまとめ

- **Node.jsのモジュールである「cheerio-httpcli」と「request」を利用して、HTMLファイルの解析やダウンロードを行うプログラムを作ってみました。**
- **Node.jsには、豊富なモジュールが用意されているので、これを利用することで、さまざまな処理を行うことができます。**
- **手軽にHTMLファイルを解析し、必要な情報を取り出す「cheerio-httpcli」モジュールを使うと、任意のタグを手軽に抽出できるのが便利な点でした。**

03 サイトを丸ごとダウンロード

前節では、HTMLファイルを解析し、リンクされている画像をダウンロードするプログラムを作りました。ここでは、プログラムをさらに改良して、サイトを丸ごとダウンロードするプログラムを作ってみます。

ここで学ぶポイント

- **Webサイトの丸ごとダウンロード**
- **再帰関数**

ツールやライブラリの一覧

- **Node.js**
- **「cheerio-httpcli」モジュール**

丸ごとダウンロードする利点とは

さて、サイトを丸ごとダウンロードすると何が便利でしょうか。それは、サイトがダウンしていたり、自分がネットにつながっていないオフラインの状態であっても、そのサイトの情報を見ることができることです。

ローカルにHTMLをダウンロードしておけば、PCローカルで細かくファイル検索にかけることも容易でしょう。また、期間限定しか掲載されない情報もあるので、消える前にサイトを丸ごとダウンロードしておきたいというニーズもあるでしょう。さらに面白い使い方としては、サイト全体をデータベースなどに保存しておいて、後でいろいろな解析処理を行うということも考えられます。

リンクをたどってダウンロードしていく

丸ごとダウンロードといっても、自分が管理していないWebサーバーに対してFTPなどでWebサーバーに接続してファイルをごっそりダウンロードすることはできません。そこで、トップページからリンクされているファイルを頼りに各HTMLを巡回して、一ページずつファイルをダウンロードしていく手順を採ることになります。

プログラムを作ってみよう

まずは3階層先までリンクしているHTMLファイルを調べて、ローカルにHTMLファイルをダウンロードするというプログラムを作ってみます。このとき、リンクされている外部ページはダウンロードしないように工夫し、指定サイト内のページだけをダウンロードするようにします。

ここでは、Node.jsの日本語マニュアルを一気にローカルにダウンロードするというプログラムを作ってみました。プログラムの冒頭（モジュール取り込み処理の下の部分）に共通の設定が書いてあります。この中のLINK_LEVELで何階層先のリンクまで調べるのかを設定できます。3を指定しているので、3階層先のHTMLファイルまで取得します。

まずは、プログラム全体を見てみましょう。プログラム中で主な処理は、downloadRec()関数です。この関数がHTMLファイルを取得し、リンクを抽出して、リンク先のHTMLを取得しています。

● file: src/ch02/03-getall/getall.js

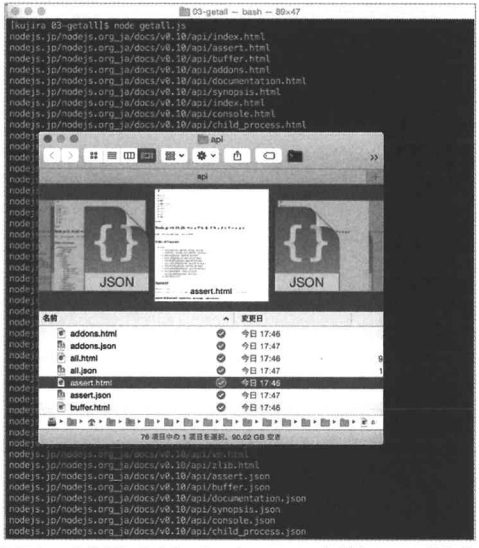
```
// リンクを解析してダウンロード for Node.js
// --- モジュールの取り込み ---
var client = require('cheerio-httpcli');
var request = require('request');
var URL = require('url');
var fs = require('fs');
var path = require('path');

// --- 共通の設定 ---
// 階層の指定
var LINK_LEVEL = 3;
// 基準となるページURL
var TARGET_URL = "http://nodejs.jp/nodejs.org_ja/docs/v0.10/api/";
var list = {};

// メイン処理
downloadRec(TARGET_URL, 0);

// 指定のurlを最大レベルlevelまでダウンロード
function downloadRec(url, level) {
  // 最大レベルチェック
  if (level >= LINK_LEVEL) return;
  // 既出のサイトは無視する
  if (list[url]) return;
  list[url] = true;
  // 基準ページ以外なら無視する
  var us = TARGET_URL.split("/");
  us.pop();
  var base = us.join("/");
  if (url.indexOf(base) < 0) return;
  // HTMLを取得する
  client.fetch(url, {}, function(err, $, res) {
    // リンクされているページを取得
    $("a").each(function(idx) {
      // <a>タグのリンク先を得る
      var href = $(this).attr('href');
      if (!href) return;
      // 絶対パスを相対パスに変更
      href = URL.resolve(url, href);
      // '#' 以降を無視する(a.html#aa と a.html#bb は同じもの)
      href = href.replace(/\#.+$/, ""); // 末尾の#を消す
      downloadRec(href, level + 1);
    });
    // ページを保存(ファイル名を決定する)
    if (url.substr(url.length-1, 1) == '/') {
      url += "index.html"; // インデックスを自動追加
    }
    var savepath = url.split("/").slice(2).join("/");
    checkSaveDir(savepath);
    console.log(savepath);
    fs.writeFileSync(savepath, $.html());
```

```
  });
}

// 保存先のディレクトリが存在するか確認
function checkSaveDir(fname) {
  // ディレクトリ部分だけ取り出す
  var dir = path.dirname(fname);
  // ディレクトリを再帰的に作成する
  var dirlist = dir.split("/");
  var p = "";
  for (var i in dirlist) {
    p += dirlist[i] + "/";
    if (!fs.existsSync(p)) {
      fs.mkdirSync(p);
    }
  }
}
```

プログラムを実行するには、下記のコマンドを実行します。

```
$ node getall.js
```

実行すると、次のようにファイル一覧をダウンロードします。

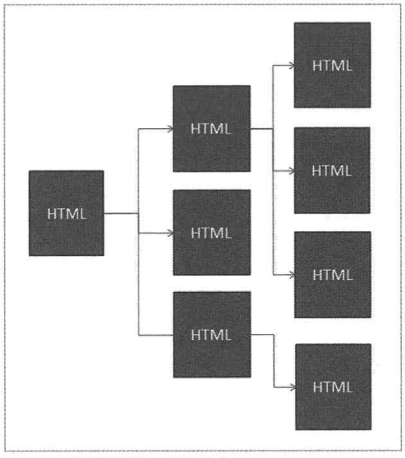


HTML からリンクされている HTML をダウンロードしたところ

では、少し詳しく処理を見ていきましょう。HTML を取得し、そこからリンクされている HTML を取得し、さらにそこからリンクされている HTML を取得して…と、再帰的に HTML ファイルを取得していくことになります。つまり、再帰処理を記述すれば良いのです。

再帰処理とは？

再帰処理というのは、プログラミング技法の一つで、ある関数の中で、その関数自身を呼び出すことを言います。例えば、関数 A があるなら、関数 A の中で、関数 A を呼び出すことを言います。では、再帰処理が何の役に立つのでしょうか。今回のように、HTML からリンクされている HTML、そこからリンクされている HTML・・・。という風に、HTML の解析処理から、さらに HTML の解析処理を呼ぶ場合などに都合が良いのです。



HTML を再帰的に解析する

再帰的な呼び出しを英語では、Recursive call と言います。再帰呼び出しを行う関数には、「_r」や「_rec」と言った接尾辞を付けることが多くあります。今回のプログラムでも、再帰的に HTML ファイルを解析し取得する関数を「downloadRec()」という関数名にしています。

さて、上に挙げたソースコードが少し長いので、動きがわかるように、この関数の再帰処理の様子を箇条書きにしてみましょう。

- **最大レベルのチェック、レベルを超えていれば関数を抜ける**
- **既出の URL であれば、関数を抜ける**
- **指定 URL の HTML を取得する**
- **HTML から <a> タグを抜き出して、以下を繰り返す**
- **リンク先情報 (href 属性) を取り出して絶対パスに変換**
- **href 属性の値を引数に、downloadRec() を呼び出す**
- **HTML をローカルに保存**

関数 downloadRec() の内容 :

こうしてみると、それほど難しい処理をしているわけではないことがわかります。とは言え、再帰処理を書くときに注意すべき点も見えてきます。

再帰処理では、関数の中でその関数自身を呼び出します。そのため、気をつけないと、永遠に関数を呼び出し続けてしまうのです。そこで、関数の呼び出し最大レベルをチェックし、最大レベルを超えていたら、すぐに関数を抜けるという処理を行っています。加えて、同じ URL を何度も取得し解析するのは無駄なので、重複チェックも行っています。

相対パスを絶対パスに変換する

前節でも紹介しましたが、HTML からリンクされているファイルは、一般的に相対パスで指定が行われています。そのため、リンクである <a> タグからリンク先を取り出したときには、相対パスから絶対パスへの変換処理を行います。今回のプログラムでも、Node.js の url モジュールを利用して変換しています。

Node.js の同期と非同期関数について

Node.js のネットワークやファイル処理などの関数は、非同期で実装されています。その関数を呼び出した時点では、処理は完了していませんが、完了した時点でコールバック関数が呼び出されるというものです。

例えば、フォルダを作成する、fs.mkdir() 関数を例にとって使い方を確認してみましょう。書式は下記の通りです。

```
[書式]
fs.mkdir(path, function() {
  // ここにフォルダ作成した後の処理
});
```

利用例は次のようになります。

● file: src/ch02/03-getall/mkdir.js

```
// モジュールの取り込み
var fs = require('fs');

// フォルダを作成する
console.log("mkdir 実行します。");
fs.mkdir("test", function () {
  console.log("フォルダ作成完了しました。");
});
console.log("mkdir 実行しました。結果待ち。");
```

このプログラムを実行すると、test というフォルダが作成されますが、プログラムを実行した結果は下記のようになります。

```
$ node mkdir.js
mkdir 実行します。
mkdir 実行しました。結果待ち。
フォルダ作成完了しました。
```

このように、fs.mkdir() を実行しても、その時点でフォルダは作成されず、すぐに次の処理が実行されます。そして、フォルダの作成が完了すると、引数に指定したコールバック関数が実行されるという訳です。

しかし、これでは不便な場合もあります。例えば、フォルダ A、A/B、A/B/C と、次々とフォルダを作成したい場合です。

そもそも、フォルダをちょっと作成するだけで、非同期処理を書くのは面倒です。そこで、Node.jsでは、同期的に処理を行う関数を用意しています。同期的に処理を行う関数には、「***Sync()」という接尾辞がついています。

Node.jsのマニュアルで、fs.mkdir()を調べると、そのすぐ下に、fs.mkdirSync()が書かれていますね。通常（非同期処理）のメソッドに加えて、同期的にフォルダを作成するメソッドが用意されていることがわかります。以下、fs.mkdirSync()関数を使ったものですが、やはり、バッチ処理を記述する場合には、こちらの方が気軽に使えます。

● file: src/ch02/03-getall/mkdirSync.js

```
// モジュールの取り込み
var fs = require('fs');

// フォルダを同期的に作成する
console.log("mkdir 実行します。");
fs.mkdirSync("test-sync");
console.log("mkdir を完了しました。");
```

なお、fs.mkdir()およびfs.mkdirSync()メソッドは、すでに存在するフォルダに対してさらに作成しようとするとエラーになります。そのため、通常は、フォルダが既に作成されているかどうかを確認してから作成します。fs.existsSync()メソッドを使うとファイルやフォルダが存在するかどうかを調べることができます。

● file: src/ch02/03-getall/mkdir3.js

```
// モジュールの取り込み
var fs = require('fs');

// フォルダを同期的に作成する
if (!fs.existsSync("test3")) {
  fs.mkdirSync("test3");
  console.log("test3 を作成しました ");
} else {
  console.log("test3 は既にあるので作成しません。");
}
```

```
$ node mkdir3.js
test3 を作成しました
$ node mkdir3.js
test3 は既にあるので作成しません。
```

この節のまとめ

- この節では、サイトを丸ごとダウンロードするプログラムを紹介しました。
- 再帰関数を利用することで、HTMLからリンクされているHTML、そこからリンクされているHTMLと、それぞれ順にダウンロードするプログラムを作ることができます。

04

XML/RSSの解析

最近では、多くのWebサイトが有益な情報をXMLやRSSの形式で提供しています。XMLやRSSで提供される情報は、HTMLと違ってデザイン情報を含んでいないため解析も容易です。ここでは、XMLやRSSの解析方法について紹介します。

ここで学ぶポイント

- XML/RSSのデータ形式について
- XML/RSSのダウンロード

ツールやライブラリの一覧

- Node.js
- 「xml2js」モジュール
- 「cheerio-httpcli」モジュール

XMLとは？

XMLというのは、Extensible Markup Languageの略で、個別の目的に応じた汎用的なデータ形式を言います。基本的にはテキストデータなのですが、個々のデータをタグ付することで、文書やデータを構造化することができます。XMLの目的は、異なる情報システム間で、構造化された文書やデータを容易に共有することです。

XMLは汎用的な形式なので、XMLを基にしたさまざまなデータ形式が存在しています。例えば、ニュースサイトの要約情報である「RSS」や、ベクターグラフィックスを扱う「SVG」なども、XMLを基にしてします。また、Excel/Wordなど、Microsoft Officeの保存データ形式も、複数のXMLファイルをZIP圧縮したものであるなど、さまざまなところでXMLが利用されています。

なんと言っても、XMLは機械にとっても人間にとっても読みやすいデータ形式として設計されている点が大きいでしょう。オープンな仕様であり、XMLを処理するために多くのツールやライブラリが備わっているのも魅力の一つです。

XML/RSSについては、本書の４章３節「データ形式の基礎知識」でも改めて扱います。

XMLの構造を確認しよう

XMLの基本構造は、要素 (element) と属性 (attribute) です。以下、基本的な要素の例です。

```
< 要素名 属性 =" 値 "> 内容 </ 要素名 >
```

そして、要素は、内部に子要素を持つことができます。例えば、商品カタログを表すXMLデータは、下記のようになります。

```
<商品カタログ>
  <商品 id="S001">
    <商品名>8GB SDカード</商品名>
    <値段>4500円</値段>
  </商品>
  <商品 id="S002">
    <商品名>USBマウス</商品名>
    <値段>2300円</値段>
  </商品>
  <商品 id="S003">
    <商品名>USBキーボード</商品名>
    <値段>3700円</値段>
  </商品>
</商品カタログ>
```

このように、XMLは木構造で、データの構造を表現することができます。

加えて、XMLデータをファイルに保存する場合には、XML宣言を含めることができます。このとき、そのXMLデータの文字符号化方式(文字コード)を宣言します。

```
<?xml version="1.0" encoding="UTF-8"?>
<商品カタログ>
...
</商品カタログ>
```

Node.jsでXMLを扱う方法

では、Node.jsでXMLを扱う方法を紹介します。Node.jsでは、先ほど利用した「cheerio-httpcli」モジュールもXML解析に使うこともできますし、他にもいろいろなモジュールが利用できます。そこで、ここでは「xml2js」モジュールを紹介してみます。このモジュールを使うと、XMLデータをJavaScriptのオブジェクトに変換することができます。

「xml2js」をインストールするには、npmを使って下記のコマンドを実行します。

```
$ npm install xml2js
```

モジュールの簡単な使い方を紹介しましょう。まずは、requireでモジュールのparseString()メソッドを取り出します。続いて、parseString()の第一引数に、文字列のXMLデータを指定します。結果は、第二引数に与えたコールバック関数で得られます。

以下は、簡単なXMLをJavaScriptのオブジェクトに変換する例です。JavaScriptのオブジェクトの内容を確認するために、JSON.stringify()メソッドを利用しています。

● file: src/ch02/04-xmlrss/test-xml.js

```
// モジュールの取り込み
var parseString = require('xml2js').parseString;

// テスト用のXMLデータ
var xml = "<fruits shop='AAA'>" +
  "<item price='140'>Banana</item>" +
  "<item price='200'>Apple</item>" +
  "</fruits>";
```

```
// XMLをパースする
parseString(xml, function (err, result) {
  // パース完了したときの処理をここに記述
  console.log(JSON.stringify(result));
});
```

プログラムを実行するには以下のコマンドを実行します。

```
$ node test-xml.js
```

実行結果は、以下のようになります（結果にわかりやすく改行を入れています）。

```
{
  "fruits": {
    "$":{"shop":"AAA"},
    "item":[
      {"_":"Banana","$":{"price":"140"}},
      {"_":"Apple","$":{"price":"200"}}
    ]
  }
}
```

この結果を見て、「おや？」と首をかしげた方もいるでしょうか。XMLに含まれない「$」や「_」というキーが存在します。そもそも、XMLには、要素の「内容」と「属性」があります。そのため、素直にXMLとJSONが一対一にならないのです。ちょっと面倒ですが、要素の内容は"_"というキーで提供され、属性は"$"というオブジェクトで与えられるのです。

それでは、もう少し、具体的な使い方を見てみます。以下は、XMLの<fruits>のshop属性を表示した後、各<item>タグの値を出力するというものです。

● file: src/ch02/04-xmlrss/test-xml2.js

```
// モジュールの取り込み
var parseString = require('xml2js').parseString;

// テスト用のXMLデータ
var xml = "<fruits shop='AAA'>" +
  "<item price='140'>Banana</item>" +
  "<item price='200'>Apple</item>" +
  "</fruits>";

// XMLをパースする
parseString(xml, function (err, result) {
  // console.log(JSON.stringify(result)); // --- (※1)
  // フルーツを提供するお店の名前
  var shop = result.fruits.$.shop;
  console.log("shop=" + shop);
  // フルーツの名前と値段を表示
  var items = result.fruits.item;
  for (var i in items) {
    var item = items[i];
    console.log("-- name=" + item._);
    console.log("   price=" + item.$.price);
  }
});
```

プログラムを実行するには以下のコマンドを実行します。

```
$ node test-xml2.js
```

実行結果は下記のようになります。

```
shop=AAA
-- name=Banana
   price=140
-- name=Apple
   price=200
```

プログラムの動作がわかりにくい場合は、プログラム中 (※ 1) のコメントを外して、改めて XML の変換結果を確認してみてください。

ただし、以下のプログラムを実行してみるとわかりますが、要素に子要素が含まれない場合、要素名がそのまま値となります。

● file: src/ch02/04-xmlrss/test-xml0.js

```
// モジュールの取り込み
var parseString = require('xml2js').parseString;

// テスト用の XML データ
var xml = "<item>Banana</item>";

// XML をパースする
parseString(xml, function (err, result) {
  console.log(result.item); // 結果 : Banana
});
```

そのため、次のような XML データであれば、"_" や "$" を含まないオブジェクトに変換されます。このようなデータであれば、XML と JavaScript がほぼ一対一で変換されます。

● file: src/ch02/04-xmlrss/test-xml3.js

```
// モジュールの取り込み
var parseString = require('xml2js').parseString;

// テスト用の XML データ
var xml =
  "<items>" +
  "<item><name>Banana</name><price>130</price></item>" +
  "<item><name>Apple</name><price>300</price></item>" +
  "<item><name>Pear</name><price>250</price></item>" +
  "</items>";

// XML をパースする
parseString(xml, function (err, r) {
  console.log(JSON.stringify(r));
  // 各要素の取り出し
  console.log("---");
  console.log(r.items.item[0].name[0]);
  console.log(r.items.item[0].price[0]);
});
```

実行すると以下のような結果になります。

```
{"items":
  {"item":[
    {"name":["Banana"],"price":["130"]},
    {"name":["Apple"],"price":["300"]},
    {"name":["Pear"],"price":["250"]}
  ]}
}
---
Banana
130
```

JavaScript のオブジェクトから XML を作成する場合

逆に、JavaScript のオブジェクトから XML を作成する場合も見ておきましょう。こちらは、Builder クラスを利用します。

● file: src/ch02/04-xmlrss/test-xml-builder.js

```
// モジュールの取り込み
var xml2js = require('xml2js');

// 変換元オブジェクト
var obj = {
  item: {name:"Banana", price:150}
};
// XML に変換
var builder = new xml2js.Builder();
var xml = builder.buildObject(obj);
console.log(xml);
```

実行すると以下のような結果になります。

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<item>
  <name>Banana</name>
  <price>150</price>
</item>
```

もしも、思ったような XML 結果が出ない場合は、先に、目的とする XML を作り JavaScript のオブジェクトに変換してみます。その後、JavaScript の構造を確認した上で、XML に変換するようにします。

以下は、XML データを、JavaScript のオブジェクトに変換し、それをさらに、XML に書き戻すというプログラムです。

● file: src/ch02/04-xmlrss/test-xml-builder2.js

```
// モジュールの取り込み
var xml2js = require('xml2js');
var parseString = xml2js.parseString;
var Builder = xml2js.Builder;

// テスト用の XML データ
var xml = "<fruits shop='AAA'>" +
  "<item price='140'>Banana</item>" +
  "<item price='200'>Apple</item>" +
  "</fruits>";
```

```
// XMLをパースする
parseString(xml, function (err, r) {
  // JavaScriptのオブジェクトを表示
  console.log(JSON.stringify(r));
  // パースしたものをXMLに変換
  var xml = new Builder().buildObject(r);
  console.log(xml);
});
```

COLUMN

廃止された「E4X」について

かつて、XMLを手軽に処理するために「E4X(ECMAScript for XML)」という仕様が「ECMA-357」で標準化され、いくつかのJavaScriptエンジンに搭載されました。これは、JavaScriptにネイティブのXMLサポートを追加する言語拡張です。単純な構文でXMLの要素にアクセスできるインターフェイスを提供しました。例えば、以下のように、E4Xを使うと手軽にJavaScriptの構文でXMLを記述することができました。

● file: src/ch02/04-xmlrss/rhino-e4x.js

```
var fruits = <fruits shop="A Mart">
  <item price="130">Banana</item>
  <item price="200">Apple</item>
  <item price="500">Mango</item>
</fruits>;
print(fruits.@shop);                    // 結果： A Mart
print(fruits.item.(@price == 500)); // 結果： Mango
```

しかし、現在、E4Xは多くのJavaScript実装で廃止されました。それまで、積極的にE4Xを推し進めていたFirefoxも、バージョン17でデフォルト無効、バージョン21で完全廃止となりました。廃止理由は、限られた将来性、関心の薄さ、コードの複雑性といったものです。筆者は個人的にE4Xが便利な機能だと思っていたので、廃止になって残念でした。それでも、現在のところ、E4XはJavaScriptエンジンの「Rhino」で利用することができます。（上記のコードもRhinoで正しく動作します。）ただし、多くの実行エンジンで実装されなくなっているので、今後は自己責任で使うことになりそうです。

RSSとは？

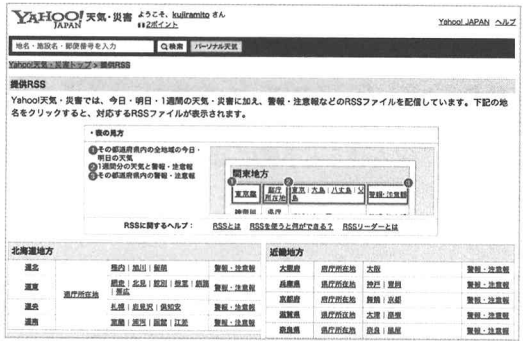
XMLがわかったところで、次に、RSSについて紹介します。RSSとは、ニュースやブログなど各種Webサイトの更新情報を、要約してまとめて配信するためのデータ形式です。ただし、RSSにはいくつかのフォーマットが存在します。有名なものには、「RSS1.0」「RSS2.0」「Atom」があり、それぞれ広く配信されていま

す。さまざまなフォーマットがあるので、少し大変そうに思うかもしれませんが、どれもXMLが基になっているので、それほど大きな差はありません。

Yahoo! の天気予報 RSS を読もう

では、実際のRSSの例として、Yahoo! Japanが提供している「天気・災害」のRSSを読んでみましょう。ここで提供されるRSSは、各地方の週間天気予報です。RSSとして提供されているので、手軽に天気予報データを取り出すことができます。

```
Yahoo! 天気・災害トップ > 提供RSS
http://weather.yahoo.co.jp/weather/rss/
```



Yahoo! 天気・災害の RSS 一覧ページ

例えば、東京のRSSデータを見てみましょう。このRSSフォーマットは「RSS2.0」です。ぱっと見ただけでは、その構成がよくわからないかもしれません。ですが、よくよく見ていくと、XMLの構造化されたデータであり、とても規則的にデータが並んでいることがわかります。

東京の週間天気予報 RSS

どのようなデータが提供されているのか詳しく見ていきましょう。Yahoo! 天気予報のRSSでは、下記のようなデータが配信されています。

```
<rss>
  <channel>
    <title>RSS 全体のタイトル </title>
    <description> 全体の説明 </description>
    ...
    <item>
```

```
      <title>記事の見出し</title>
      <link>記事のURL</link>
      <description>記事の説明</description>
      <pubDate>記事の日付</pubDate>
    </item>
    <item>
      <title>記事の見出し</title>
      ...
    </item>
    ...
  </channel>
</rss>
```

週間天気予報 RSS を取得してみよう

では、実際にプログラムを作ってみましょう。以下のプログラムは、東京の週間天気予報を表示するものです。Yahoo! の天気予報 RSS をダウンロードし、それを解析して週間予報をコンソールに出力します。

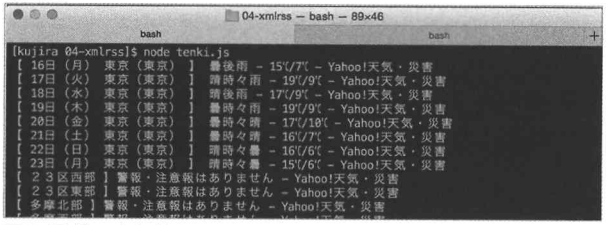
● file: src/ch02/04-xmlrss/tenki.js

```
// Yahoo!Japan 天気予報 RSS
var RSS = "http://rss.weather.yahoo.co.jp/rss/days/4410.xml";

// モジュールの取り込み
var parseString = require('xml2js').parseString;
var request = require('request');

// RSS をダウンロード ---- (※1)
request(RSS, function (err, response, body) {
  if (!err && response.statusCode == 200) {
    analyzeRSS(body);
  }
});

// RSS を解析する ---- (※2)
function analyzeRSS(xml) {
  // XML を JS のオブジェクトに変換
  parseString(xml, function(err, obj) {
    if (err) { console.log(err); return; }
    // 天気を表示 ----- (※3)
    // console.log(JSON.stringify(obj)); // ----- (※4)
    var items = obj.rss.channel[0].item;
    for (var i in items) {
      var item = items[i];
      console.log(item.title[0]);
    }
  });
}
```

このプログラムを実行するには以下のコマンドを実行します。

```
$ node tenki.js
```

実行すると次のような結果になります。



天気予報の実行結果

ここでは、RSS の階層から、/rss/channel/item/title のデータだけを取り出して表示しています。

プログラムを少しずつ見ていきましょう。プログラム中の (※ 1) の部分で、RSS を Web から取得 (ダウンロード) するのが「request」モジュールの request() メソッドです。Yahoo! 天気予報の RSS をダウンロードし、引数として得られる response の statusCode を見て 200 であれば正しくデータが取得できたことになります。

正しくデータが取得できたなら、関数 analyzeRSS() を呼びます (※ 2 の部分)。ここでは、「xml2js」モジュールの、parseString() メソッドを使うことにより、XML データ（つまり、Yahoo! の天気予報 RSS) を JavaScript のオブジェクトに変換します。

プログラム中の (※ 3) では、変換した天気予報の情報を for 構文で順次コンソールに出力しています。

どんな形の JavaScript オブジェクトになっているのかを確認するには、プログラムの (※ 4) のように、JSON.stringify() でオブジェクトを JSON に変換したものを出力すると良いでしょう。このとき出力される JSON オブジェクトは人間が目視しても構造がわかりにくいものです。そこで、JSON オブジェクトを自動整形してくれる Web サービスなどを利用すると、わかりやすく構造化してくれます。

XML/RSS の解析に「cheerio-httpcli」を使う方法

さて、上記の部分では「xml2js」モジュールを使って XML/RSS を解析しましたが、実は「cheerio-httpcli」モジュールを使うと、より手軽にプログラムを記述できます。同じプログラムを、「cheerio-httpcli」使って書き直してみましょう。

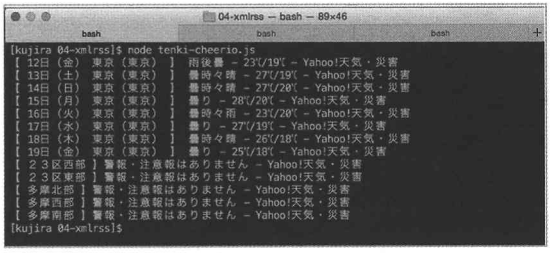
● file: src/ch02/04-xmlrss/tenki-cheerio.js

```
// Yahoo!Japan 天気予報 RSS(cheerio 利用版 ) for Node.js

// 対象 RSS
var RSS = "http://rss.weather.yahoo.co.jp/rss/days/4410.xml";

// モジュールを読む
var client = require('cheerio-httpcli');

// RSS をダウンロード
client.fetch(RSS, {}, function(err, $, res) {
  if (err) { console.log("error"); return; }
  // 必要な項目を抽出して表示 ---------------------- ( ※ 1)
  $("item > title").each(function(idx) {
    var title = $(this).text();
    console.log(title);
  });
});
```

プログラムを実行するには以下のコマンドを実行します。

```
$ node tenki-cheerio.js
```

実行すると以下のような結果になります。



天気予報の実行結果

注目したい点としては、プログラムの（※1）の部分です。RSS の正確な階層を指定することなく、「item > title」だけを取り出すように指定しています。「cheerio-httpcli」モジュールを使うと CSS クエリーを使って好きな部分の情報を手軽に取り出すことができるのです。どの部分の情報が欲しいのかにもよりますが、より気軽にプログラムを記述できます。

この節のまとめ

- **ここでは、XML と RSS について紹介し、Node.js を使ってそれらを解析する方法について紹介しました。**
- **多くの Web サービスが、積極的に RSS を配信しているので、その情報をうまく利用すれば、非常に有用です。**

05 定期的にダウンロードする

特定のデータを定期的にダウンロードしたい場合があります。そこで、ここでは定期的なダウンロードを行うための方法を紹介します。定期的な処理を実行する方法は、OSにより異なります。ここでは、Mac OS X/Linuxの場合とWindowsの場合とに分けて紹介します。

ここで学ぶポイント

- cronを使う
- Windowsでタスク スケジューラを使う

ツールやライブラリの一覧

- cron
- タスク スケジューラ

定期的な処理を実行したい

Webで公開されるデータの中には、定期的にデータが更新されるものが多くあります。すぐに思いつくところでは、株価や為替を初めとする金融関連データ、また、先に紹介した天気予報なども定期的に更新されるものです。

そこで利用したいのが、定期的に処理を実行する仕掛けです。Mac OS XやLinuxでは、「cron」というデーモンプロセスを利用します。cronを利用すると、スクリプトを自動実行することができます。同様に、Windowsでは、「タスク スケジューラ」が用意されています。ここでは、これらの利用方法を紹介します。

定期実行のためのヒント

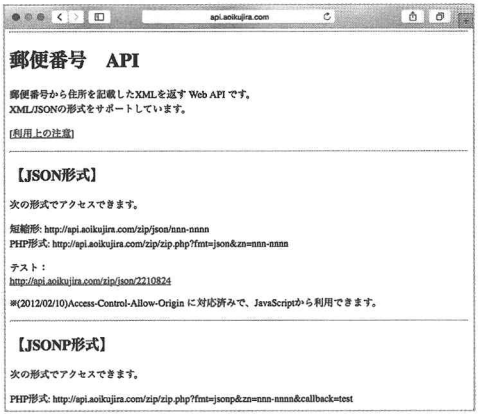
「タスク スケジューラ」や「cron」を使って何を定期実行すれば良いでしょうか。定期実行したい処理を体系的に分類するとしたら、以下のようになります。

(1) **データ収集などアプリ内で必要となる定期処理**
(2) **ログやバックアップなどシステム関連の定期処理**
(3) **システムが正しく動作しているか定期的に監視する処理**

本書で扱うような定期的なデータのダウンロードは(1)に分類されるでしょう。他にも、データベースを集計するなど、アプリケーションがスムーズに動作するために必要となる処理を定期的に実行するために利用します。また(2)では、システム自体の動作を軽快にするために、ログファイルを差し替えたり、必要なデータをバックアップしたりといった用途で利用します。それから、(3)のように、システムが正しく動作しているかを確認する用途でも多く利用されています。もし、システムが止まっていれば、管理者にメールで通知して該当システムを再起動させることができるでしょう。

為替の変動を確認する API を使う

ここでは、定期的に実行するサンプルの処理として、為替確認 API という Web API を利用してみます。



為替確認 API

```
クジラ外国為替確認 API
http://api.aoikujira.com/kawase/
```

これは、筆者が趣味で公開しているものですが、書籍のサンプルとして便利なので仕様例として紹介します（ただし、この値は積極的な FX などの為替トレードに使うことを想定していません。更新頻度も低く、1 日に数回更新されるだけなので、参考程度にご利用ください）。

為替 API では、以下のような URL にアクセスすると、JSON 形式や XML 形式でその時の為替レートを返します。

```
JSON 形式で為替レートを得る
http://api.aoikujira.com/kawase/get.php?code=USD&format=json

XML 形式で為替を得る
http://api.aoikujira.com/kawase/get.php?code=USD&format=xml
```

上記の値は、アメリカドル (USD) を基本とした値となっています。API の引数を変えることで、基本通貨を変更することができます。上記の「code=USD」という部分を「code=MYR」に変えればマレーシア・リンギットに、「code=CNY」に変えれば中国・人民元を基本とすることができます。もちろん日本円 (JPY) を基本としたレートも表示できますが、小数点以下の精度が低いため、不正確になってしまうので、アメリカドルを基本にしています。

さっそくプログラムを作ってみましょう。毎日プログラムを実行することを考えて、日付を含むファイル名に、1 ドル何円なのか為替レートを書き込むという処理を作ってみます。

● file: src/ch02/05-cron/kawase-usd_jpy.js

```
// 為替情報を取得 for Node.js

// 為替 API の URL
var API = "http://api.aoikujira.com/kawase/get.php?code=USD&format=json";
```

```
// モジュールの取り込み
var request = require('request');
var fs = require('fs');

// Web APIにアクセスする
request(API, function(err, response, body) {
  // HTTPのエラーをチェック
  if (err || response.statusCode != 200) {
    console.log("ERROR", err); return;
  }
  // JSONをJSのオブジェクトに変換
  var r = JSON.parse(body);
  var jpy = r["JPY"];
  // 為替レートをファイルへ保存(ファイル名には日付を入れる)
  var t = new Date();
  var fname = "USD_JPY_" +
      t.getFullYear() + "-" + (t.getMonth()+1) +
      "-" + t.getDay() + ".txt";
  var text = "1usd=" + jpy + "jpy";
  console.log(text);
  fs.writeFile(fname, text);
});
```

このプログラムを実行すると、例えば「USD_JPY_2015-4-2.txt」というテキストファイルに「1usd=119.62386jpy」という値が書き込まれます。為替は日々上がったり、下がったりしますから、毎日、定期的に自動実行されるようになっていると便利ですね。

それでは、定期的なプログラムの実行を行うようにしましょう。LinuxとMac OS Xで設定する方法、また、Windowsで設定する方法とで分けて紹介します。

Linux/Mac OS Xの場合

CentOSを含めたLinuxやMac OS Xの場合は、「cron」を利用します。Mac OS Xを含め、UNIX系のOSであれば、たいていのシステムにcronが標準でインストールされています。cronを利用するには、指定された設定ファイルに指定された形式で実行間隔を記述します。設定ファイルにテキスト形式で記述するというUNIX系OSの伝統に則っているため、人によっては難しいと感じることもあるでしょう。

必要に応じて「nano」エディターをインストール

cronを設定するには、ターミナルより「crontab」というコマンドを実行し、そこで開かれたファイルを編集します。標準では「vi」エディターが起動し、そのエディターを利用してファイルの編集を行います。しかし、「vi」は操作体系が独特なので、不慣れな人には、ストレスかもしれません。

「vi」を使ったことがないという方は、「nano」をインストールして、これでファイルの編集を行うのがよいでしょう。CentOSで、nanoをインストールするには、以下のコマンドを実行します。

```
$ sudo yum install nano
```

次に、cronの編集でnanoが起動するように設定してみます。ちなみに、この設定を行うのにも、nanoを使います。コマンドラインから以下のコマンドを実行しましょう。ホームディレクトリにある設定ファイル「.bash_profile」をnanoで編集します。

```
$ nano ~/.bash_profile
```

nano が起動したら、ファイルの末尾に以下を追加しましょう。編集が終わったら、[Control] キー＋ [x] キーでエディターを閉じます。エディターを閉じる前に、ファイルを保存するか聞かれるので [y] キーで保存することを選び、ファイル名を確認して [Enter] キーを押すとファイルの編集が完了します。

```
# cron の編集に nano を利用する
export EDITOR=nano
```

nano で .bash_profile を編集しているところ

「~/.bash_profile」を編集後、ログインし直すか、「source ~/.bash_profile」とコマンドを実行して設定を反映させましょう。

cron の設定を行う「crontab」

準備ができたら、「crontab」コマンドを使って、cron の設定を行いましょう。crontab を起動するときは、「-e」オプションをつけて起動します。すると、cron の設定画面が開きます。初めて「crontab -e」を実行した時には、設定も何も記述されていないでしょう。

```
$ crontab -e
```

例えば、毎日、朝の 7 時に、本節の冒頭で作成した「kawase-usd_jpy.js」を実行するプログラムを起動するように設定してみましょう。**

```
0 7 * * * /usr/bin/node /path/to/kawase-usd_jpy.js
```

上記のように記述したら保存してエディターを終了します。nano エディターを利用した場合には、Ctrl+X キーを押します。すると保存するか尋ねられるので Y キーを押し [Enter] で確定します。これで、毎朝 7 時に為替情報を取得するようになります。

** プログラムのパスは、ご自身の環境に合わせて変更してください

環境変数に注意

cronの実行時には、環境変数が最低限しか設定されません。そのため、パスが通らずにコマンドが実行できない、Node.jsのモジュールパスが見つからないなどの問題が発生することがあります。そこで、crontabでは、設定ファイルの冒頭で環境変数を設定できるようになっています。

```
PATH=/usr/local/bin:/usr/bin/:/bin
NODE_PATH=/usr/lib/node_modules/

0 7 * * * node /path/to/kawase-usd_jpy.js
```

ただし、crontabの冒頭で指定する環境変数は右辺を展開しないという仕様になっているので、注意が必要です。

```
# crontabで環境変数などは展開されない
# 間違った指定
PATH=/usr/local/bin:$PATH
# 正しい指定
PATH=/usr/local/bin:/usr/bin:/bin
```

あるいは、同じですが、各行で個別の環境変数を設定することもできます。

```
0 7 * * * export NODE_PATH=/usr/lib/node_modules/ && /usr/bin/node /path/to/
kawase-usd_jpy.js
```

カレントディレクトリに注意

cronの実行時には、カレントディレクトリがユーザーのホームディレクトリとなります。そのため、ログなどを保存する際には、フルパスを指定するか、カレントディレクトリを変更するなどの対策が必要となります。

そのため、今回のプログラムをcronに登録すると、為替情報が、ユーザーのホームディレクトリに保存されてしまうことになります。それはそれで良いものの、スクリプトと同じディレクトリにログファイルを保存するには、次のようなシェルスクリプトを書いて実行環境を整えた上で、それをcronに登録すると良いでしょう。

● file: src/ch02/05-cron/kawase.sh

```
#!/bin/sh

# パスを設定
PATH=/usr/local/bin:/usr/bin:/bin
NODE_PATH=/usr/lib/node_modules

# カレントディレクトリをスクリプトのパスに変更
cd `dirname $0`
# 為替スクリプトを実行
node kawase-usd_jpy.js
```

上記のシェルスクリプトをcronに登録するには、crontabへ以下のように記述します。

```
0 7 * * * /path/to/kawase.sh
```

crontabのさまざまな指定方法

さて、もう少しcrontabへの記述方法について見ていきましょう。基本的に、次のような書式で記述します。

```
[書式] crontab
分 時 日 月 曜日 実行するコマンド
```

各フィールドに設定可能な数値は、以下の通りです。

フィールド	数値
分	0-59
時	0-23
日	1-31
月	1-12
曜日	0-7 (0または7は日曜日)

また、数値以外にも、次の指定が可能です。

名前	利用例	説明
リスト	0,10,30	0,10,30という値をそれぞれ指定します
範囲	1-5	1,2,3,4,5という範囲を指定します
間隔	*/10	10,20,30と10間隔で指定します
ワイルドカード	*	ワイルドカードで指定します

これを踏まえて、具体的な設定例を見てみましょう。
以下は、Mac OS Xで、毎時0分になると、"Hello"と挨拶をするように設定する例です。

```
0 * * * * say "Hello"
```

そして、以下は、毎朝8時30分になると、"Good morning"と挨拶をするように設定する例です。

```
30 8 * * * say "Good morning"
```

さらに、以下は、毎月20日の18時32分に、"Use money with care"と注意喚起するよう設定する例です。

```
32 18 20 * * say "Use money with care"
```

特定の日時を指定する場合、例えば、毎年5月6日7時8分に、"Have a nice day"と挨拶するように設定する例です。

```
08 07 06 05 * say "Have a nice day"
```

こちらは、毎週月曜日の朝 7 時 50 分に「ゴミの日だよ」と教えてくれるように設定する例です。

```
50 07 * * 1 say "ゴミの日だよ"
```

ところで、中国語では、月曜日を星期一、火曜日を星期二、水曜日を星期三・・・と数えますが、Cron における曜日の指定においても、同じように数値で指定するようになっています。

数値と曜日の対応は以下のようになっています。

曜日	数値
月曜日	1
火曜日	2
水曜日	3
木曜日	4
金曜日	5
土曜日	6
日曜日	7 あるいは 0

ちなみに、月末のみ処理を実行したい場合などは、crontab の指定だけでは実現できないのですが、test コマンドと組み合わせることで、月末処理を実行することができるようになっています。

```
50 23 28-31 * * /usr/bin/test $( date -d '+1 day' +%d ) -eq 1 && 実行したいコマンド
```

cron を実行したとき、標準出力やエラー出力があると、メールで通知してくれますが、これを無効にしたい場合には、crontab の冒頭で、MAILTO を空にしておきます。

```
MAILTO=""
```

Windows の場合

Windwos の『タスク スケジューラ』には、GUI が用意されているので、画面の指示に沿って、プログラムの定期実行を設定することができます。

ちなみに『タスク スケジューラ』の名前が紛らわしいのは、検索の時に「タスク」と「スケジューラ」の間に半角スペースがあるという点です。Windows メニューの文字列検索で、半角スペースを忘れて「タスクスケジューラ」としたり、半角スペースを入れても音を伸ばして「タスク スケジューラー」としたりしても見つけられません。

半角なしでは見つからない

「タスク スケジューラ」が見つかった

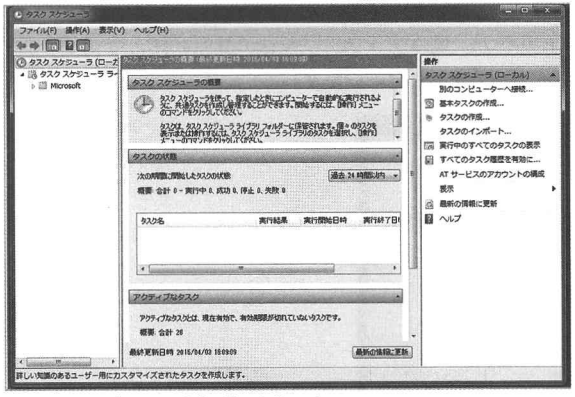
Windows7 で検索せずに起動するには、Windows の [スタート] ボタンから、[コントロール パネル > システムとセキュリティ > 管理ツール > タスク スケジューラ] をクリックします。

Windows8/8.1 では、Windows キーを押しながら「X」キーを押し、表示された一覧から、「コントロールパネル」をクリックします。それから、[システムとセキュリティ > 管理ツール > タスク スケジューラ] を選んでクリックします。

このとき、管理者のアクセス許可が必要となり管理者のパスワード入力が必要な場合もあります。

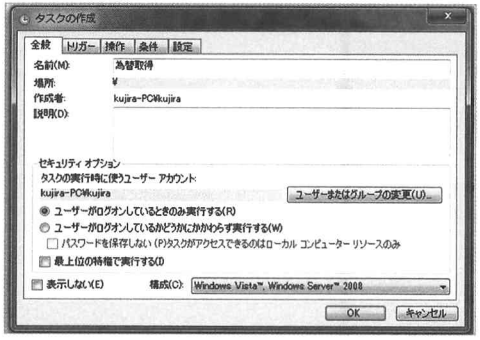
新規タスクの作成

Windows の『タスク スケジューラ』では、定期的に実行する処理を「タスク」と呼び、タスクを作成することで、定期的な自動実行を行う事ができます。新しくタスクを作成するには、まず、タスクスケジューラの左ペインのツリーで［タスク スケジューラ (ローカル)］フォルダーをクリックします。次いで、右側の操作ペインで［基本タスクの作成］をクリックします。これで「タスクの作成」ウィザードが起動します。



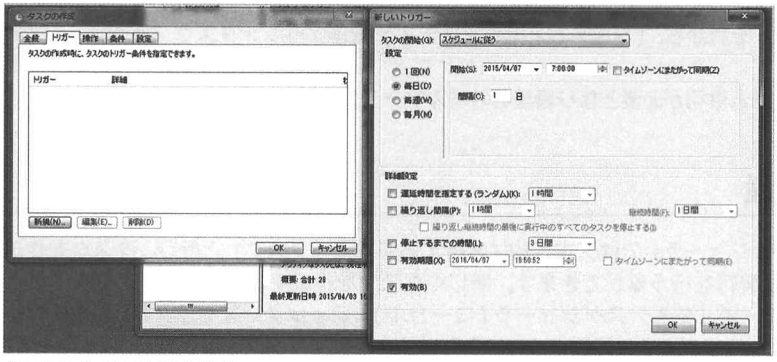
タスク スケジューラを起動したところ

まずは、「全般」のタブから「タスクの作成」ウィザードを実行し、名前を「為替取得」などに設定します。



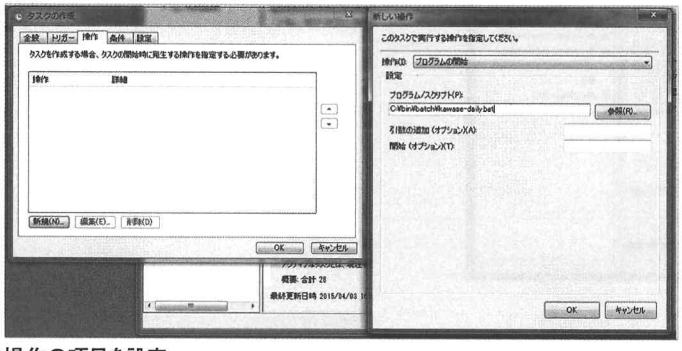
全般タブを設定

次に、トリガーを指定します。「トリガー」のタブで「新規」ボタンをクリックします。すると、「新しいトリガー」ダイアログが出ます。ここで、次の図のように、タスクの開始を「スケジュールに従う」に、設定を「毎日」に、開始を指定日の「7:00:00」に、間隔を 1 にします。また、詳細設定「有効」にチェックします。



タスクの作成の項目を設定

それから、「操作」タブを開いて、実行するスクリプトを登録しましょう。「新規」ボタンをクリックすると、「新しい操作」ダイアログが表示されます。それで、操作を「プログラムの開始」に、プログラム / スクリプトへ定期的に実行したいバッチファイルやプログラムを指定します。



操作の項目を設定

これで、操作としては完成なのですが、「プログラムの開始」ダイアログで指定するバッチファイルは、以下のようなものにします。これを「kawase-daily.bat」などの名前で保存して利用します。

```
rem 実行ディレクトリを指定
cd "C:\sampledir\06-cron\"

rem Node とプログラムのパスを指定
"C:\Program Files\nodejs\node.exe" kawase-usd_jpy.js
```

バッチファイルをテストするために、上記のバッチファイルの最下行に「PAUSE」と書いた後に、バッチファイルをダブルクリックで実行してみてください。バッチファイルの実行に失敗する場合は、エラーメッセージが表示されますので、メッセージが正しく実行されるか確認してみてください。「タスク スケジューラ」で実行する際には「PAUSE」を消すのを忘れないようにしましょう。

ちなみに、最近の Windows では、ネットワークドライブ上で、バッチファイルを実行することはできないようになっています。この場合には、バッチファイルではなく WSH で起動スクリプトを書いたり、スクリプトをローカルドライブにコピーしてから実行するなどの工夫が必要です。

この節のまとめ

- **この節では、「タスク スケジューラ」や「cron」を使って、定期的なデータ収集を行う方法を紹介しました。**
- **題材とした為替のダウンロードプログラムも実用的なものでしょう。**
- **ここでは為替レートのダウンロードしか行いませんが、これを発展させてデータベースに登録したり、長期的な視点で為替売買の予想をしたりすることもできると思います。**

第3章

ログインの必要なWebサイトをクロールする

前章では基本的なWebコンテンツのダウンロードやWebページの解析方法を紹介しました。本章では、さらに推し進めて、Webサイトのスクリーンショットを撮ったり、ログインの必要なWebサイトを対象としてダウンロードしたりする方法を解説します。

01 PhantomJSとCasperJS

Webの自動化処理について語るとき、外せないツールがあります。それがPhantomJSとCasperJSです。ここでは、それぞれのインストールと基本的な使い方を紹介します。これらのツールはWebスクレイピングの強力なツールです。

ここで学ぶポイント	ツールやライブラリの一覧
●PhantomJSとCasperJSについて	●PhantomJS ●CasperJS

PhantomJSとCasperJSについて

複数のページを遷移したり、ログイン後のデータを取得したりする時には専用のツールを利用するのが一番です。PhantomJSとCasperJSは、フォームに値を設定したり、特定のボタンをクリックしたりと、Webブラウザーにおける UI の自動化作業を強力にサポートしてくれるツールです。

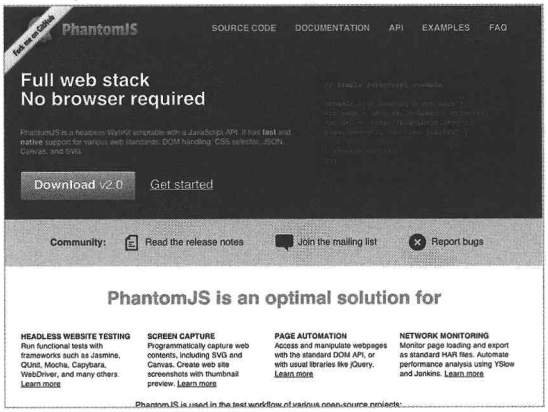
それぞれの役割を解説すると「PhantomJS」が画面のないWebブラウザーで、「CasperJS」はPhantomJSを簡単に使うためのライブラリとなっています。

ツール名	説明
PhantomJS	コマンドラインから使えるブラウザー
CasperJS	PhantomJSを手軽に使うライブラリ

PhantomJSについて

そもそも、PhantomJSは、コマンドラインから使えるWebブラウザーです。レンダリングエンジンには、「WebKit」が採用されています。「WebKit」とは、AppleのWebブラウザー「Safari」に使われているレンダリングエンジンです。また、今でこそGoogle Chromeは「Blink」というレンダリングエンジンを使っていますが、もともとはWebKitを使用していました。Blinkもここから分岐したものです。このほか、PhantomJSのJavaScriptエンジンにはJavaScriptCoreが使われており、これもSafariと同じです。

PhantomJSを利用すると、コマンドラインから、Webブラウザーを操作して、ブラウザー内に表示されるデータを取得したり、スクリーンショットを撮ったりすることができます。Webサイトからデータを取り出すスクレイピングに使ったり、UIテストの自動化を行ったりと、さまざまな活用方法が考えられます。



PhantomJS の Web サイト

```
PhantomJS の Web サイト
[URL] http://phantomjs.org/
```

CasperJS について

CasperJS は、PhantomJS をより簡単に使うために用意されているライブラリです。そのため、CasperJS を使うには、PhantomJS のインストールが必須となります。また、本書では扱いませんが、PhantomJS だけではなく、Firefox のレンダリングエンジンである Gecko 向けの SlimerJS を使うことも考慮されています。

```
CasperJS の Web サイト
[URL] http://casperjs.org/
```

CasperJS の Web サイト

PhantomJS と CasperJS のインストール

では、PhantomJS と CasperJS をインストールしましょう。インストールは、Node.js のパッケージ管理ツールである npm を使うことができます。Windows や Mac OS X では、次のようにコマンドを実行すると PhantomJS と CasperJS のインストールが完了します。

```
# PhantomJS のインストール
$ sudo npm install -g phantomjs
# CasperJS のインストール
$ sudo npm install -g casperjs
```

ただし、CentOS6 では、上記のコマンドを実行する前に、freetype と fontconfig のインストールが必要となります。以下のコマンドを実行してこれらをインストールしてから、上記の npm のコマンドを実行しましょう。

```
$ sudo yum install freetype
$ sudo yum install fontconfig
```

PhantomJS と CasperJS のインストールが完了したら、以下のコマンドを実行してみてください。これは、PhantomJS(本体) のバージョン番号を表示するものとなっています（注意したい点として、PhantomJS の本体バージョンと、npm でインストールするモジュールのバージョンが若干異なります)。

```
$ phantomjs -v
1.9.8
```

サンプルプログラムが動かない場合

ところで、筆者が原稿執筆時に npm でインストールしたときのバージョンは、PhantomJS が 1.9.17(本体バージョンは 1.9.8)、CasperJS が 1.1.0-beta3 でした。もし、インストールに失敗する、あるいは、新しいバージョンになって、サンプルが動かないという場合には、下記のようにバージョンを指定してインストールしてみてください。

```
# 特定のバージョンを指定してインストールする場合
$ npm install -g phantomjs@1.9.17
$ npm install -g casperjs@1.1.0-beta3
```

npm でモジュールのどのバージョンがインストールされているかを調べるには、「npm list」コマンドを利用します。グローバルインストールしているものを調べたければオプション「-g」を付けます。

```
# phantomjs のバージョンを調べる
$ npm list -g phantomjs
$ npm list -g casperjs
```

簡単なサンプルプログラム

それでは、PhantomJSとCasperJSを利用して、Webサイトにアクセスして、そのサイトのタイトルを表示するプログラムを作ってみましょう。次のようなプログラムになります。

● file: src/ch03/01-phantomjs/getTitle.js

```
// Webサイトからタイトルを表示する
var TARGET_URL = "http://kujirahand.com";

// CaperJSのオブジェクトを作成 ---- (※1)
var casper = require('casper').create();

// 指定のWebサイトを開く ---- (※2)
casper.start(TARGET_URL, function() {
  // タイトルを表示する ---- (※3)
  this.echo(casper.getTitle());
});

// 処理を実行する --- (※4)
casper.run();
```

プログラムを実行するには、次のようなコマンドを実行します。

```
$ casperjs getTitle.js
```

実行すると、kujirahand.comにアクセスが行われて「くじらはんど」とコンソールに出力があります。

プログラムを確認してみましょう。プログラムの(※1)では、CasperJSのオブジェクトを作成します。CasperJSを利用するには、まず、require('casper')と続くcreate()メソッドでCasperJSのオブジェクトを作成するところから始まります。

プログラムの(※2)では、一番はじめに表示するページを、start()メソッドで指定しています。ただし、この時点では、CasperJSは実行されません。実際にプログラムが実行されるのは、プログラムの(※4)の場所で、つまりrun()メソッドを実行した時点です。ところで、start()メソッドの引数には、URLと指定したページを開いたときに実行する処理を記述しています。また、プログラムの(※3)は、ページを表示したときに実行する処理ですが、現在表示しているサイトのタイトルを取得して表示します。

CasperJSでの実行に関して、より詳しい情報が欲しい場合には、コマンド実行時に「--verbose」や「--log-level=debug」を指定します。これにより、より詳しいデバッグ情報が色つきで表示されます。

デバッグを実行してみたところ

画面キャプチャを撮るプログラム

それでは、もう少し気の利いたサンプルとして、画面キャプチャを撮るプログラムを作ってみます。以下は、Google のサイトのトップページを画面キャプチャする例です。

● file: src/ch03/01-phantomjs/screenshot.js

```
// CasperJS でスクリーンショットを撮る

// Casper オブジェクトを作成 ---- (※1)
var casper = require('casper').create();

// 開始する --- (※2)
casper.start();

// ページを開く --- (※3)
casper.open('http://google.co.jp');

// その後、スクリーンショット撮影 --- (※4)
casper.then(function() {
  casper.capture("screenshot.png");
});

// 実行 --- (※5)
casper.run();
```

プログラムを実行するには、次のように入力します。

```
$ casperjs screenshot.js
```

プログラムを実行すると、次のような画像ファイルが保存されます。非常に簡潔なコードでありながら簡単に画面キャプチャが撮影できました。

CasperJS でスクリーンショットを撮ったところ

では、プログラムを確認してみましょう。プログラムの (※1) の部分で、CasperJS のオブジェクトを作成します。

プログラムの (※2) で CasperJS の処理を開始します。とは言え、先のプログラムでも指摘したように、実際にプログラムが実行されるのは、プログラムの (※5) で run() メソッドを実行したときです。そのため、ここでいう start() メソッド以降の部分は、「ここから CasperJS で自動処理を開始」という感じのものと考えてください。

プログラムの(※3)の、open()メソッドで任意のURLを開きます。続く、プログラムの(※4)部分の、then()メソッドでページを開いた後に何をするかを指定します。then()メソッドの引数には、何をするのかに相当する関数オブジェクトを指定することになっています。

最後に、プログラムの(※5)で実際の処理を実行します。

Flickrで検索結果を表示してみる

次に、画像共有サイトのFlickrを使って、公開されているネコの写真を検索し、その検索結果のスクリーンショットを保存するプログラムを作ってみようと思います。

ネコの写真を検索して画面キャプチャしたところ

このプログラムでは、単にWebサイトにアクセスしてスクリーンショットを撮るだけではありません。まず、Flickrのサイトにアクセスし、次に、画面上部にある検索フォームに「ネコ」という値を設定してフォームを送信し、その検索結果をキャプチャするという仕組みになっています。

● file: src/ch03/01-phantomjs/flickrShot_2.js

```
// Flickrで検索しスクリーンショット撮る for CasperJS
// CasperJSのオブジェクトを作成
var casper = require('casper').create();

// CasperJSの処理を開始する ---- (※1)
casper.start();

// 画面サイズを指定する ---- (※2)
casper.viewport(1400, 800);

// UserAgentの指定
casper.userAgent('User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/37.0.2062.120 Safari/537.36');

// Flickrのサイトでネコを検索 ---- (※3)
var text = encodeURIComponent("ネコ");
casper.open('https://www.flickr.com/search/?text=' + text);

// その後、画面をキャプチャ ---- (※4)
casper.then(function(){
  this.capture('flickr-cat.png',{
    top:0, left:0, width: 1400, height: 800
  });
});
```

```
// 実行開始
casper.run();
```

このプログラムを実行するには、コマンドラインから以下のコマンドを実行します。

```
$ casperjs flickrShot_2.js
```

さて、プログラムを見ていきましょう。CasperJS のオブジェクトを作成したら、プログラムの (※ 1) の部分にあるように、start() メソッドを呼びます。start() メソッドの引数に URL を与えれば、そのサイトを開きますが、このように、start() メソッドに引数を与えない場合は、CasperJS に空のページを作成することになります。その後に、open() メソッドを使って任意の Web サイトを開きます。ところで、ブラウザーの画面サイズを指定する際には、プログラムの (※ 2) にあるように、viewport() メソッドでサイズを指定できます。

プログラムの (※ 3) では、ネコを検索する Flickr の Web サイトを開きます。そして、(※ 4) の then() メソッドで、サイトを開いた後の処理、つまり、画面キャプチャを行います。ここでは、キャプチャする範囲を引数として指定領域をキャプチャします。

改めて CasperJS の流れを確認

このように、CasperJS では、start() メソッドから run() メソッドの間に、次々と実行したい処理を then() メソッドで指定するというのが基本的な流れとなります。CasperJS でも非同期処理が基本となります。そこで、then() メソッドを使って、順に実行したい処理を引数として与えることで、処理が終わるごとに順々と次の処理を行わせることができます。

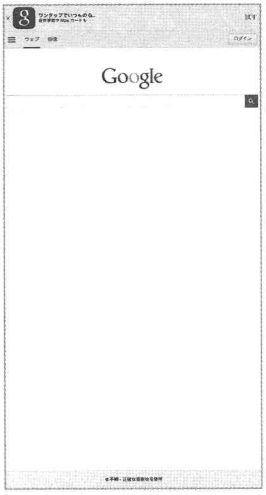
```
//--------------------------------
// CasperJS の基本的な流れ
//--------------------------------

// CasperJS のオブジェクトを作成
var casper = require('casper').create();

// 開始
casper.start();

// URL を開く
casper.open(URL);

// URL に対して何かしらの処理を行う
casper.then(function() { ... });
casper.then(function() { ... });
.
.
.

// 処理を実行する
casper.run();
```

iPhone のふりをして Web サイトを撮る

ところで、iPhone 用の画面を撮影したい場面は比較的よくあります。そこで、ここでは、ユーザーエージェント (UserAgent) を iPhone に設定した上でモバイル対応サイトにアクセスしてみます。

ユーザーエージェントとは Web サイトへのアクセスの際に使用されるプログラムのことを指します。つまり、どの Web ブラウザーを利用してサイトにアクセスしたかを示すものです。このユーザーエージェントを任意のものに変更することで、携帯サイトのテストをすることができます。

先ほど撮影した Google のスクリーンショットと比べてみると、異なるデザインが表示されたことがわかると思います。



iPhone のふりをしてスクリーンショットを撮ったところ

プログラムは、次のようになります。

● file: src/ch03/01-phantomjs/iphoneShot.js

```
// iPhone のふりをしてキャプチャ for CasperJS

var TARGET_URL = "http://google.co.jp";

// Casper を生成
var casper = require('casper').create();
casper.start();

// iPhone のふり --- (※1)
casper.userAgent('Mozilla/5.0 (iPhone; CPU iPhone OS 7_0 like Mac OS
X) AppleWebKit/537.51.1 (KHTML, like Gecko) Version/7.0 Mobile/11A465
Safari/9537.53');

// 画面サイズを指定 --- (※2)
casper.viewport(750, 1334);

casper.open(TARGET_URL);
```

```
// 画面キャプチャ
casper.then(function(){
  this.capture('screenshot.png');
});
// 実行
casper.run();
```

ここでのポイントは、プログラムの(※1)で、iPhoneのユーザーエージェントを設定している部分、それから、(※2)で画面サイズを指定している部分です。

コマンドライン・スクリーンショット撮影便利ツールを作る

このように、CasperJSを使うと手軽にスクリーンキャプチャを撮ることができます。そこで、コマンドライン引数にURLを与えると、そのURLのスクリーンショットを撮って保存するという手軽なツールを作ってみたいと思います。

CasperJSでは、実行時のコマンドライン引数が、casper.cli.argsに配列形式で入っています。これを利用すれば、コマンドラインからの入力を得ることができます。

ここでは、次のようなプログラムを作ってみました。

● file: src/ch03/01-phantomjs/shot-tool.js

```
// コマンドライン引数を得てスクリーンショットを撮る for CasperJS

var casper = require('casper').create();

 中略

// CasperJSの処理を開始
casper.start();
casper.viewport(1024, 768);
casper.open(url);
casper.then(function(){
  this.capture(savepath, {
    top:0, left:0, width:1024, height:768
  });
});
casper.run();
```

このツールを使って、キャプチャを撮るには次のようにします。

```
$ casperjs shot-tool.js http://google.com
```

すると、screenshot.PNGという名前の画像ファイルが作成されます。

Linux(CentOS)やMac OS Xで次のようなシェルスクリプトを作っておけば、もう少し手軽に呼び出せるように工夫できます。

● file: src/ch03/01-phantomjs/shot-tool.sh

```
#!/bin/sh
SCRIPT_DIR=`dirname $0`
/usr/local/bin/casperjs $SCRIPT_DIR/shot-tool.js $*
```

コマンドの使用例は次のようになります。

```
$ shot-tool http://google.com
```

また、ここでは、Webブラウザーの画面を1024x768ピクセルと決め打ちにしていますが、オプションで指定のサイズに変更できたり、ユーザーエージェントをiPhoneに変更できたりと、改良の余地がたくさんあります。ぜひ、皆さんがこのツールを改良してみてください。

この節のまとめ

- **PhantomJSとCasperJSのインストールから簡単な使い方までを紹介しました。**
- **ここで見たように、これらのツールを利用することで、一連の処理を行うのが比較的簡単に実現できるというのがわかるのではないでしょうか。**

02

ログイン後のデータをダウンロードする

ログインが必要な Web サイトでは、ログインした後でなければ、見られないコンテンツがあります。そして、そうしたコンテンツをダウンロードしたい場面も多くあります。ここでは、ログイン後のサイトからのダウンロードについて考察します。

ここで学ぶポイント	ツールやライブラリの一覧
●PhantomJSでログインする	●PhantomJS/CaperJS

ログインが必要な場面

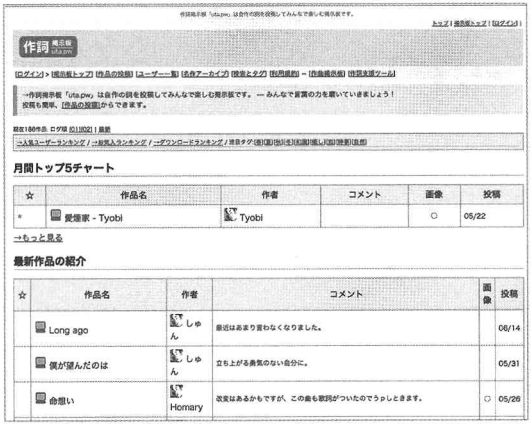
昨今の多くの Web サイトでは、ログインが必要となっています。SNS をはじめ、通販サイト、会員制のサイトなど、ログインしてはじめてさまざまなデータを読み書きすることができるようになっています。

ここでは、前節に引き続き、PhantomJS と CasperJS を利用して、会員制サイトへのログインを実現するプログラムを紹介します。また、CasperJS の便利なメソッドについても紹介します。

作詞掲示板にログインしてデータを取得しよう

それでは、ログインの機能があるサイトを例にとって、ログインデータを取得してみたいと思います。とは言え、既存の Web サイトだと、頻繁にログイン画面が変更される可能性があり、書籍のサンプルとして残すには、都合が悪いものです。ですから今後、数年の間、ログイン画面が変更されないサイトを例にとって解説したいと思います。

そこで、手前味噌で申し訳ないのですが、筆者が運営している、「作詞掲示板 (uta.pw)」のサイトを例にとって紹介してみます。このサイトは、自由に誰でも自作の詞を投稿できる Web サイトです。



作詞掲示板

作詞掲示板
[URL] http://uta.pw/sakusibbs/

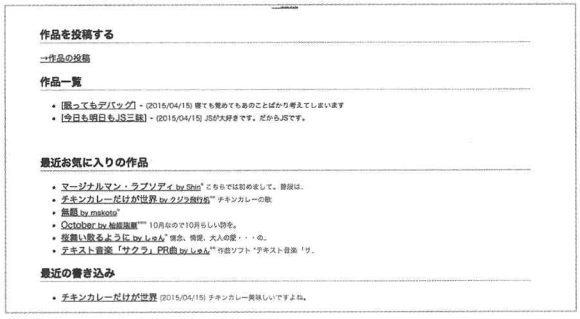
では、作詞掲示板でテストを行うために、アカウントを作ってみましょう。画面右上の [ログイン] ボタンを押して、続けて「→新規ユーザー登録」リンクをクリックすると、新規ユーザーが作成できます。普通の会員制サイトと同じ作りになっていますので適当なユーザーを作成してください。

ここでは、以下のテスト用アカウントを作成しました。アカウント作成が面倒な方は、以下のアカウントで試してみてください (ただしこのアカウントで書き込んだデータは定期的に削除されるのでご注意ください)。

テスト用アカウント	
ユーザー名	JS-TESTER
パスワード	ipCU12ySxl

CasperJS を使ってお気に入りを取得するプログラム

作詞掲示板では「マイページ」を開くと自分が投稿した作品に加えて、お気に入りの作品や書き込みが表示されるようになっています。



ログイン後のマイページ

CasperJS を利用して、作詞掲示板にログインして、お気に入りの一覧を取得するプログラムを作ってみましょう。

ログインして、マイページにアクセスするまでの手順は、次のようになります。

```
作詞掲示板のログインページを開く
    ↓
フォームにユーザー名とパスワードを設定して送信
    ↓
マイページの URL を取得
    ↓
マイページにアクセス
```

ログインしてマイページを開くだけですが、それなりに手順が必要となります。では、先にプログラムの雰囲気をつかむために全体を見てみましょう。

● file: src/ch03/02-login/getfav.js

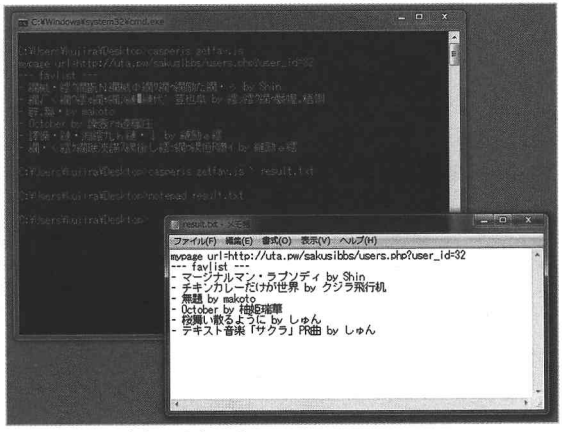
```
/// お気に入りの作品を取り出す for CasperJS

// 作詞掲示場のユーザー名とパスワード
var BBS_USER = "JS-TESTER";
var BBS_PASS = "ipCU12ySxI";

// CasperJS を使えるようにする
var casper = require('casper').create();
casper.start();

// 作詞掲示板のログインページを開く ---- (※1)
casper.open("http://uta.pw/sakusibbs/users.php?action=login");

// ログインする ---- (※2)
casper.then(function(){
  // フォームにユーザー名とパスワードを設定して送信
  this.fill("form", {
    username_mmlbbs6: BBS_USER,
    password_mmlbbs6: BBS_PASS
  }, true);
});

// マイページを開く ----- (※3)
casper.then(function(){
  // マイページの URL を取得する関数を定義
  var getLink = function () {
    var q = document.querySelector('#header_menu_linkbar a');
    return q.href;
  };
  // ページ内で評価
  var mypage_url = this.evaluate(getLink);
  this.echo("mypage url=" + mypage_url);
  // マイページを開く
  this.open(mypage_url);
});

// マイページのお気に入りを抽出する ---- (※4)
casper.then(function(){
  // お気に入りの作品を取得する関数を定義
  var pickupFav = function () {

 後略
```

プログラムを実行するには以下のコマンドを入力します。

```
$ casperjs getfav.js
```

ただし、出力される文字コードがUTF-8であるため、Windowsのコマンドプロンプトでは文字化けしてしまいます。そこで、実行結果をテキストファイルに保存し、メモ帳で開いて結果を確認してみてください。Windowsの場合には次のようにコマンドを実行します。

```
> casperjs getfav.js > result.txt
> notepad result.txt
```



Windowsで実行したところ

お気に入り取得までのプログラムの流れ

作詞掲示板でお気に入りを取得するプログラムは、次のような流れとなっています。プログラムの大まかな部分を追ってみましょう。

前節で見たように、CasperJSを使う場合、start()メソッドの後、必要な処理に応じて、open()メソッドやthen()メソッドで逐次処理を記述していくようになっていました。「getfav.js」でも同じように、ユーザーがサイトにアクセスして行うのと同じ要領で作業を行うようになっています。まず。流れを確認してみましょう。

(1) 作詞掲示板のログインページを開く
↓
(2) フォームに値を記入してログイン
↓
(3) マイページを開く
↓
(4) マイページのお気に入り一覧を取得

では、ひとつずつ見ていきましょう。手順(1)のログインページを開く部分は、ログインフォームのあるページをopen()メソッドで開いているだけです。

そして、手順 (2) では、then() メソッドで、ログインページを開いた後の処理を指定しています。fill() メソッドを使って、ログインフォームに、ユーザー名とパスワードを設定しフォームを投稿します。改めて、ここで、fill() メソッドの書式を確認しておきましょう。

```
[ 書式 ] フォームに値を設定し投稿する
casper.fill( セレクタ , 値オブジェクト [, 投稿するか ])

セレクタ : フォームを特定するための CSS セレクタを指定
値オブジェクト : name と value 属性をオブジェクトで指定
投稿するか : true であれば、投稿 ( サブミット ) する
```

例えば、HTML で次のようなフォームがあるとします。

● file: src/ch03/02-login/testform.html

```
<form action="/post" id="post-form">
  <input type="text" name="title" />
  <textarea name="body"></textarea>
  <input type="radio" name="ftype" value="book" /> 予約
  <input type="radio" name="ftype" value="question" /> 質問
  <input type="radio" name="ftype" value="comment" /> コメント
  <input type="file" name="attachment"/>
  <input type="checkbox" name="agreement" /> 規約に同意します
  <input type="submit" value=" 投稿 "/>
</form>
```

これに値を設定してフォームを投稿する場合には、次のように記述します。

```
// フォームに値を設定して投稿する
casper.fill('form#post-form', {
  title: " 商品の質問 ",
  body: " 一度に 30 個買うと安くなりますか? ",
  ftype: "question",
  attachment: "/Users/kujira/photo.jpg",
  agreement: true
}, true);
```

このように、<input type="text"> や <input type="radio"> タグであれば、name 属性をオブジェクトのキーに、value 属性に値を指定します。<input type="checkbox"> の場合は、値に true か false を指定することができます。また、<input type="file"> であれば、フォームに添付ファイルを付けることもできます。この場合、添付ファイルのパスを指定することになっています。

次に、前ページ手順 (3) ですが、マイページを開く処理を記述しています。作詞掲示板では、users.php?user_id=XXX の形式で特定のユーザーの情報を表示できるようになっています。そのため、マイページの URL は一意ではありません。ただし、ログインすると画面の上部のナビゲーションバーに「マイページ」へのリンクが表示されます。そこで、このマイページに張られているリンクを調べると、マイページの URL がわかります。

そのため、ログイン後のページ内にある <a> タグを特定し、そこから URL を取得するプログラムを記述しなくてはなりません。必要なのは、作詞掲示板の HTML の中で、任意の JavaScript のコードを実行することです。それを実現するのが、casper.evaluate() メソッドです。このメソッドの書式は次のようになっています。

```
[書式] ページ内で特定のJavaScriptコードを実行する
casper.evaluate(関数 [, 引数1[, 引数2[, …]]])

関数： ページ内で実行する関数オブジェクト
引数： 関数に与える引数
```

今回のプログラムからevaluate()メソッドを使っている部分を抜粋してみましょう。

```
// マイページのURLを取得する関数を定義
var getLink = function () {
  var q = document.querySelector('#header_menu_linkbar a');
  return q.href;
};
// ページ内で評価
var mypage_url = casper.evaluate(getLink);
this.echo("mypage url=" + mypage_url);
```

ここでは、getLinkという関数を定義しています。この関数のスコープは、作詞掲示板のHTMLで実行されるJavaScriptのものとなっています。つまり、HTML内のdocumentオブジェクトも利用することができます。document.querySelector()メソッドを使うと、CSSセレクタを利用して、特定のDOMオブジェクトを取得することができます。ここでは、ナビゲーションバーにある<a>タグで先頭にあるもの（マイページへのリンク）を取得し、そのhref属性を取得して戻り値として返します。「casper.evaluate(getLink)」を実行すると、HTMLページ内で関数getLinkを実行し、その結果を得ることができます。結果、上記のコードが実行されると、マイページのURLが得られるという訳です。

そして、103ページの手順(4)では、マイページを開き、そこに記述されているお気に入り作品を取得します。どうやってページ内のデータを取得したら良いでしょうか。そうです、先ほどと同じように、casper.evaluate()メソッドを使って、特定のDOM要素を取り出し、結果をCasperJSのメインスレッド側で受け取り、コンソール画面に出力するのです。

特定ユーザーの作品全部をお気に入りにする

最後に、せっかく、CasperJSを使うのですから、これを使って、データ収集とはちょっと違う面白いプログラムを作ってみましょう。

特定ユーザーの作品を調べて、それを全部をお気に入りにするという大胆なプログラムです。例えば、そのユーザーの作品が10個あれば、10個の作品全部をお気に入りに追加するというものです。

このプログラム自体は、作詞掲示板専用で汎用性はないのですが、ここで使われているテクニックを応用すれば、他のSNS用のプログラムを作ることができるでしょう。例えば、Facebookで特定ユーザーの発言すべてに「いいね」を付けるということもできるようになります。

それでは、作詞掲示板用の全作品お気に入り追加のプログラムを見てみましょう。

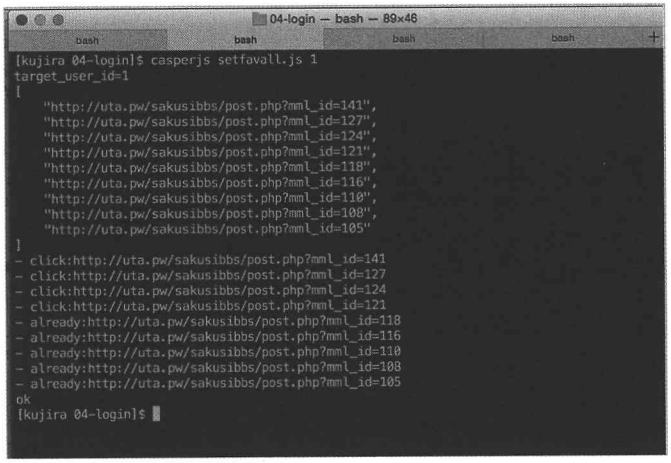
● file: src/ch03/02-login/setfavall.js

```
// あるユーザーの作品をすべてお気に入りに for CasperJS

// --- 設定 ---
// 作詞掲示板のユーザー名とパスワード
var BBS_USER = "JS-TESTER";

中略
```

```
// まずログインする ---- (※1)
casper.open("http://uta.pw/sakusibbs/users.php?action=login");
casper.then(function(){
  // フォームにユーザー名とパスワードを設定して送信
  this.fill("form", {
    username_mmlbbs6: BBS_USER,
    password_mmlbbs6: BBS_PASS
  }, true);
});

// 指定ユーザーのページを開く --- (※2)
casper.thenOpen(
  "http://uta.pw/sakusibbs/users.php?user_id=" +
  target_user_id);

// 作品一覧を取得する ---- (※3)
casper.then(function (){
  // ページ内で実行する関数を定義
  var getList = function () {
    var links = [];
    var list = document.querySelectorAll("ul#mmlist a");
    for (var i = 0; i < list.length; i++) {
      var a = list[i];
      // 作品ページへのリンクかどうかを確認
      if (a.href.indexOf('post.php') > 0) {
        links.push(a.href);
      }
    }
    return links;
  };
  // ページ内で関数を実行し作品一覧を列挙 --- (※4)
  var links = this.evaluate(getList);
  utils.dump(links);
  // 各リンクをすべて処理する
  casper.each(links, function(self, link){
    // 作品ページを開く --- (※5)

self.thenOpen(link, function(){
      // お気に入りに追加ボタンがあればクリック
      if (this.exists('#fav_add_btn')) {
        this.mouseEvent('click', '#fav_add_btn');
        this.echo('- click:' + link);
      } else {
        this.echo('- already:' + link);
      }
    });
  });
});
// 最後にメッセージを表示
casper.then(function(){
  this.echo('ok');
});
casper.run();
```

プログラムを実行するには、コマンドラインから次のようなコマンドを実行します。以下は、ユーザー ID が 1 のユーザーの作品をすべてお気に入りにする場合の実行例です。

```
$ casperjs setfavall.js 1
```



プログラムを実行したところ

プログラムを実行すると、次のように、お気に入りマーク（作品名の後の＊マーク）がすべての作品に付いていることがわかることでしょう。

作品一覧

- [リトマス紙と私]* - (2015/01/08) 人に流されず自分で決めることが重要です
- [サラダよ皿に乗れ]* - (2014/10/11) なかなか人と同じ道に進むのは難しいものです
- [ずーっとるーむ] - (2014/10/11) どうして人は引きこもるのか
- [チキンカレーだけが世界]** - (2014/09/13) チキンカレーの歌
- [夕焼けと船]* - (2014/08/26) あの頃はゆっくり船をずっと見る時間があったけど、今は・・・
- [逃亡]* - (2014/08/24) 実験室から逃げるマウス。逃げ切れるのか
- [ゆうひ]* - (2014/08/21) 海辺の夕日を見て
- [おもちゃ箱ガラガラ行進曲の歌詞]* - (2014/08/19) 動画URLの投稿テストです
- [海辺でゆったり]** - (2014/08/21) 海辺でゆったりしているところを思い出して作りました。

お気に入りマークがすべての作品に付いた

それでは、プログラムを見ていきましょう。ところで、今回はデバッグのために、utils モジュールを取り込んでいます。utils モジュールには、JavaScript のオブジェクトの内容を列挙する便利な dump() メソッドが用意されています。

```
// utils モジュールの読み込み
var utils = require('utils');
// 変数 obj の内容をダンプする
utils.dump(obj);
```

プログラムの(※1)を見てみると、ログイン画面を表示し、ログイン処理を実行しています。この部分は、先ほどのプログラムと同じです。

次に、プログラムの(※2)ですが、ここでは、指定のユーザーのページを開いています。thenOpen()メソッドを利用しています。このメソッドは名前の通り、then()メソッドの中でopen()メソッドを呼び出すのと同じ動作を行います。

プログラムの(※3)の部分で、作品一覧を取得しています。実際に作品一覧のデータの取得は、ページ内で行った方が早いので、casper.evaluate()メソッドを使っています。ここで定義しているgetList()関数が実際に作品一覧を取得していますが、CSSクエリの「ul#mmlist a」で作品へのリンク一覧を取り出すことができます。ただ、取り出した<a>タグが本当に作品ページへのリンクかどうかわからないので確認してから結果を保持する変数linksに追加しています。

プログラムの(※4)では、作品一覧へのリンクを全て処理するために、casper.each()メソッドを利用しています。このメソッドを利用することで、配列要素の数だけ、コールバック関数に指定した処理が実行されます。

プログラム(※5)のコールバック関数の中では、作品ページを開いて、そこにある「お気に入りに追加」ボタンをクリックします。CasperJSでは、exists()メソッドを使うことで、特定の要素があるかどうか調べることができます。引数にCSSセレクタを指定して、その要素があればtrueを返し、なければfalseを返します。

```
[書式] 特定のDOM要素があるか調べる
casper.exists(セレクタ)
```

便利なことに、casper.mouseEvent()メソッドを使って、特定のボタンをクリックすることができるようになっています。

```
[書式] 特定のDOM要素に対してマウスイベントを実行する
casper.mouseEvent(タイプ, セレクタ)
- タイプ: mouseup, mousedown, click, mousemove,
          mouseover, mouseout のマウスイベントの種類を指定
- セレクタ: イベントを適用する要素をセレクタで指定
```

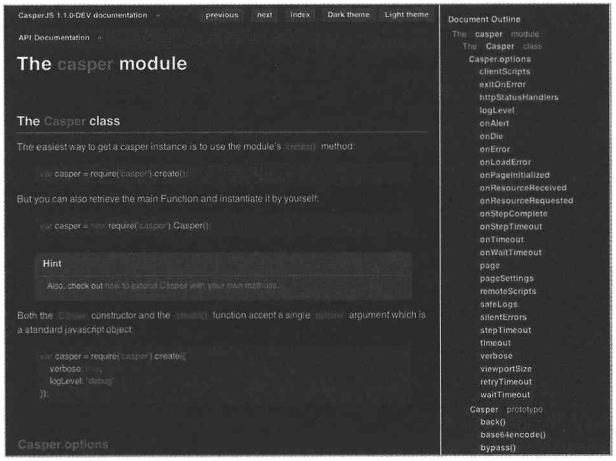
ここでは、「お気に入りに追加」ボタンに付与されているID「#fav_add_btn」があるかどうかを調べてクリックしています。

この節のまとめ

- CasperJSを利用して、ログインが必要なサイトからデータを取り出す方法を紹介しました。
- CasperJSを使えば、Webサイトを自由自在に操作できることがわかったのではないでしょうか。
- CasperJSには、まだまだいろいろな機能があります。CasperJSの基本と応用がわかれば、CasperJSのAPIマニュアルが役立つでしょう。

※参考※

CasperJSのAPI Documentation
http://casperjs.readthedocs.org/en/latest/modules/index.html



CasperJS API マニュアル

03

DOM解析手法とCSSセレクタ

ここまで、Web サイトに対して任意の操作を行う方法を紹介してきました。いずれの操作も DOM 内の要素をセレクタを指定して特定する方法が用いられていました。ここでは、DOM 解析の手法を紹介します。

ここで学ぶポイント

- HTMLの構造を知る方法

ツールやライブラリの一覧

- Webブラウザのデバッグツール

Web ブラウザーの開発者ツールを使う

HTML の構造を知る手っ取り早い方法は、おそらく、Web ブラウザに用意されている開発者ツールを使うことではないでしょうか。

Google Chrome で解析したい Web ページを開き、右クリックして表示されるコンテキストメニューで「要素の検証」を選ぶと、デベロッパーツール（開発者ツール）が開きます。

Chrome で開発者ツールを開く

デベロッパーツールの左上にある虫眼鏡のアイコンをクリックしてから、ページ上の調査したい要素をクリックすると、その場所の DOM 要素がフォーカスされます。

例えば、前節で作ったプログラムで、「マイページ」へのリンクについて詳しく調べたいと思った場合、虫眼鏡アイコンをクリックした後で、「マイページ」のリンクをクリックすると、デベロッパーツールに、要素の詳しい情報が表示されます。

特定の要素をクリックしたところ

ここでは、 というタグの下にある <a> タグであるという情報が得られました。そうであれば、次のようなJavaScriptのコードで、この <a> タグの要素を取得することができます。

```
var a = document.querySelector(
  "#header_menu_linkbar a");
```

CSSセレクタのクエリについて

せっかくの機会なので、改めて、document.querySelector() メソッドについて紹介しておきます。

［書式］特定のDOM要素を返す
```
document.querySelector(セレクタ)
```

ここでは、引数として与えられたCSSセレクタにマッチする文書中の最初の要素を返します。もし、マッチする要素がない場合、nullが返ります。

引数に与えるのがCSSセレクタというのがポイントです。例えば、 という要素を取得したければ、「#hoge」というCSSセレクタを指定すれば、要素を特定できます。また、 であれば、「.fuga」というCSSセレクタを指定すれば良いことになります。

ところで、id属性が付与された要素は、HTMLの中でおそらく一つだけと思われますが、class属性が付与された要素は複数あることでしょう。このように、複数の要素を取得したい場合には、document.querySelectorAll() メソッドを使います。

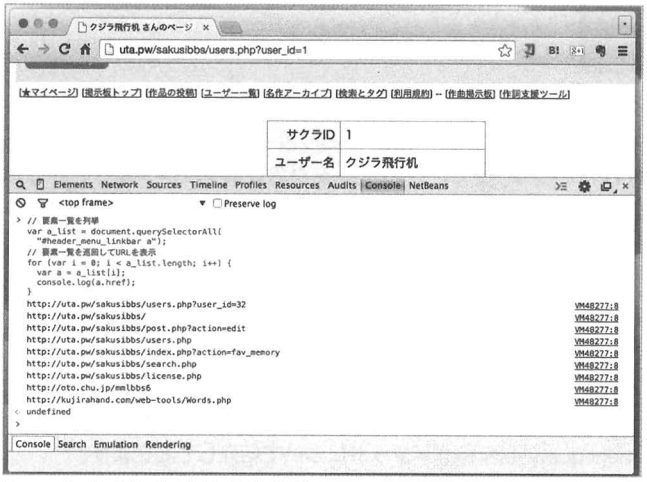
［書式］特定のDOM要素一覧を返す
```
document.querySelectorAll(セレクタ)
```

使い方は、querySelector() と同じなのですが、複数の値が戻り値に得られます。先のプログラム例で、ナビゲーションバー以下にある複数の <a> タグを全て取得して、そのURLを得るには、次のようなプログラムを書きます。

```
// 要素一覧を列挙
var a_list = document.querySelectorAll(
  "#header_menu_linkbar a");
```

```
// 要素一覧を巡回して URL を表示
for (var i = 0; i < a_list.length; i++) {
  var a = a_list[i];
  console.log(a.href);
}
```

Chrome のデベロッパーツールで、[Console] タブを開くと、そのページ内で任意のスクリプトを実行することができます。上記のプログラムを実行してみると、正しくナビゲーションバーからリンクされているページの URL 一覧が表示されました。



Console タブ内でスクリプトを実行

CSS セレクタの指定方法

ところで、Web サイトのスクレイピングを上手に行う上で重要となるのは、CSS セレクタをどのように記述するのかという部分です。エレガントな CSS セレクタを書けば、一発で特定の要素を取り出すことができて、非常に便利です。

ここでは、CSS セレクタについて少し理解を深めておきましょう。CSS セレクタとして指定可能な書式を確認してみます。

書式	説明
*	すべての要素
要素名	要素名の要素 (例) p div
. クラス名	クラス名をつけた要素
#id 名	id 属性をつけた要素

セレクタの基本書式

書式	説明
セレクタ , セレクタ	列挙された複数のセレクタ (例) h1, h2
セレクタ セレクタ	下の階層の子孫要素 (例) div h1
セレクタ > セレクタ	直下の階層の子要素 (例) div>h1
セレクタ + セレクタ	同じ階層で直後に隣接している要素 (例) h1+h2
セレクタ 1~ セレクタ 2	セレクタ 1 からセレクタ 2 までの要素 (例) p~ul

セレクタ同士の関係を指定する書式

書式	説明
要素 [att]	特定の属性名を持つ要素
要素 [att="val"]	att 属性に val という値を持つ要素
要素 [att~="val"]	att 属性の値候補 val(ホワイトスペース区切り) に一致した要素
要素 [att | ="val"]	att 属性の値が val で始まる要素 (但しハイフン区切り)
要素 [att^="val"]	att 属性の値が val で始まる要素
要素 [att$="val"]	att 属性の値が val で終わる要素
要素 [att*="val"]	att 属性の値に val を含む要素

セレクタの属性による指定書式

書式	説明
要素 :root	ルートとなる要素
要素 :nth-child(n)	n 番目の子となる要素
要素 :nth-last-child(n)	後ろから n 番目となる要素
要素 :nth-of-type(n)	n 番目のその種類の要素
要素 :first-child	子として最初の要素
要素 :last-child	子として最後の要素
要素 :first-of-type	最初のその種類の要素
要素 :last-of-type	最後のその種類の要素
要素 :only-child	子として唯一の要素
要素 :only-of-type	子として唯一の種類の要素
要素 :empty	要素内容が空となる要素
要素 :lang(code)	特定の言語に code を指定された要素
要素 :not(s)	s 以外の要素
要素 :enabled	有効となっている UI 要素
要素 :disabled	無効となっている UI 要素
要素 :checked	チェックされている UI 要素

位置や状態を指定する書式

このように、非常に豊富な書式で記述できるようになっています。

CSS セレクタ実践編

それでは、CSS セレクタ実践編として、実際の HTML コードを見て、特定の要素を選び出すプログラムを考えてみましょう。

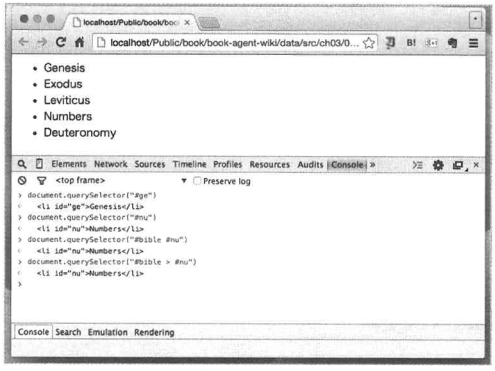
例えば、次のように記述されている HTML 文書の要素について考えてみましょう。

● file: src/ch03/03-dom/test-id.html

```
<ul id="bible">
  <li id="ge">Genesis</li>
  <li id="ex">Exodus</li>
  <li id="le">Leviticus</li>
  <li id="nu">Numbers</li>
  <li id="de">Deuteronomy</li>
</ul>
```

この HTML から <li id="nu">Numbers の要素を取り出したいと思います。どんな CSS クエリを書けばよいのでしょうか。

```
document.querySelector("#nu")
document.querySelector("#bible #nu")
document.querySelector("#bible > #nu")
document.querySelector("li[id='nu']")
document.querySelector("#bible li:nth-child(4)")
```



さまざまな指定で要素を取り出した

まず、id 属性ならば、他とあまり被らないので「#nu」が単純で簡単な指定方法です。

親要素と合わせて「#bible #nu」や「#bible > #nu」と指定するなら、より確実に要素を特定できると言えるでしょう。なお、この両者の違いですが、「#bible > #nu」の場合、親子関係が直下である場合に限り合致し、「#bible #nu」の場合は親子だけでなく孫以下の関係も含むという点が異なります。

「li[id='nu']」の指定は、 要素で id が nu であるものという指定になり、つまり「li#nu」と同じ意味になります。

それから「#bible li:nth-child(4)」というのは、#bible 以下で 4 つ目の <li> という意味になります。

CSS セレクタ問題

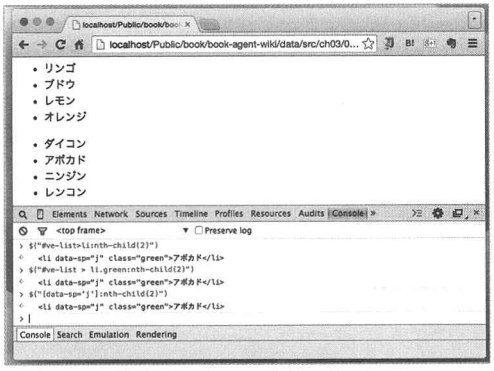
次の問題です。まずは、対象となる HTML をご覧ください。

● file: src/ch03/03-dom/test-attr.html

```
<div id="main-goods" role="page">
  <h1>フルーツや野菜</h1>
  <ul id="fr-list">
    <li data-sp="j" class="red green">リンゴ</li>
    <li class="purple">ブドウ</li>
    <li class="yellow">レモン</li>
    <li class="yellow">オレンジ</li>
  </ul>
  <ul id="ve-list">
    <li data-sp="b" class="white green">ダイコン</li>
    <li data-sp="j" class="green">アボカド</li>
    <li class="red green">ニンジン</li>
    <li data-sp="b" class="white">レンコン</li>
  </ul>
</div>
```

この中で「アボカド」を抜き出したいのですが、どのようなセレクタを書けばよいのでしょうか。以下の答えを見ないで挑戦してみてください。なお、長くなるため「document.querySelector() メソッド」を「$()」と省略して書いているので注意してください。

```
// #ve-list の直下 li タグの 2 つ目 ---(※ 1)
$("#ve-list>li:nth-child(2)")

// #ve-list の直下、li タグでクラス green の 2 つ目 --- (※ 2)
$("#ve-list > li.green:nth-child(2)")

// data-sp 属性が "j" である 2 つ目 --- (※ 3)
$("[data-sp='j']:nth-child(2)")
```



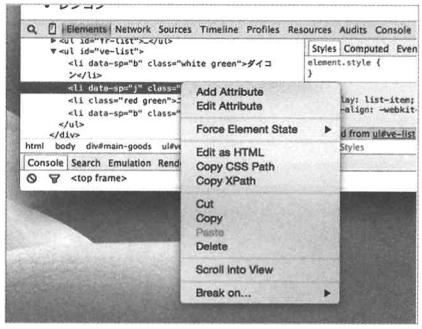
アボカドの要素を取り出してみたところ

どうでしょうか。親要素に id が付いているので、#ve-list 直下 <li> タグの 2 つ目という (※ 1) の指定方法が一番、素直でしょうか。クラス名が green であることを利用して、.#ve-list 直下クラス名 green の二つ目

という (※ 2) の指定も正確に取り出せそうです。そして、data-sp 属性という属性値に注目して、その 2 つ目の要素という (※ 3) の指定も考えられますね。

実は、Chrome であれば、要素を選択した上で、コンテキストメニューの「Copy CSS Path」を実行すると、そのものずばりの CSS セレクタを、クリップボードにコピーしてくれます。

ちなみに、先ほどの問題「アボカド」を選択して、Copy CSS Path を試してみたところ「#ve-list > li:nth-child(2)」という値がコピーされました。



Copy CSS Path でセレクタをコピーできる

この節のまとめ

- この節では、DOM を調べる方法や、CSS セレクタについて紹介しました。
- Web ブラウザの開発者ツールを使って、任意の要素を素早く特定できるかどうかが、スクレイピングの明暗を分けます。
- CSS セレクタの指定方法に精通しておきましょう。

04

Electronで デスクトップアプリを作る

画面のスクリーンショットを撮影するなど、Webの高度な操作を行うのに、Electronも使うことができます。これを使うとWebの技術を利用して気軽にデスクトップアプリを作ることができます。PhantomJS/CasperJSを使わないでWebの自動操作ができる環境として、Electronを利用したアプリの作り方を紹介します。

ここで学ぶポイント

- Electronについて
- インストールと基本的なプログラムの作り方

ツールやライブラリの一覧

- Electron

Electronとは？

Electron(旧:atom-shell)とは、GitHubが公開している次世代テキストエディター「Atom」に使われているライブラリです。Electronを利用すると、JavaScriptを利用してクロスプラットフォームなデスクトップアプリケーションを作ることができます。

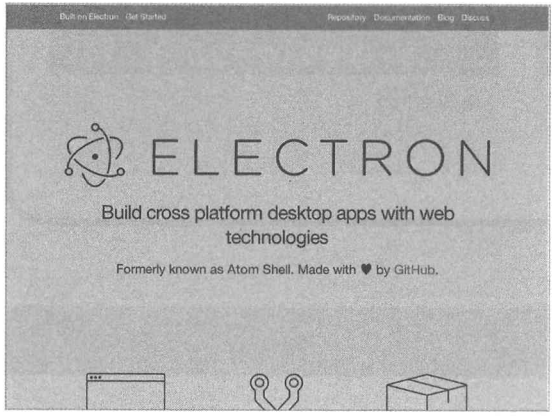
わかりやすくいえば「Webブラウザーの中にNode.jsを取り込んだもの」と言うこともできるでしょう。実際、ElectronはWebブラウザーのChromium*とNode.jsを組み合わせて作られています。そのため、Node.jsの機能に加えてWebブラウザーの機能を利用することができます。

ところで、Electronの実力は、Atomエディターを使ったことのある人ならば納得できるものです。(Atomエディターに関しては、1章でも紹介しました。) Atomエディターは Webブラウザーの機能をベースに作られているので、CSSを利用して手軽にデザインを変更できるなど、デザイナーにも優しい作りになっています。

---

* Chromium（クロミウム）はオープンソースのWebブラウザーです。Google Chromeはこのソースコードを基にして開発されています。

GitHub の Atom エディター



Electron の Web サイト

```
Electron（旧：atom-shell）
[URL] http://electron.atom.io/
[URL] (GitHub) https://github.com/atom/electron
```

類似ライブラリの「NW.js」

類似ライブラリに「NW.js(元 node-Webkit)」というライブラリもあります。これも、Chromium に Node.js(io.js) を組み合わせたものです。

```
NW.js
[URL] http://nwjs.io/
```

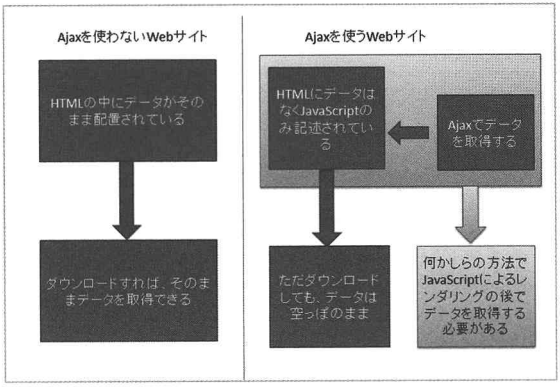
Electron のメリット・デメリット

Electron を使う最大のメリットは、Web の技術 (HTML5/JavaScript/CSS) を利用してネイティブアプリを手軽に作れることです。Node.js の API やモジュールをそのまま利用できるというのも大きなメリットです。

ただし、Web ブラウザーの Chromium をそのまま梱包していることになるので、配布サイズは大きく、どんなに簡単なアプリであったとしても、数十〜数百 MB のストレージ領域が必要となります。

JavaScript でレンダリングされるページも OK

ところで、本書で Electron を紹介するのはデスクトップアプリの作り方を学習するためではありません。最近では、JavaScript などでレンダリングがゼロから行われる高度な Web ページも増えています。そのため、HTML をただダウンロードしただけでは、内容が空っぽであるということも少なくありません。



JavaScript でレンダリングされるページについて

その点、Electron を利用すると、JavaScript によるレンダリングが終了した後で、そのコンテンツを取り出すことができるようになります。これは、大きなメリットと言えるでしょう。

Electron のインストール方法

Node.js の npm を利用して、Electron をインストールすることができます。このとき、仮想マシンに CentOS をインストールしている場合、デスクトップ画面 (X Window) をインストールする必要がありますが、インストールが大変なので、ここでは、仮想マシンを使わず、Mac OS X か Windows を使ってみてください。

```
$ npm install electron-prebuilt -g
```

また、Electron は GitHub のページより、バイナリを直接ダウンロードすることもできます。

```
https://github.com/atom/electron/releases
```

Electronで一番簡単なアプリ作成手順

はじめに、Electronで一番簡単なプログラムの作成手順を紹介します。Electronのプログラムでは、最低限、アプリの設定ファイルとメインプログラムの二つのファイルを用意します。そして、そのファイルを一つのディレクトリに配置します。また、Windowsでは、Electronの実行ファイルにディレクトリをドラッグ・アンド・ドロップすることで実行できます。

手順 1: アプリに必要なファイルを用意する

では、実際に作成してみましょう。

まず、「easy-app」というディレクトリを作成してください。(実際のところ名前は何でも良いのですが、ここではeasy-appにしました。) そして、そのディレクトリ以下に、設定ファイルを作成します。「package.json」という名前で次のような内容のテキストファイル(JSONファイル)を用意します。

● file: src/ch03/04-atom-shell/easy-app/package.json

```
{
  "name"    : "easy-app",
  "version" : "0.1.0",
  "main"    : "main.js"
}
```

次に、同じく「easy-app」ディレクトリ以下に、メインプログラムのプログラムを作成します。「main.js」という名前で保存します。

● file: src/ch03/04-atom-shell/easy-app/main.js

```
// Wikipediaを表示するだけのプログラム for Electron
var TARGET_URL = "https://ja.wikipedia.org/";

// 必要モジュールの取り込み
var app = require('app');
var BrowserWindow = require('browser-window');

// メインウインドウを起動
var mainWindow = null;
// 準備ができたタイミングで呼ばれるイベント
app.on('ready', function(){
  // メインウィンドウを作成
  mainWindow = new BrowserWindow({width:800, height:600});
  // 指定のURLを読み込み
  mainWindow.loadUrl(TARGET_URL);
});
```

ここで用意したファイル一覧

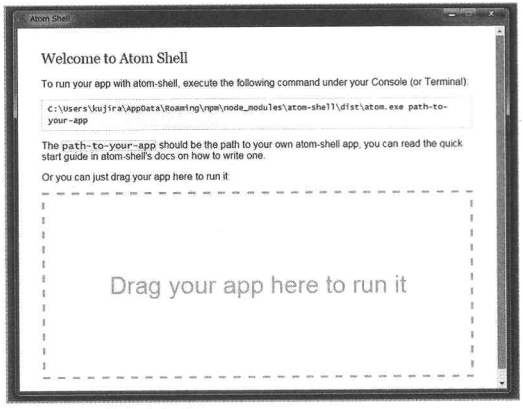
手順 2:Electron にパラメーターを与えて実行する

これで準備は完了です。さっそく実行してみましょう。npm 経由でインストールしたのであれば、「electron」コマンドが利用できるようになっています。次のような書式でプログラムを実行できます。

```
$ electron /path/to/easy-app/
```

Windows であれば、Electron の実行ファイルである「atom.exe」に「test-app」のフォルダをドラッグ・アンド・ドロップすることでも実行できます。

また、Electron コマンドを実行すると、次のようなウィンドウが表示されるので、そこにアプリのディレクトリをドラッグ・アンド・ドロップすることでも実行できます。



atom.exe を実行した

プログラムを実行すると、次のように、Electron の画面の中に Wikipedia が表示されます。

「easy-app」を実行してみた

自分で用意したHTMLをElectronに表示する

もちろん、Electronでは、外部のWebサイトを表示するだけでなく、自分で用意したHTMLファイルを表示することができます。

今回は、次のようなファイル構成でファイルを準備しましょう。

```
test-app/
  |-- package.json
  |-- main.js
  |-- index.html
```

「test-app」のために準備したファイル一覧

はじめに全体の設定を「package.json」に記述します。

● file: src/ch03/04-atom-shell/test-app/package.json

```
{
  "name"    : "test-app",
  "version" : "0.1.0",
  "main"    : "main.js"
}
```

次に、メインプログラムである「main.js」のコードを記述します。このファイルには、メインウィンドウを表示したり、システムイベントを記述したりするなどの処理を記述します。

● file: src/ch03/04-atom-shell/test-app/main.js

```
// モジュールの取り込み
var app = require('app');
var BrowserWindow = require('browser-window');

// メインウインドウを起動 --- (※1)
var mainWindow = null;
app.on('ready', function(){
  mainWindow = new BrowserWindow({width:800, height:600});
  mainWindow.loadUrl('file://' + __dirname + '/index.html'); // ----(※2)
  mainWindow.on('closed', function(){
    mainWindow = null;
  });
});
```

今度は、少しプログラムについて解説します。

ここで注目したいのは、メインウィンドウを表示している (※1) の部分です。アプリの準備ができた時点で実行されるシステムイベントの「ready」のタイミングで、メインウィンドウを表示します。このとき、(※2) の部分では、メインウィンドウに表示する HTML として「index.html」を指定しています。そのため、メインウィンドウに、「index.html」の HTML を読み込みます。

次に、メインウィンドウから読み出される「index.html」を記述します。この HTML は、普通の HTML です。しかし、よく見ると、通常の HTML 内の JavaScript では使えない process オブジェクトが利用できるのが確認できます。

● file: src/ch03/04-atom-shell/test-app/index.html

```
<!DOCTYPE html>
<html><head>
<meta charset="UTF-8">
<title>Test App</title>
<script>
  window.onload = function () {
    var info = document.getElementById("info");
      info.innerHTML =
        "Node ver." + process.version + "<br>" +
        "atom ver." + process.versions['electron'];
  };
</script>
</head>
<body>
  <h1> テストアプリです </h1>
  <p id="info"></p>
</body>
</html>
```

プログラムを実行します。npm で Electron をインストールした場合には「electron」コマンドが利用できますので、以下のコマンドを実行します。

```
$ electron test-app
```

Windows で test-app を実行してみたところ

Mac OS X で test-app を実行してみたところ

メインプロセスとレンダラープロセスの通信

さて、ここまでの部分で、Electron で作るアプリの基本を押さえることができました。もう一度、プログラムが実行されるまでを整理すると、Electron では、package.json に書かれているメインプログラム (JavaScript) が実行されます。そして、メインプログラムの中で、ブラウザー画面を作成し、そこに HTML ファイルを読み込むというものです。
箇条書きにしてまとめてみましょう。

【Electron の基本的な流れ】

(1) **Electron が起動**
(2) **設定ファイル (package.json) に応じてメインプログラム (JavaScript) が実行される**
(3) **メインプログラムで、ブラウザーウィンドウを作成**
(4) **ブラウザーウィンドウに、任意の HTML を表示する**

ところで、アプリを作ろうと思った時、HTML ファイルの中でも、JavaScript を実行します。そうすると、メインプログラムと、ブラウザー内で実行される JavaScript と二つのプログラムが実行されることになります。この２つのプログラムは、異なるプロセス（コンテキスト）で実行されるという点がポイントです。

Electron では、前者のメインプログラムの方を「メインプロセス」と呼び、後者の実際に表示される HTML の中で実行されるプログラムを「レンダラープロセス」と呼んでいます。

なぜ、プロセスが２つに分かれているのでしょうか。

通常の Web ブラウザーでは、セキュリティを保護するサンドボックス内で HTML が実行されます。そのため、通常、PC 内のファイルなどローカルなリソースにアクセスすることは禁止されています。これは、

Electron においても同様で、HTML を実行するレンダラープロセスで実行される HTML からは危険な操作を行うことはできません。

しかし、メインプロセスからは、Node.js の API を自由に呼び出すことができるようになっています。そのため、レンダラープロセスからメインプロセスに対して、必要に応じた処理を依頼することによって、これまでの Web ブラウザーでは実現できなかった各種の処理を行うことができるようになります。メインプロセスとレンダラープロセスの間で通信を行うために、IPC というモジュールが提供されています。

それでは、レンダラープロセスからメインプロセスへ二つの値を送信し、メインプロセスではかけ算を行って、レンダラープロセスへ計算結果を返すプログラムを見てみましょう。

同期的な IPC 通信

IPC 通信には、同期的な通信と、非同期的な通信という２つの通信方法があります。同期的な通信が簡単なので、その方法から見ていきましょう。

まずは、ブラウザー側のレンダラープロセスのコードから見ていきましょう。sendSync() メソッドを利用して、メインプロセス側に値を送信します。「mul-sync」というチャンネルに対して、JavaScript のオブジェクトでかけ算に使う a と b の二つの値を送信します。このとき、送信する値は文字列でも JavaScript のオブジェクトでも大丈夫です。

```
<!-- ブラウザー側（レンダラープロセス） -->
<script>
// IPC モジュールを読込
var ipc = require('ipc');

// 同期的にメインプロセスに対して値を二つ送信し結果を得る
var res = ipc.sendSync('mul-sync', {a:30, b:2});
alert("res=" + res);
</script>
```

次に、メインプロセス側です。こちら側では、ipc.on(チャンネル名 ,..) でイベントハンドラーを設定して、レンダラープロセスから通信があるのを待機します。ここでは、「mul-sync」というチャンネルメッセージを受け取った時に処理するイベントハンドラーを定義しています。event 引数の returnValue プロパティに返却値を設定することで、レンダラープロセス側の呼び出しに値を返すことができます。

```
// IPC モジュールを読込
var ipc = require('ipc');

// 同期メッセージの受信
ipc.on('mul-sync', function(event, arg) {
  console.log(arg); // コンソールに表示
  event.returnValue = arg.a * arg.b;
});
```

非同期的な IPC 通信

同期的な通信の方法がわかれば、非同期的な通信もほとんど同じです。ただし、非同期的な通信では、即時、任意の値を返すことができません。そのため、処理が完了した時点で結果を送信するという方法で通信を行います。

まずは、ブラウザー側のレンダラーの処理を見ていきましょう。同期通信では、send() メソッドを利用します。非同期通信では、すぐにメインプロセスからの結果を得られないので、ipc.on(チャンネル名 ,..) でイベントハンドラーを設定して、メインプロセスから結果が送信されてくるのを待つようにします。

```
// IPC モジュールを読込
var ipc = require('ipc');

// メインプロセスに引数を送信
ipc.send('mul-async', {a:30, b:2});
// 非同期通信の結果を受けたとき
ipc.on('mul-async-reply', function(arg) {
  alert("res=" + arg);
});
```

そして、メインプロセス側のプログラムです。こちらでは、メッセージを受信し、計算を行って、その結果を、send() メソッドで返信するという流れです。

```
// IPC モジュールを読込
var ipc = require('ipc');

// メッセージの受信イベント
ipc.on('mul-async', function(event, arg) {
  console.log(arg);// コンソールに表示
  // レンダラープロセスへ返信
  var result = arg.a * arg.b;
  event.sender.send('mul-async-reply', result);
});
```

IPC 通信を行う実際のプログラム

では、再度、完全なプログラムで確認してみましょう。先のサンプルで行っているように、かけ算を行うだけのプログラムです。
プロジェクトのファイル構成は次の通りです。

```
ipc-app/
  | -- pakcage.json
  | -- main.js
  | -- index.html
```

pc-app のファイル一覧

まずは、メインプロセスのプログラムです。

● file: src/ch03/04-atom-shell/ipc-app/main.js

```
前略

// メインウインドウを起動
var mainWindow = null;
app.on('ready', function(){
  mainWindow = new BrowserWindow({width:800, height:600});
  mainWindow.loadUrl('file://' + __dirname + '/index.html');
  mainWindow.on('closed', function(){
    mainWindow = null;
  });
});

// 同期メッセージの受信
ipc.on('mul-sync', function(event, arg) {
  console.log(arg); // コンソールに表示
  event.returnValue = arg.a * arg.b;
});

// 非同期メッセージの受信
ipc.on('mul-async', function(event, arg) {
  console.log(arg);// コンソールに表示
  // レンダラープロセスへ返信
  var result = arg.a * arg.b;
  event.sender.send('mul-async-reply', result);
});
```

次に、レンダラーのプログラムです。

● file: src/ch03/04-atom-shell/ipc-app/index.html

```
前略

// 同期的に通信を行う
function testSync() {
  // メインプロセスに引数を送信して答えを得る
  var res = ipc.sendSync('mul-sync', {a:30, b:2});
  msg("sync result=" + res);
}

// 非同期に通信を行う
function testASync() {
  // メインプロセスに引数を送信
  ipc.send('mul-async', {a:30, b:2});
  // 非同期通信の結果を受けたとき
  ipc.on('mul-async-reply', function(arg) {
    msg("async result=" + arg);
  });
}

function msg(msg) {
  info.innerHTML += msg + "<br>";
}
</script></head><body>
  <h1>IPC のテスト </h1>
  <p id="info"></p>
</body></html>
```

それでは、プログラムを実行してみましょう。

```
$ electron ipc-app
```

ipc-app を実行してみたところ

この節のまとめ

- この節では、簡単に Electron の使い方について紹介しました。
- プログラムの仕組みを紹介するための一番簡単なサンプルなので地味なものでしたが、Electron を使うと、簡単にクロスプラットフォーム対応のデスクトップアプリが作成できることがわかりました。

05 Electronでスクリーンキャプチャ

前節では、Electron について基本的な事柄を紹介しました。本節ではもう少し Electron に親しみましょう。ここで作るのは、Web サイトのスクリーンショットを撮るというアプリです。

ここで学ぶポイント	ツールやライブラリの一覧
●Webページのスクリーンショットを撮る方法	●Electron

スクリーンショットを撮る最速の方法

Web サイトのスクリーンショット（画面キャプチャ）を撮る方法は、いろいろあります。ただ、画面が欲しいだけならば、Web ブラウザでページを表示して、キャプチャ専用ソフトや OS のコマンドを利用して、スクリーンショットを撮るのが一番簡単で早いでしょう。

しかし、プログラムからスクリーンショットを撮ることができれば、プログラムのテストや、定期的な自動撮影など、便利な場面がたくさんあります。

ここでは、Electron を利用したスクリーンショットを撮る方法を紹介します。しかし、よりプログラムが簡単なのは、この章の最初で説明した PhantomJS/CaspserJS を使う方法です。そちらも参考にしてみてください。

Electron でスクリーンショットを撮る方法

では、はじめに、スクリーンショットを撮るための方法を確認しておきましょう。Electron では、このために、BrowserWindow.capturePage() メソッドが用意されています。ですので、これを利用してブラウザ画面をキャプチャすることができます。

```
// ブラウザを作成
win = new BrowserWindow();
// ...
// ブラウザ画面をキャプチャする
win.capturePage(function(img) {
  var png = img.toPng();
  fs.writeFileSync('screenshot.png', png);
});
```

capturePage() メソッドの引数に、キャプチャが完了した際に行う処理を関数オブジェクトで指定します。そのコールバック関数では、引数にキャプチャしたデータを含むオブジェクトが得られるので、PNG 画像に変換した上で、ファイルに保存するという手順になります。しかし、Web サイトが読み込まれていないうちにキャプチャをとっても仕方がありません。そこで、Web サイトの読み込みが完了するタイミングを知る必要があります。

Electron では、Web ブラウザがページの読み込みが完了した際など、各イベントを取得できるようになっています。読み込みが完了した際には「did-finish-load」イベントが発生します。そこで、このイベントを利用して、読み込みが完了してから、スクリーンショットを撮るようにします。

```
// ブラウザを作成
var win = new BrowserWindow({width:1024, height:800});
// 指定のページを読み込む
win.loadUrl(TARGET_URL);
// ページがロード完了したらキャプチャ処理を実行する
win.webContents.on('did-finish-load',captureFunc);
```

スクリーンショットを撮るプログラム

では、実際の完全なプログラムで見てみましょう。Electron で実行するために、次のようなフォルダ構造でファイルを作成します。

```
screenshot-app
  | -- package.json
  | -- main.js
```

以下に示すのは、Electron の設定ファイルです。メインプログラムとして、main.js を指定しているところがポイントです。

● file: src/ch03/05-screenshot/screenshot-app/package.json

```
{
  "name"    : "screenshot-app",
  "version" : "0.1.0",
  "main"    : "main.js"
}
```

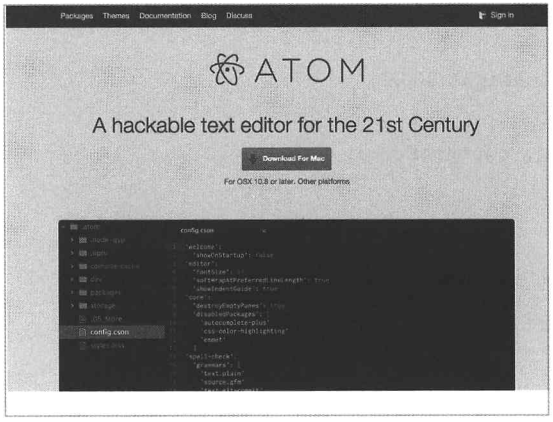
次に、メインプログラムを見ていきましょう。ポイントは、ページのロードが完了したタイミング（did-finish-load イベント）で、キャプチャ処理を実行している部分です。

● file: src/ch03/05-screenshot/screenshot-app/main.js

```
// 変数などの宣言
var TARGET_URL = "https://atom.io"; // 対象 Web サイト

// モジュールの取り込み
var app = require('app');
var BrowserWindow = require('browser-window');
var fs = require('fs');

// メインウインドウを起動
var win = null;
app.on('ready', function(){
  win = new BrowserWindow({width:1024, height:800});
  win.loadUrl(TARGET_URL);
```

```
  // ページがロード完了したらキャプチャする
  win.webContents.on('did-finish-load',captureFunc);
});

// キャプチャ処理
function captureFunc() {
  win.capturePage(function(img) {
    fs.writeFileSync('screenshot.png', img.toPng());
  });
}
```

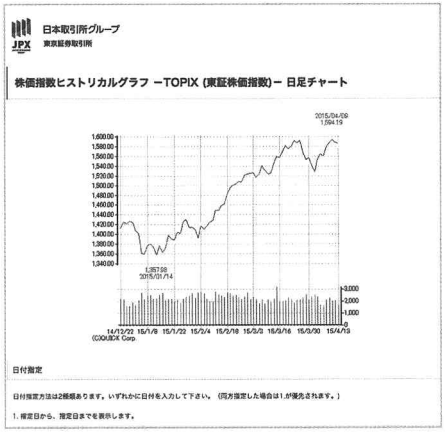
プログラムを実行すると、次のようなスクリーンショットが保存されます。



プログラムを実行したところ

毎朝株価チャートのスクリーンショットを保存する

2章で紹介した定期実行を行うcronなどと組み合わせれば、定期的に株価チャートのスクリーンショットを保存することもできます。例えば、東京証券取引所のページにある株価チャートを定期的に保存するプログラムを作ってみましょう。次のような画像を定期的にファイルに保存します。



株価チャートを定期的に保存する

プログラムは、先ほどとほとんど同じですが、違いを確認するために、全リストを見てみましょう。

● file: src/ch03/05-screenshot/kabu-chart-app/main.js

```
/// 東京証券取引所の株価指数グラフのページ
var TARGET_URL = "http://quote.jpx.co.jp/jpx/template/quote.cgi?F=tmp/hist_
index&basequote=151&mode=D";

// モジュールの取り込み
var app = require('app');
var BrowserWindow = require('browser-window');
var fs = require('fs');

// メインウインドウを起動
var win = null;
app.on('ready', function(){
  win = new BrowserWindow({width:800, height:800});
  win.loadUrl(TARGET_URL);
  // ページがロード完了したらキャプチャする
  win.webContents.on('did-finish-load', captureFunc);
});

// キャプチャ処理
function captureFunc() {
  // 日付付きのファイル名で保存する ---(※1)
  var t = new Date();
  var fname = "kabu-" + t.getFullYear() +
    "-" + (1 + t.getMonth()) +
    "-" + t.getDate() + ".png";
  win.capturePage(function(img) {
    fs.writeFileSync(fname, img.toPng());
    app.quit(); // アプリを自動終了
  });
}
```

まず、プログラムの冒頭でチャートを表示するURLを指定して、プログラムの(※1)でスクリーンショットを保存するファイルに日付を入れてみました。また、定期的に実行する時に便利なように、プログラムを自動で終了するように、app.quit()メソッドを入れています。

あとは、cron環境でもElectronが起動するように環境変数を設定するか、あるいは、起動用のバッチファイルなどを作っておくと良いでしょう。例えば、Mac OS X用に次のようなバッチファイルを作成し、cronに登録してみました。

「crontab -e」を実行し、毎朝7時12分にスクリーンショットを保存します。

```
12 7 * * * /path/to/kabu-chart-cron.sh
```

このように、既存サイトに表示されているグラフや画像などを定期的に保存します。

微調整のためのディレイを入れる

ところで、Web サイトによっては、今回のプログラムでうまくスクリーンショットを撮影できない場合もあります。did-finish-load イベントが発生したときに、まだ完全に画面の描画が完了していない場合があるのです。

そんなときには、did-finish-load イベントから撮影までの間に、若干の待ち時間を入れると良いでしょう。以下は、一秒のディレイ撮影を行うようプログラムを改良したものです。Google の画像検索でネコを検索し、その画面を画像ファイルに保存します。

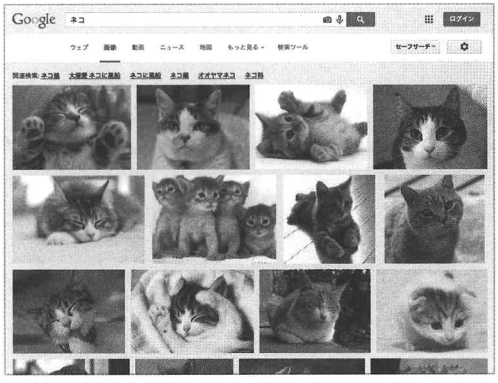
実現の方法ですが、通常の JavaScript と同じで、次のように、setTimeout() メソッドを利用します。

```
setTimeout(function() {
  //
  // ここで画面キャプチャを行う
  //
}, 2000);
```

では、実際のプログラムを確認してみましょう。このプログラムも、先ほどとプロジェクトの構成はほとんど同じなので、メイン JavaScript のみ紹介します。

● file: src/ch03/05-screenshot/delayshot-app/main.js

```
// 撮影までの待ち時間
var DELAY_TIME = 1000 * 1; // 1秒

中略

// キャプチャ処理 --- (※1)
function captureFunc() {
  // ディレイ処理をはさむ
  setTimeout(function () {
    // 適当な名前で保存する
    var fname = "cat-" + (new Date()).getTime() + ".png";
    win.capturePage(function(img) {
      fs.writeFileSync(fname, img.toPng());
      app.quit(); // アプリを自動終了
    });
  }, DELAY_TIME);
}
```



待ち時間を入れることでレンダリングの精度が向上する

特にプログラム(※1)の部分で、関数「captureFunc()」に注目してみてください。setTimeout() メソッドを利用して、一定時間時間を遅らせています。

キャプチャする範囲を指定する

capturePage() メソッドでは、画面キャプチャを行う領域を指定することができます。

```
[書式]
BrowserWindow.capturePage([rect, ]callback)
```

この「rect」オブジェクトにキャプチャする領域を指定します。

パラメーター	説明
x	左上座標 X
y	左上座標 Y
width	領域の幅
height	領域の高さ

引数 rect に指定するオブジェクトのプロパティ

Web サイトのロゴ部分や空白領域が不要な場合、領域を指定できると便利ですよね。

この節のまとめ

この節では、Electron を利用して、Web サイトのスクリーンショットを撮影する方法を紹介しました。

Web サイトのデータをダウンロードするよりも、サイトそのままを見た方が一目瞭然という場合もあります。

以下に、スクリーンショットを撮影する際の基本的なポイントを挙げておきます。

- **スクリーンショットを撮影するタイミングに注意**
- **Web サイトの準備が整った後まで待つようにする**
- **撮影が終わったらすぐプロセスを終了する**
- **did-finish-load イベントやディレイを入れて確実に描画が終わるまで待つことができる**

第4章

データの整形と保存

ここまで、Webからさまざまなデータのダウンロードや、スクレイピングの手法を紹介してきました。本章では、収集したデータを保存する方法について考えます。どんな形式のデータがあるのか、また、どのように保存するのかを紹介します。

01

データの文字コードと変換について

Web には英語や日本語だけではなく、さまざまな種類の言語データが流通しています。その言語のデータをどのように表記するのかもさまざまです。ここでは文字コードの問題について考えましょう。

ここで学ぶポイント

- 文字コードについて
- 文字コードの変換

ツールやライブラリの一覧

- node.js
- iconvモジュール
- iconv-liteモジュール
- jschardetモジュール
- Rhino

文字コードとは？

コンピューター上では、文字を表すために、文字に対応した固有の番号を利用します。文字に割り当てられた番号のことを文字コードといいます。文字をどのような方法で扱うかを定めた文字コード体系のことを、文字コード表と呼びます。

例えば ASCII コードで「A」は 65 という値で、また「B」は 66 で表されます。Web ブラウザーの画面に表示される多くの文字は、実は、数値の羅列であり、その数値の羅列を、文字と認識して表示しているのです。

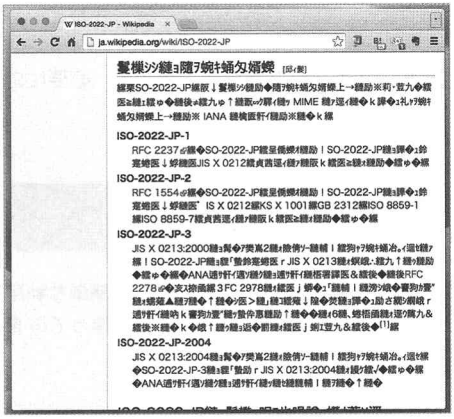
文字コードが難しい理由

アルファベットや数字だけを表すのであれば、1 バイトの範囲で事足ります。1 バイトは 8 ビットですから 0 から 255 まで 256 通りの文字が表現できることになり、26 文字のアルファベットに大文字・小文字さらに数字やコントロール用の記号などをふくめても十分な数があるからです。

しかし、日本語や中国語など漢字を使う言語圏では、文字の種類が多いため 2 バイト (0 から 65535) を利用しないと表すことができません。このため、どのようにして、文字をコンピューター上で扱うかについて、さまざまな方法が考案され利用されています。

コンピューターで扱えるデータが貧弱だった過去の経緯もあり、日本語だけを考えても、JIS コード、Shift_JIS コード、EUC コード、Unicode などなど、さまざまな文字コード体系が存在します。

そこで問題となるのが、文字コードの互換性です。文字コードの認識に失敗したり、文字コード同士の変換が間違っていたりすると、「文字化け」と呼ばれる現象を引き起こすことになります。



文字コード判定に失敗すると文字化けしてしまう

現在の主流 Unicode

現在では、世界中でコンピューターが使われています。そのため、さまざまな言語の文字を扱う必要があり、異なる言語を同時に表示したいという要望もあります。そこで、考えられたのが、世界中で使われるすべての文字を共通の文字集合で利用できるような文字コード体系（Unicode ＝ユニコード）です。各国で利用されている文字集合を持ち寄り、これを収録したものとなっています。

すでに、Unicode は、Windwos、Mac OS X、Unix 系 OS など多くの OS やプログラミング言語で採用されています。

文字集合と文字符号化スキーム

Unicode では、各文字に対するコード番号を定義した「文字集合」だけでなく、それをどのように表すかという「文字符号化スキーム (Character Encoding Scheme)」も定義されています。

そもそも、一口に「文字コード」と言っても、文字集合は同じでも符号化方式が異なる文字コードがあります。Unicode を例に取ると、Unicode を 2 バイト (16 ビット) で表したのが「UTF-16」で、1 バイト (8 ビット) で表したのが「UTF-8」です。そのため、どちらも Unicode の文字集合を利用している点では変わりはありませんが、異なる符号化方式で表されるため、テキストをバイナリレベルで見たときには、異なるものになります。

とは言え、同じ文字集合を利用しているのであれば相互の変換が容易です。

JavaScript での文字コード

JavaScript では UTF-16 を利用することが決まっています。HTML の文字コードが Shift_JIS や EUC であったとしても、変換されて内部では、UTF-16 が使われています。ですから、その点を意識したプログラムを書く必要があります。

ところで、日本語で配布されているデータの文字コードに関してですが、最近では、多くのサイトで、文字コード「UTF-8」で配布されることが多くなってきました。HTML5 でも標準の文字コードは UTF-8 になっており、それに伴ってデータも UTF-8 で配布されることが多くなってきています。しかしながら、ま

だまだ多くのサイトで配布されているデータは、Shift_JIS であったり、EUC であったりと統一されていません。

そこで、これらの混在するデータを JavaScript で扱うためには、文字コードを正しく判定し、必要に応じて変換する必要があります。

Node.js の場合

Node.js で、ファイルを書き出すときなど、標準で UTF-8 が採用されるようになっています。簡単な利用例として、fs モジュールを利用して、UTF-8 のテキストファイルを読み書きするプログラムを作ってみましょう。

● file: src/ch04/01-charset/fs-readwrite.js

```
// ファイルの読み書き for Node.js
var fs = require('fs');

// UTF-8 のファイルを読み込む
var txt = fs.readFileSync("sample-utf8.txt", "utf-8");
console.log(txt);

// ファイルを UTF-8 で書き込む
fs.writeFileSync("test.txt", txt);
```

プログラムを実行するには、以下のコマンドを実行します。

```
$ node fs-readwrite.js
```

UTF-8 のテキストを読み書きするプログラムの実行例

ファイルを同期的に読むには、fs.readFileSync() メソッドを、書き出すには、fs.writeFileSync() メソッドを利用します。

読み書き共に、オプションとして、文字エンコーディングが指定できるようになっています。とは言え、書き出しの時には、明示的に "utf-8" を指定しなくても、"utf-8" でデータが書き出されます。

ただし、以下のように読み出した時に、UTF-8 を指定しないと、正しく UTF-8 のテキストデータとしては読み出されません。この場合、読み込んだデータを、テキストデータとは認識しません。

```
// UTF-8のテキストを読むが文字コードを指定しない場合
var txt = fs.readFileSync("sample-utf8.txt");
console.log(txt);
```

console.log() の出力は以下のようになります。つまり、読み出したデータは、Buffer オブジェクトとなっているのです。

```
<Buffer e9 8a 80 e6 b2 b3 e9 89 84 e9 81 93 e3 81 ae e5 a4 9c 0d 0a e5 ae ae
e6 b2 a2 e8 b3 a2 e6 b2 bb 0d 0a 0d 0a e4 b8 80 e3 80 81 e5 8d 88 e5 90 8e
e3 81 94 ...>
```

残念ながら、この fs.readFileSync() メソッドでは、文字コードの Shift_JIS に対応していません。文字コードのオプションに、Shit_JIS を指定するとサポートしていない旨のメッセージが表示されてしまいます。

それでは、どのようにすれば、Node.js で Shift_JIS のテキストを扱えるのでしょうか。

Node.js で文字コード変換する場合

Node.js では、標準の状態では Shift_JIS などの文字コードを扱うことができないので、外部モジュールをインストールして利用します。

文字コードの変換には iconv というライブラリが広く使われていますが、Node.js 用にも iconv のモジュールが用意されています。また、文字コードの自動判定用のモジュール「jschardet」も用意されています。そのため、ここではこの2つのライブラリを利用します。

以下のようにして、npm を利用して、iconv と jschardet をインストールしましょう。

```
npm install iconv
npm install jschardet
```

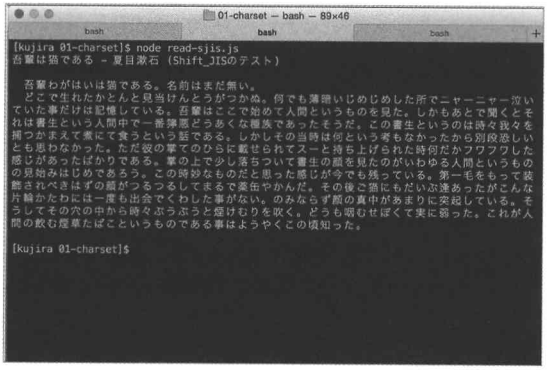
このモジュールを使って、先ほどの問いの答え、Shift_JIS で書かれたテキストファイルを読み込んで、UTF-8 で保存するプログラムを作ってみましょう。

● file: src/ch04/01-charset/read-sjis.js

```
// Shift_JISを読んでUTF-8で保存する for Node.js

var fs = require('fs');
var Iconv = require('iconv').Iconv;

// Shift_JISからUTF-8へ変換するオブジェクト
var sjis_utf8 = new Iconv('SHIFT_JIS', 'utf-8');

// Shift_JISのファイルを読み込む
var buf = fs.readFileSync('sample-sjis.txt');

var buf2 = sjis_utf8.convert(buf); // Shift_JISをUTF-8に変換
var txt = buf2.toString('utf-8');  // バッファーを文字列に変換
console.log(txt);

// UTF-8でファイルへ保存
fs.writeFileSync('test.txt', txt, 'utf-8');
```

プログラムを実行するには、上記の iconv モジュールをインストールした上で、以下のコマンドを実行します。

```
$ node read-sjis.js
```



Shift_JIS のテキストを読むプログラム

iconv モジュールは、多くの文字コード変換をサポートしています。日本語の文字コードだけでも次のものをサポートしています。

```
EUC-JP, SHIFT_JIS, CP932, ISO-2022-JP, ISO-2022-JP-2, ISO-2022-JP-1, EUC-
JISX0213, Shift_JISX0213, ISO-2022-JP-3
```

そのため、どのような変換を行うのか、Iconv オブジェクトを作成するときに指定するようになっています。例えば、Shift_JIS を UTF-8 に変換するのであれば、次のようにオブジェクトを作成します。

```
var sjis_utf8 = new Iconv('SHIFT_JIS', 'utf-8');
```

このオブジェクトの convert() メソッドを呼び出すことで文字コードを変換します。ただし、iconv モジュールでは、Buffer オブジェクトの中のデータを変換しているに過ぎません。これを JavaScript で文字列として使うには、toString() メソッドを呼び出して文字列に変換する必要があります。

```
// 文字コードの変換
var buf2 = sjis_utf8.convert(buf);
// バッファーを文字列に変換
var text = buf2.toString('utf-8');
```

文字コードのわからないテキストを読むとき

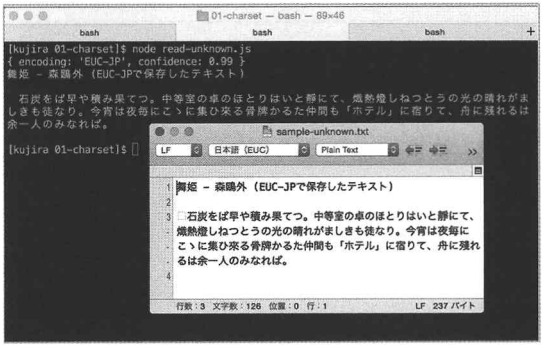
次に、テキストファイルが、Shift_JIS だとわからなかった場合を考慮してみましょう。この場合、まず、jschardet モジュールを使って、文字コードを特定します。

● file:src/ch04/01-charset/read-unknown.js

```
// 文字コードがわからない場合 for Node.js

var fs = require('fs');
var Iconv = require('iconv').Iconv;
var jschardet = require('jschardet');

// 文字コードがわからないファイルを読み込む
var buf = fs.readFileSync('sample-unknown.txt');

// 文字コード判定を行う
var det = jschardet.detect(buf);
console.log(det);
// Iconv で utf-8 に変換するオブジェクトを作る
var iconv = new Iconv(det.encoding, "utf-8");
var buf2 = iconv.convert(buf); // UTF-8 に変換
var txt = buf2.toString('utf-8');  // バッファーを文字列に変換
console.log(txt);
```

プログラムを実行するには、jschardet と iconv モジュールをインストールした上で、以下のコマンドを実行します。

```
$node read-unknown.js
```



文字コードを類推して読み込むプログラム

文字コードを特定するには、jschardet.detect() メソッドを使います。

```
var det = jschardet.detect(buf);
console.log(det);
```

すると、実行結果として、結果オブジェクトが返るのですが、これは、以下のような文字コード名とその信頼度が記されたオブジェクトになっています。

```
{
  encoding: 'EUC-JP',
  confidence: 0.99
}
```

そこで、この結果オブジェクトの encoding を、Iconv オブジェクトの作成時の引数に与えて、UTF-8 に変換することで、未知の文字コードのテキストを、文字列として扱うことができます。

```
// 文字コードを特定する
var det = jschardet.detect(buf);
// UTF-8に変換する
var iconv = new Iconv(det.encoding, "utf-8");
var buf2 = iconv.convert(buf);
// バッファーを文字列に変換
var txt = buf2.toString('utf-8');
```

iconv-liteを使って文字コードを変換する

Iconvモジュールは機能も安定しておりいろいろなプロジェクトで利用されているのですが、バージョンによってインストールがうまくいかないなど問題が起きることも報告されています。

そこで利用したいのが、純粋なJavaScriptで実装された文字コード変換モジュールのIconv-liteです。Iconvほど多くの文字コードに対応していませんが、CP932, CP936, CP949, CP950, GB2313, GBK, GB18030, Big5, Shift_JIS, EUC-JPが利用可能です。日本語・中国語・韓国語とアジア圏で必要な言語がサポートされています。JavaScriptだけで実装されているというと変換速度が心配なのですが、実際のところ、それほど遅くなく、Iconvよりも速いと言われています。

iconv-liteのインストールは以下のように行います。

```
$ npm install iconv-lite
```

使い方ですが、以下のように行います。Iconvと互換性はありません。encode()でJavaScriptの文字コードを他の文字コードに、decode()で他の文字コードをNode.jsの文字コードに変換します。

● file: src/ch04/01-charset/iconv-lite-test.js

```
// iconv-liteの利用例 for Node.js
var iconv = require('iconv-lite');
var fs = require('fs');

// テキストをShift-JISで書き込む
var str = "拙者は忍者だ、ニンニン!!";
var fname = "iconv-lite-test-sjis.txt";
// Shift_JISに変換
var buf = iconv.encode(str, "SHIFT_JIS");
// 保存
fs.writeFileSync(fname, buf, "binary");

// Shift_JISのテキストを読み出して表示
var bin = fs.readFileSync(fname, "binary");
// Shift_JISのテキストをUTF-8に変換
var txt = iconv.decode(bin, "SHIFT_JIS");
console.log(txt);
```

実行エンジンにRhinoを使う方法

ところで、JavaScriptの実行エンジンに、Javaランタイム上(JRE)で実行されるRhino/Nashornを使う場合はどうでしょうか。Rhino上でファイルの読み書きを行う場合には、JavaのAPIを利用します。JavaのAPIでは、各種文字コードをサポートしています。

Rhinoを利用して、本節で読み込んだテキストファイルを読み込んでみると以下のようになります。

● file: src/ch04/01-charset/rhino-read.js

```
// Rhinoでテキストファイルを読む for Rhino

// UTF-8
var txt;
txt = readFile("sample-utf8.txt", "utf-8");
print(txt);

// Shift_JIS
txt = readFile("sample-sjis.txt", "sjis");
print(txt);

// EUC
txt = readFile("sample-unknown.txt", "euc-jp");
print(txt);
```

Rhinoに用意されている、readFile()メソッドでは、引数に、ファイル名と文字コードを与えることで、正しくテキストファイルを読み込むことができます。

この節のまとめ

- ここでは、Web上にある文字コードについて、その取り扱い方を考えてみました
- さまざまな文字コードのテキストファイルを読み込むプログラム例を紹介しています。
- さらに、文字コードの変換方法についても解説しました。

02

データの整形と正規表現について

文字列の抽出や置換などの作業に欠かせないのが正規表現です。スクレイピングしたデータを使える形式に整形する際、正規表現を利用するとスマートに整えることができます。ここでは、正規表現の使い方を紹介します。

ここで学ぶポイント

- **正規表現について**

ツールやライブラリの一覧

- **Node.js**
- **ブラウザーの開発者ツールのコンソール**

正規表現とは

正規表現（英 :regular expression）とは、文字列のパターンを表現する表記方法のことです。文字列の検索や置換を行う際に活躍します。文字列の集合を「メタ文字」と呼ばれる特別な意味を持った記号を組み合わせて表現し、文字を検索したり、置換を行ったりします。

正規表現は、大抵のプログラミング言語に搭載されており、もちろん、JavaScript の標準機能として装備されています。ここでは、正規表現の基本的な記法と、JavaScript での使い方を紹介します。

正規表現は、使うほどに良さがわかります。噛めば噛むほど味が出るスルメイカのような存在です。文字列を扱うのであれば、ぜひマスターしておきたい技術の一つといえるでしょう。

JavaScript における正規表現の使い方

JavaScript では、ソースコード中へ「/ パターン /」のように書くことで、正規表現オブジェクトを生成できます。これを「正規表現リテラル」と言います。

```
var re = /value=\d+/;
```

また、RegExp オブジェクトを生成することで、正規表現の機能を使うこともできます。

```
var re = new RegExp("value=\d+");
```

正規表現リテラルを使うと、スクリプトが評価されるタイミングで正規表現がコンパイルされるので、パフォーマンスが良くなります。これに対して、「RegExp オブジェクト」を使う方法もあります。RegExp オブジェクトを使えば、文字列（String 型）の値で正規表現パターンを指定することができます。そのため、動的に正規表現文字列を生成することが可能です。

正規表現メソッド

JavaScript の正規表現では、次のメソッドが利用できます。

メソッド名	説明
exec	文字列中で一致するものを検索します。結果情報の配列を返します。検索に失敗した場合、null を返します。RegExp のメソッド
test	文字列中で一致するものがあるかどうかをテストします。true/false のいずれかを返します。RegExp のメソッド
match	文字列中で一致するものを検索します。結果情報の配列を返します。マッチしない場合、null を返します。String のメソッド
search	文字列中で一致するものがあるかどうかをテストします。マッチした場所のインデックスを返します。検索に失敗した場合 -1 を返します。String のメソッド
replace	文字列中で一致するものを検索し、別の文字列で置換します
split	文字列を正規表現で分割し、部分文字列の配列を返します

JavaScript の正規表現メソッド一覧

いくつか似たメソッドがありますが、文字列の中に正規表現パターンがあるかどうかを確認したい時には、test() または search() メソッドを使用します。より詳細な情報が知りたい時には、exec() あるいは match() メソッドを使用します。

具体的な例で確認してみましょう。RegExp.exec() メソッドでは、対象文字列を引数に取り、一致した部分文字列、そして、何文字目に一致したかのインデックス情報を配列で返します。

Node.js には、いわゆるコマンドラインから対話的に実行を行う「REPL 機能」というものが搭載されていて、プログラムのテストを簡単に実行することができます。REPL を起動するには、コマンドライン引数を与えずに、Node.js を実行します。

それでは早速コマンドラインから Node.js の REPL を実行し、簡単な計算を行ってみましょう。

```
$ node
> 3 + 5
8
```

簡単な足し算ですが、動作が確認できたでしょうか？

それでは、次に REPL で正規表現をテストしてみましょう。ちなみに、Node.js は、Google Chrome の V8 エンジンを利用して JavaScript を実行していますので、Chrome の開発者ツールにある、コンソール画面で実行するのとまったく同じ結果がでます。Node.js の REPL と Chrome のコンソール画面、どちらかお好きな方で試してみてください。

RegExp.exec() メソッド

exec() メソッドを使うと、正規表現マッチを実行します。結果の配列あるいは、null が返ります。マッチが成功した場合、配列を返し、正規表現オブジェクトのプロパティを更新します。

結果の配列には、マッチした部分、括弧で囲まれた部分文字列のマッチが格納されます。加えて、この配列は拡張プロパティを持っており、対象文字列がどこでマッチしたかのインデックス情報などが設定されます。

以下のプログラムを実行して結果を確認してみましょう。

```
// 正規表現で数値＋英小文字のパターン
var re = /([0-9]+)([a-z]+)/g;  // ---(※1)
// 対象文字列
var str = "111jpy,8usd,xxx";  // ---(※2)

// 1回目の実行  ---(※3)
console.log( re.exec(str) );
// 2回目の実行
console.log( re.exec(str) );
// 3回目の実行
console.log( re.exec(str) );
```

このプログラムの (※1) では、正規表現のパターンを指定して、正規表現オブジェクトを変数 re に代入しています。プログラムの (※2) では、正規表現で解析をしたい文字列を変数 str に代入しています。そして、プログラムの (※3) 以降の部分では、繰り返しパターンマッチを行います。

なお、(※1) で指定している正規表現のパターンは、数字＋小文字の組み合わせを取り出すというものです。

REPL で実行した結果を抜粋すると、以下のようになります。

```
...
> // 1回目の実行
> console.log( re.exec(str) );
[ '111jpy', '111', 'jpy', index: 0, input: '111jpy,8usd,xxx' ]

> // 2回目の実行
> console.log( re.exec(str) );
[ '8usd', '8', 'usd', index: 7, input: '111jpy,8usd,xxx' ]

> // 3回目の実行
> console.log( re.exec(str) );
null
```

正規表現の "g" フラグを指定すると、パターンに一致する全ての結果を取り出すという意味になりますが、この場合、exec() メソッドを複数回呼び出すことで、複数の結果を得ることができます。

ただし、test() メソッドや search() メソッドに比べると実行速度が遅くなりますので、単にマッチが成功するかだけ調べたい場合には、それらを使うことが推奨されています。

RegExp.test() メソッド

正規表現パターンと対象文字列がマッチするかどうかを調べます。結果は、true か false の真偽値を返します。

では、さっそく実際のプログラムで確認してみましょう。

```
// 郵便番号 nnn-nnnn を表すパターン
var re = /^\d{3}-\d{4}$/;

// 値にマッチするかどうか調べる
re.test("123-1234"); // -------- (※1)
re.test("12-1234");  // -------- (※2)
re.test("440-0011"); // -------- (※3)
re.test("aaa-bbbb"); // -------- (※4)
```

REPL の結果を抜粋すると以下のようになります。

```
> re.test("123-1234");
true
> re.test("12-1234");
false
> re.test("440-0011");
true
> re.test("aaa-bbbb");
false
```

(※1) と (※3) が郵便番号のパターンに一致しています。(※2) が false なのは数値の桁不足、(※4) が false なのは数値でなくアルファベットが指定されているからです。

String.match() メソッド

文字列オブジェクトのメソッドで正規表現マッチを行います。正規表現に "g" フラグを含んでいない場合、RegExp.exec() メソッドと同じ結果を返しますが、"g" フラグを含む場合には、マッチした部分を全て含む配列を返します。マッチしなければ、null を返します。

まずは、"g" フラグがあるかないかで結果の違いを確かめてみましょう。

```
// 対象文字列
var str = "v=20,n=40,c=30";
```

```
// 数値をマッチ
str.match(/[0-9]+/);
// 全ての数値をマッチ
str.match(/[0-9]+/g);
// 変数と値の組み合わせをマッチ
str.match(/\w+=\d+/g);
```

REPL で実行した結果を抜粋したものです。

```
> str.match(/[0-9]+/);
[ '20',
  index: 2,
  input: 'v=20,n=40,c=30' ]

> str.match(/[0-9]+/g);
[ '20', '40', '30' ]

> str.match(/\w+=\d+/g);
[ 'v=20', 'n=40', 'c=30' ]
```

String.search() メソッド

対象文字列と正規表現パターンがマッチするか調べます。正規表現が見つかったなら、見つかったインデックスを返し、見つからなければ、-1 を返します。

では、利用例を見てみましょう。

```
var str = "zip:999-9999, mail:a@example.com";

// 郵便番号を検索
str.search(/\d{3}-\d{4}/);
// E メールらしきものを検索
str.search(/\w+\@\w+\.\w+/);
// URL があるか検索
str.search(/https?:\/\//);
```

REPL の実行結果の抜粋です。

```
// 郵便番号を検索
> str.search(/\d{3}-\d{4}/);
4

// E メールらしきものを検索
> str.search(/\w+\@\w+\.\w+/);
19

// URL があるか検索
> str.search(/https?:\/\//);
-1
```

String.replace() メソッド

次に、正規表現による置換を行う、replace() メソッドについて見ていきましょう。異なる書式が指定できるので順に見ていきましょう。

```
[ 書式 1] 正規表現による置換
str.replace( 正規表現 , 置換文字列 )
```

実際のプログラムで雰囲気をつかみましょう。

```
// 対象文字列
var str = "Today 10per OFF";

// 数値を30に置換
str.replace(/\d+/, "30");
// 数値に続く英小文字を置換
str.replace(/\d+[a-z]+/, "500yen");
// アルファベットを消して数字だけ残す
str.replace(/[a-zA-Z]+/g, "");
```

REPLでの実行結果の抜粋です。

```
> // 数値を30に置換
> str.replace(/\d+/, "30");
'Today 30per OFF'

> // 数値に続く英小文字を置換
> str.replace(/\d+[a-z]+/, "500yen");
'Today 500yen OFF'

> // アルファベットを消して数字だけ残す
> str.replace(/[a-zA-Z]+/g, "");
' 10 '
```

また、置換文字列の中には、特殊な置換パターンを含めることができます。

パタ-	説明
$$	「$」を挿入します
$&	マッチした部分文字列を挿入します
$`	マッチした部分文字列の直前の文字列を挿入します
$'	マッチした部分文字列の直後の文字列を挿入します
$1 $2 $3 ...	括弧でキャプチャされた部分文字列を挿入します

置換文字列に使える特殊記号

実際のプログラムで動作を確認してみます。

```
// 対象文字列
var str = "tel:045-111-222";
// 先頭を括弧でくくる
str.replace(/(\d+)-(\d+)-(\d)/, "($1)-$2-$3");
```

REPLでの実行結果の抜粋です。

```
> str.replace(/(\d+)-(\d+)-(\d)/, "($1)-$2-$3");
'tel:(045)-111-222'
```

次に、コールバック関数を利用したreplace()の利用例を見ていきましょう。

```
[書式2] コールバック関数を利用した置換
str.replace( 正規表現 , 置換関数 )
```

文字列 str の中に、正規表現が見つかると、置換関数を実行します。そして、置換関数の戻り値を利用して文字列を置換します。では、プログラムで確認してみましょう。以下は、英単語の小文字を大文字に置換するというものです。

```
// 対象文字列
var str = "piano GUITAR";

// 大文字を小文字に変換
str.replace(/[a-z]+/g, function(v) {
  return v.toUpperCase();
});
```

上記のプログラムを実行すると、'PIANO GUITAR' と表示されます。
次にテキスト中の数値を探して、税金 8% を加算するというプログラムを書いてみましょう。

```
// 対象文字列
var str = "price: 100yen";

// 値段をマッチさせて、税金 8% を加算して置換
str.replace(/\d+/, function(v) {
  v = parseInt(v);
  return Math.ceil(1.08 v);
});
```

プログラムを実行すると、'price: 108yen' と表示されます。
このように、コールバック関数を利用することで、単なる置換では処理できないようなことが可能になり、プログラミングの自由度があがります。

この節のまとめ

- この節ではデータの整形に役立つ正規表現について説明しました。
- 正規表現に初めて触れる人ならば、REPL へいろいろな正規表現を打ち込んでその動作をマスターしましょう。

03 データ形式の基礎知識

Web上にはさまざまなデータが公開されていますが、公開されているデータ形式もさまざまです。ここでは、どんなデータフォーマットがあるのか、どのようにしてJavaScriptで扱うのかを紹介します。

ここで学ぶポイント

- データフォーマットについて
- JSON/JSON5/CSON/XML/RSS/YAML/INI/CTV/TSV

ツールやライブラリの一覧

- Node.jsなど

Webにあるデータ形式

「Web上で公開されているデータがすべてJSON形式になったら良いのに！」

多くのJavaScriptプログラマーはこのように思うことでしょう。JSON形式とは、JavaScriptにおけるオブジェクトリテラルの記述方式で、非常に可読性が良い上に、JavaScriptから扱いやすいという特徴があります。

しかしながら、Webにデータを公開している人が、すべてJavaScriptプログラマーではありません。

データ形式にも、さまざま流行廃りがあるものです。ある一時期、XMLが大流行した頃は、みんながこぞってXMLでデータを公開しましたし、JavaScriptが流行し、JSON形式が盛り上がると、多くの人がJSONデータを公開しました。そして、Ruby on Railsが流行すると、その設定データとして採用されたYAMLが流行し、多くのデータがYAMLでも書かれるようになりました。今後も、別の新しいデータ形式が登場してくることでしょう。

データの作成者は、どんなデータを、どんな目的で公開するのかによって、フォーマットを選んでいると思いますが、中には、流行っているからとか、データ作成に利用したツールが対応しているからという理由で選ばれたものもあるかもしれません。

幸いなことに、ある程度著名なデータ形式であれば、そのデータ形式を扱うためのライブラリも存在するので、新しいデータ形式を怖がる必要はありません。

ここでは、さまざまなデータ形式を紹介し、JavaScriptでどのように処理したら良いのかを紹介していきます。

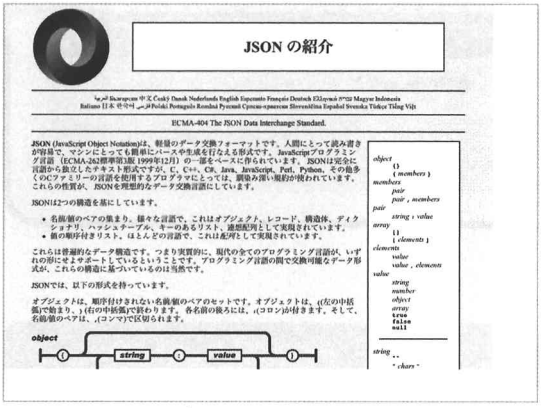
JSON 形式とはなにか

はじめに、JavaScript から最も扱いやすいデータ形式の JSON 形式から見ていきましょう。

JSON 形式はテキストで作られたデータです。そのため、テキストエディターで編集することができます。そして、JavaScript のオブジェクトを記述する際に使用するオブジェクトリテラルの形式で記述できます。

詳しい仕様が以下で公開されています。

```
json.org > JSON の紹介
http://json.org/json-ja.html
```



JSON の解説ページ

JSON は、いろいろな Web API の出力形式としても使われています。また、プログラムなどの設定を記述した、いわゆる設定ファイルにも使われています。

さて、JavaScript の得意な皆さんは、既にご存じのことと思いますが、ここでは、簡単に JSON の記述方法を紹介します。

JSON 形式で、配列は [n1,n2,n3,n4,...] のように表します。オブジェクトであれば、{key1:value1, key2:value2, key3:value3...} のように表します。これを組み合わせることで、複雑なデータを表現することができます。

まずは、配列データの例です。数値データであれば、数値をそのままカンマで区切って記述できます。

```
[31,20,55,90,34]
```

文字列データであれば、ダブルコーテーション ("...") で囲って記述します。

```
["apple", "banana", "tomato"]
```

オブジェクトデータの記述は以下のように記述します。JavaScript のリテラル式と違うのは、必ずキー名もダブルコーテーションで囲う必要があるという点です。

```
{
  "id"    : 1004,
  "name"  : "Tomato",
  "price" : 3400,
  "memo"  : "Fresh & sweet"
}
```

そして、オブジェクトの中に、配列を、配列の中にオブジェクトを入れ子状に書くことができます。

```
{
  "group" : "vegetable",
  "items" : [
    {"name":"Tomato", "price":300},
    {"name":"Banana", "price":170},
    {"name":"Apple", "price":210},
  ]
}
```

JavaScript で JSON を扱うのは簡単です。基本的に、JSON 形式のデータを eval() メソッドで評価すれば、JavaScript のオブジェクトとして扱うことができるからです。しかし、セキュリティの観点から、現在 JSON データを eval() で評価することは推奨されていません。JSON.parse() という専用メソッドが用意されています。

ここでは、以下のような果物の名前と価格を記した JSON データを用意しました。

● file: src/ch04/03-data-format/test.json

```
{
  "title"   : "Fruits Database",
  "version" : 2.13,
  "items" : [
    {"name":"Tomato", "price":300},
    {"name":"Banana", "price":170},
    {"name":"Apple", "price":210},
    {"name":"strawberry", "price":520},
    {"name":"persimmon", "price":490},
    {"name":"kiwi", "price":320}
  ]
}
```

これを、Node.js で読み込んで、果物名と価格を表示させてみます。

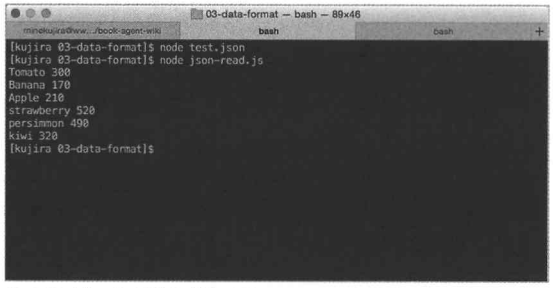
● file: src/ch04/03-data-format/json-read.js

```
// JSON データを読む for Node.js

var fs = require('fs');

// JSON ファイルを読む
var json = fs.readFileSync("test.json", "utf-8");

// JS のオブジェクトに変換
var obj = JSON.parse(json);

// アイテム一覧を表示
var items = obj.items;
for (var i in items) {
  var item = items[i];
  var name = item.name;
```

```
  var price = item.price;
  console.log(name, price);
}
```

プログラムを実行するには、以下のコマンドを実行します。

```
$ node json-read.js
```



JSON データを読んだところ

最後に、JavaScript の JSON オブジェクトの書式を紹介します。

```
[書式] JSON を扱う
// JSON データを JS オブジェクトに変換
var obj = JSON.parse(text);

// JS オブジェクトを JSON データに変換
var text = JSON.stringify(obj);
```

このように、JavaScript で JSON データを扱うのは非常に簡単です。なんと言っても、JavaScript のオブジェクトとデータが、1:1 になっているので、余計なことを考えなくて済みます。

JSON の改良版 JSON5 形式

前項で紹介した通り、「JavaScript 使い」にとって、JSON はとても便利なデータ形式です。とは言え、欠点もあります。よく聞かれる欠点は以下の通りです。

- **データ中にコメントが入力できない**
- **オブジェクトのキーもダブルコーテーションで囲う必要がある**
- **データの末尾にカンマを書くとパースエラーになる**

これらの欠点を改善したのが、JSON5 形式です。基本的には、JSON と同じなのですが、以下のように記述することが可能になります。

```
// 「JSON5 のデータ記述例です」
// と、このようにコメントが記入できます。
{
  // キー名をダブルコーテーションで囲う必要がない
  key: "value",

  // 複数行の文字列も「\」で区切って記述可能
  multi_line: "This is a pen.\
This is a eraser.\
This is a bookmark.",

  /*
   * 範囲コメントも記述可能
   */

  // 16 進数も記述可能
  hex_data_sample: 0xC0FFEE,

  // 小数点の 0 を省略可能
  half_sample: .5,

  // 明示的に +/- を記述可能
  delta: +94,

  // Inifinity や NaN も記述可能
  value_a: Infinity,
  value_b: NaN,

  // 配列末尾にカンマがあっても大丈夫
  items: [
    "hoge",
    "bar",
    "foo", // <--- 末尾データの最後のカンマ！
  ],       // <---

}
```

さて、このように記述した JSON5 のデータですが、ほぼ JSON と同じように、JavaScript のオブジェクトに変換することができます。

JSON5 のライブラリが npm でインストールできます。

```
$ npm install json5
```

ライブラリをインストールしたら、次のように JSON5 データを扱うことができます。JSON とほぼ同じですね。

```
// モジュールの読み込み
var JSON5 = require('json5');

// JSON5 の利用
var obj = JSON5.parse('[1,2,3,]');
var str = JSON5.stringify(obj);
```

CSON 形式

JSON 形式が JavaScript のオブジェクト記述式を基にしているのに対し、CSON 形式は CoffeeScript のオブジェクト記述式を基にしたデータ形式です。CoffeeScript のプログラムは JavaScript に変換されて実行されますし、CoffeeScript は JavaScript の子供のようなものです。そのため、ざっくり言うと、CSON は、JSON の記述法をより簡潔にしたものと言えます。

具体的なデータを見てみましょう。先ほど紹介した JSON5 をより簡潔に記述できるようにしたものであることがわかるでしょう。

```
{
  # 配列の記述
  items: [
    'banana'
    'tomato'
    'kiwi'
  ]

  # オブジェクトの記述
  dog:
    name : 'taro'
    age  : 30

  # 複数行のテキストデータも記述可能
  memo: '''
    His name is Taro.
    Taro is 4 years old.
    He is my friend.
    '''
}
```

CSON を操作するためにも、ライブラリが公開されています。npm を利用してインストールできます。

```
$ npm install cson
```

CSON の場合も、JSON オブジェクトと同じメソッドが定義されているので便利ですね。

```
// モジュールの読み込み
var CSON = require('cson');

// CSON の利用
var obj = CSON.parse( cson_str );
var str = CSON.stringify(obj);
```

では、CSON データを読む実際のプログラムを確認してみましょう。ここでは、先ほど利用した JSON データを、CSON にそのまま書き写してみました。

● file: src/ch04/03-data-format/test.cson

```
title: "Fruits Database"
version: 2.13
items: [
  {
    name: "Tomato"
    price: 300
  }
```

```
  {
    name: "Banana"
    price: 170
  }
  {
    name: "Apple"
    price: 210
  }
  {
    name: "strawberry"
    price: 520
  }
  {
    name: "persimmon"
    price: 490
  }
  {
    name: "kiwi"
    price: 320
  }
]
```

以下がCSONを読み込んで内容を表示するプログラムです。

● file: src/ch04/03-data-format/cson-read.js

```
// CSONデータを読む for Node.js

var CSON = require('cson');
var fs = require('fs');

// CSONファイルを読み出す
var cson = fs.readFileSync('test.cson', 'utf-8');

// CSONをJSのオブジェクトにパース
var obj = CSON.parse(cson);

// 内容を出力
for (var i in obj.items) {
  var it = obj.items[i];
  console.log(it.name, it.price);
}

// JSオブジェクトをCSONに変換
var cson_out = CSON.stringify(obj);
console.log(cson_out);
```

プログラムを実行するには、以下のコマンドを入力します。

```
$ node read-cson.js
```

加えて、CSONをインストールすると、JSONからCSONへの変換、CSONからJSONへの変換を行うコマンドラインツールも利用できるようになります。

例えば、今回利用したCSONのデータファイル「test.cson」をJSONに変換してみます。以下のコマンドを実行すると、outout.jsonファイルが出力されます。

```
$ cson2json test.cson > output.json
```

XML/RSS 形式

XML や RSS に関しては、本書の 2 章「XML/RSS の解析」でも紹介しましたが、ここでは異なる角度から改めて紹介します。2 章で取り上げたように、XML 形式は、バイナリデータではなく、テキストデータです。テキストファイルを利用して編集することができます。タグによって、データをマークアップすることができます。タグを、入れ子状の階層構造にすることもできます。

以下は、XML の基本構造を表すものです。

```
< 要素名 属性 =" 属性値 "> 内容 </ 要素名 >
```

データの内容を、任意の < 要素 > タグでマークアップします。好きな要素名を指定してマークアップできます。また、一つの要素にも、属性を使うことで、複数の値を指定することができます。

```
< 商品 id="S001" 値段 ="4500">SD カード </ 商品 >
```

そして、この要素を異なる要素でグループ化することができます。これにより、要素を階層化することができます。

```
< 商品グループ type=" 電子機器 ">
  < 商品 id="S001" 値段 ="4500">SD カード </ 商品 >
  < 商品 id="S002" 値段 ="3200"> マウス </ 商品 >
</ 商品グループ >
```

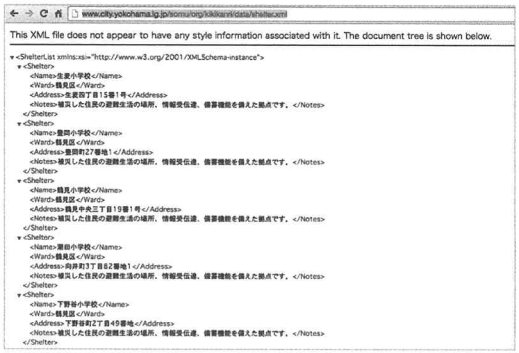
このような、XML 形式のデータを扱うために、2 章では、XML を JavaScript のオブジェクトに変換する「xml2js」というモジュールを使ってみました。今回は、JavaScript のライブラリとして有名な「jQuery」のように、DOM へアクセスできる「cheerio」モジュールを利用して解析してみます。

地域防災拠点 XML を解析しよう

ここでは、横浜市が公開している地域防災拠点データが XML で公開されていますので、これを利用するプログラムを作ってみます。

```
横浜市 > 防災関連データ > 地域防災拠点
http://www.city.yokohama.lg.jp/somu/org/kikikanri/data/
```

上記の URL より XML ファイルを取得して「shelter.xml」という名前で保存しておきます。



XML データをブラウザで見たところ

そして、今回のプログラムで利用する「cheerio」モジュールを先にインストールしておきましょう。コマンドラインから以下のコマンドを実行します。

```
$ npm install cheerio
```

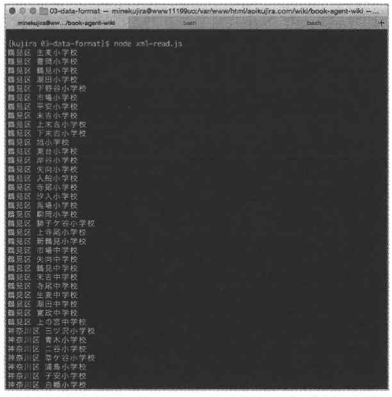
今回のプログラムでは、地域防災拠点のデータを読み込んで、各拠点について、地区と名前を一覧出力するというものを作ってみます。
以下がプログラムです。

● file: src/ch04/03-data-format/xml-read.js

```
// 地域防災拠点を読む for Node.js

// モジュールの取り込み
var fs = require('fs');
var cheerio = require('cheerio');

// XML ファイルを読む --- (※1)
var xml = fs.readFileSync("shelter.xml", "utf-8");

// XML ファイルをパースする --- (※2)
$ = cheerio.load(xml);

// 各防災拠点を順にチェック --- (※3)
$("Shelter").each(function(i, el){
  // 名前と地区を画面に表示 --- (※4)
  var name = $(this).children("Name").text();
  var ward = $(this).children("Ward").text();
  console.log(ward, name);
});
```

プログラムを実行するには、次のコマンドを実行します。

```
$ node xml-read.js
```



地域防災拠点の XML から地域と施設名称を表示したもの

「cheerio」モジュールを利用すると、特定の CSS セレクタに合致する要素を手軽に取り出すことができます。もともと、地域防災拠点の XML は複雑なものではありませんが、各地点が <Shelter> タグでマークアップされている点に注目して処理を行っています。

まず、ポイントとなるのは、「cheerio」モジュールでXMLをパースする部分です。今回は、Webサイトではなく、XMLファイルからの読み込みなので、プログラムの(※1)の部分で、fs.readFileSync()メソッドでテキストデータを読み出して、(※2)のcheerio.load()メソッドでXMLのパースを行います。

そして、パースが完了すれば(※3)の部分のように、「$(" セレクタ ")」の書式で、任意の要素にアクセスすることができます。ここでは、<Shelter>タグの一覧を取得しているので、each()メソッドを利用して、各<Shelter>タグを処理していきます。

プログラムの(※4)の部分にあるように、each()メソッドの中では、各要素がthisあるいはコールバック引数のelに入っているので、$(this).children(" セレクタ ")のようにして、さらに子要素を取り出すことができます。

YAML形式

YAML（ヤムル）とは、インデントを利用して階層構造を表現するのが特徴のデータ形式です。テキストデータなので、テキストエディターを用いて編集することができます。XMLよりもシンプルで、上述のJSONと似ています。

インデントを用いて階層構造を記述しますが、インデントにはタブが利用できず、スペースのみが利用できます。

YAMLの基本は、配列、ハッシュ、スカラー(文字列・数値・真偽値など)です。

配列を表すには、行頭にハイフン「-」を付けます。ハイフンの後には半角スペースが必要です。

```
- banana
- kiwi
- mango
```

このとき、半角スペースでインデントすると、配列の中に配列を表現することができます。ただし、インデントする直前には、以下のように空の要素を入れてください。

```
- Yellow
-
  - Banana
  - Orange
- Red
-
  - Apple
  - Strawberry
```

次にハッシュの書き方を紹介します。ハッシュは、JavaScriptのオブジェクトに相当するものです。「キー:値」の形式で記述します。

```
name: Taro
age:  4
color: brown
```

こちらもインデントすることで、階層構造を表現できます。

```
name: Taro
property:
  age: 4
  color: brown
```

配列とハッシュを組み合わせて、複雑なデータを表現することができます。

```
- name: Taro
  color: brown
  age: 4
  favorites:
    - Banana
    - Miso soup
- name: Mike
  color: white
  age: 8
  favorites:
    - Orange
    - Candy
- name: Kuro
  color: black
  age: 3
  favorites:
    - Banana
    - Mango
```

加えて、YAMLにはフロースタイルが用意されており、これを利用すると、JSONと同じように、配列を[n1, n2, n3...]で表現し、ハッシュを{ key1:value1, key2:value2 ...}のように表現できます。ただし、カンマ「,」やコロン「:」の後には半角空白を入れる必要があります。

```
- name: Taro
  favorites: ["Banana", "Miso soup"]
- name: Mike
  favorites: ["Orange", "Candy"]
- name: Kuro
  favorites: ["Banana", "Mango"]
```

YAMLではコメントを記述することができます。コメントは「#」から始めます。

```
# YAMLにはコメントを残すことが可能
# 南国のフルーツが好き
- Banana
- Mango
```

また、複数行の文字列を指定することもできます。

```
multi-line: |
  I like Banana.
  I like Mango.
  I like Orange.
```

さらに、YAML ではアンカーとエイリアスが利用できます。「&name」で印をつけておいて、「*name」で参照することができます。「&name」がアンカーで、「*name」がエイリアスです。

例で見てみましょう。以下の YAML データでは、冒頭で色名を定義し、それ以降の部分で、実際に色データを指定しています。

```
# 色を定義する
color_define:
  - &color1 #FF0000
  - &color2 #00FF00
  - &color3 #00FFFF

# 色設定を記述する
frame_color:
  title: *color1
  logo: *color2

article_color:
  title: *color2
  back: *color3
```

例えば、上記の「frame_color.title」は、アンカー「color1」を参照してるので、実際には「#FF0000」を指定してるのと同じ意味になります。この仕組みはとても便利ですね。

Node.js で YAML を利用する場合

それでは、YAML データを JavaScript のプログラムで扱ってみましょう。こちらも、npm で YAML を利用するモジュールをインストールすることができます。

```
$ npm install js-yaml
```

サンプル YAML データは以下の通りです。アンカーとエイリアスの機能を利用して、果物の値段をコイン枚数で指定するようにしてみました。

● file: src/ch04/03-data-format/test.yml

```
title: Fruits Database
version: 3.2
price-define:
  - &one-coin 100
  - &two-coin 200
  - &three-coin 300
items:
  - name: Tomato
    price: *three-coin
  - name: Banana
    price: *two-coin
  - name: Apple
    price: *two-coin
  - name: Kiwi
    price: *three-coin
```

サンプルデータを読み込んで果物と値段一覧を表示するプログラムは、以下の通りです。

● file: src/ch04/03-data-format/yaml-read.js

```
// YAMLを読む for Node.js
var yaml = require('js-yaml');
var fs = require('fs');

// YAMLデータを読み込む
var txt = fs.readFileSync('test.yml', 'utf-8');

// JavaScriptのオブジェクトに変換
var obj = yaml.safeLoad(txt);

// 果物の内容を表示
for (var i in obj.items) {
  var it = obj.items[i];
  console.log(it.name, it.price);
}
```

プログラムを実行するには、以下のコマンドを入力します。

```
$ node yaml-read.js
```

YAMLデータを読み込むプログラム

プログラム自体は、それほど難しい点はないでしょう。YAMLデータをパースするには、js-yamlモジュールのsafeLoad()メソッドを利用します。js-yamlモジュールには、load()メソッドも用意されていますが、安全にデータを読み込むために、safeLoad()メソッドを利用することが推奨されています。

INIファイル形式

INIファイル形式のデータがWebで配布されることは少ないかもしれませんが、今でも、設定ファイルを記述するのに利用されているのがINIファイル形式です。この形式は、単純なテキストファイルであり、一時は、設定ファイルのデファクトスタンダードであるとも言われていました。主に、Windowsで利用されていますが、PHPの設定ファイルなども、INIファイル形式で記述されています。

基本的な構造は、以下のようにシンプルなものです。設定名と実際の値が「name=value」の形式で並んでいるだけです。

```
name1=value1
name2=value2
name3=value3
```

これに加えて、各設定値をグルーピングするためのセクションがあります。セクションは「[セクション名]」のように記述します。

```
[section1]
name1=value1
name2=value2

[section2]
name3=value3
name4=value4
```

他には、セミコロン「;」から始まるコメントを記述することもできます。
このようにとてもシンプルな形式のため広く利用されています。

Node.js で INI ファイルを扱う方法

Node.js で INI ファイルを扱うには、npm で「ini」モジュールをインストールします。

```
$ npm install ini
```

使い方は、JSON と同じで、INI データを JavaScript のオブジェクトに変換する ini.parse() メソッドと、オブジェクトを ini データに変換する ini.stringify() メソッドが用意されています。

注意しないといけない点として、多くの INI ファイルが日本語 Windows の標準文字コードである Shift_JIS で書かれているという点です。場合によっては、Shift_JIS を UTF-8 に変換してから読み込む必要があるでしょう。詳しくは、本章の文字コードの節、あるいは、後述の CSV ファイル形式を参考にしてください。

実際のデータとプログラム例で見ていきましょう。まずは、サンプルデータです。

● file: src/ch04/03-data-format/test.ini

```
; Fruits Database
; version 1.2

[Banana]
price=300
color=yellow

[Apple]
price=300
color=red

[Mango]
price=320
color=yellow

[Strawberry]
price=430
color=red
```

次に上記の INI ファイルを読み込むプログラムです。

● file: src/ch04/03-data-format/ini-read.js

```
// INI ファイルを読み込む for Node.js

var fs  = require('fs'),
    ini = require('ini');

// ここでは、INI ファイルが UTF-8 であると仮定して読む
var txt = fs.readFileSync('test.ini', 'utf-8');

// JS のオブジェクトに変換
var obj = ini.parse(txt);

// 内容を表示
for (var name in obj) {
  var it = obj[name];
  console.log(name, it.price, it.color);
}
```

プログラムを実行するには、以下のコマンドを入力します。

```
$ node ini-read.js
```

INI ファイルを読み込むプログラム

上記のプログラムを見るとわかるように、INI ファイルのセクションが、JavaScript オブジェクトのキーにマッピングされるようになっています。

例えば、以下のような INI ファイルを読み込んだとします。

```
[section1]
name1=value1
name2=value2
[section2]
name3=value3
```

すると、次のような JS のオブジェクトにマッピングされます。

```
{
  "section1": {
    "name1": "value1"
    "name2": "value2"
  },
  "section2": {
    "name3": "value3"
  }
}
```

CSV ファイル /TSV ファイル形式

Web で公開されているデータで、実は XML 以上に流通してるのではないかと思われるのが CSV ファイル、TSV ファイル形式です。その理由は、フォーマットが非常にシンプルであることに加えて、各種のデータが Excel で作成されていることや、多くのデータベースが CSV 形式のエクスポートをサポートしているためだと思います。

CSV(Comma-Separated Values) ファイルは、カンマ区切りデータとも呼ばれています。それは、各フィールドがカンマで区切られているデータであるからです。CSV ファイルはテキストファイルであるので、テキストエディターで容易に編集することができます。

さまざまな表計算ソフトやアドレス帳、データベースが、データ交換用のフォーマットとして CSV ファイルを採用しています。とは言え、CSV ファイルの実装は、アプリケーションによって若干の差異があり、それが原因でデータが破壊されるという事態も発生します。CSV のデータ仕様を定めた、RFC 4180 もあります。

類似フォーマットとしては、カンマでなくタブでフィールドを区切った TSV(Tab-Separated Values) や、半角スペースでフィールドを区切った、SSV(Space-Separated Values) もよく使われています。

では、CSV ファイルの構造を紹介します。CSV ファイルは、一つ以上のレコードから成り立っています。レコードは、改行 (CRLF、つまり、U+000D U+000A) で区切られます。

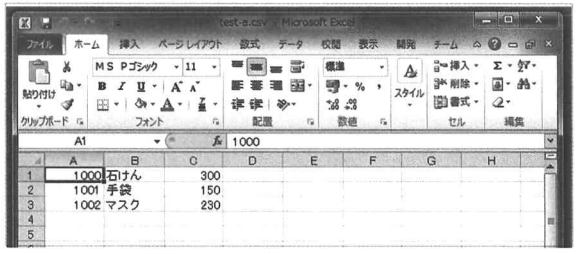
```
レコード 1
レコード 2
レコード 3
...
```

レコードは一つ以上の同じ個数のフィールドで成り立っています。フィールドはカンマ「,」で区切られます。ちなみに、ファイルの先頭のレコードには、ヘッダ行があっても良いことになっています。

```
ID, 商品名 , 値段
1000, 石けん , 300
1001, 手袋 , 150
1002, マスク , 230
```

各フィールドは、以下のように、ダブルコーテーション「"..."」で囲んでも良いことになっています。(囲んでも良いということで、囲まなくても良いのです。)

```
"1000"," 石けん ","300"
"1001"," 手袋 ","150"
"1002"," マスク ","230"
```

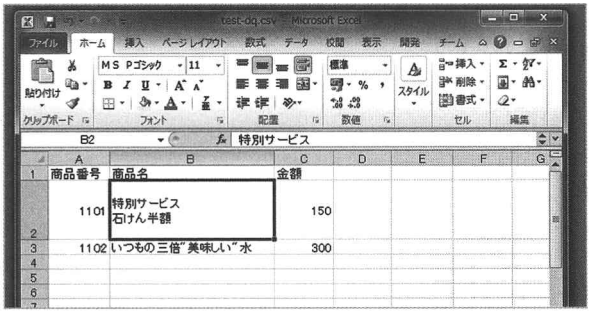


Excel で CSV を開いたところ

ただし、フィールドの中に、ダブルコーテーション「"」やカンマ、改行を含む場合には、必ず、ダブルコーテーションで囲むことになっています。

このとき、ダブルコーテーションをエスケープするには、二重にダブルコーテーションを記述し「"aa""bb"」のように書きます。

```
" 商品番号 "," 商品名 "," 金額 "
"1101"," 特別サービス
石けん半額 ","150"
"1102"," いつもの三倍 "" 美味しい "" 水 ","300"
```



Excel で表示したところ

Node.js で CSV ファイルを扱う方法

では、CSV を扱うプログラムを見ていきましょう。

日本語データを含む大多数の CSV ファイルは、文字コードが Shift_JIS となっているので、JavaScript で CSV ファイルを読み込む際には、文字コードを UTF-8 に変換する必要があります。

そのため、CSV ファイルを扱う「csv」モジュールに加えて、文字コードを変換する「iconv」モジュールも必要となります。

```
$ npm install comma-separated-values
$ npm install iconv
```

ここでは、以下のような CSV ファイルを「test.csv」という名前で、Shift_JIS で保存します。そしてプログラムから読み込んでみます。

```
商品名 , 値段 , 説明
バナナ , 180, フィリピン産の上等バナナ
イチゴ , 450, 愛知県産のブランドイチゴ
パイナップル , 390, 沖縄産で甘みが強い
ミカン ( 一箱 ), 1200, 静岡県産糖度の高いミカン
```

以下がプログラムです。

● file: src/ch04/03-data-format/csv-read.js

```
// CSV ファイルを読む for Node.js

var fs    = require('fs');
var CSV   = require('comma-separated-values');
var Iconv = require('iconv').Iconv;

// Shift_JIS を UTF-8 に変換するオブジェクトを生成 --- (※1)
var iconv = new Iconv('SHIFT_JIS', 'UTF-8');
// Shift_JIS に変換
var buf = fs.readFileSync("test.csv");
var txt = iconv.convert(buf).toString("utf-8");

// CSV ファイルをパースする ---- (※2)
var csv = new CSV(txt, {header:false});
var records = csv.parse();

// 一行目はヘッダなので捨てる --- (※3)
records.shift();

// 結果を出力する --- (※4)
for (var i = 0; i < records.length; i++) {
  var fs = records[i];
  var name = fs[0];
  var price = fs[1];
  var memo = fs[2];
  console.log(name, price, memo);
}
```

プログラムを実行するには、以下のコマンドを入力します。

```
$ node csv-read.js
```

CSV ファイルを読むプログラム

CSV を読み込むプログラムの流れを確認してみましょう。まず、(※1) の部分では、CSV ファイルの内容をメモリに読み込みます。CSV ファイルの文字コードは大抵 Shift_JIS なので、この時、文字コードの変換も行っておきます。続いて、(※2) で CSV ファイルのパース処理（つまり、解析）を行います。結果は、二次

元配列となります。(※3) では、二次元配列の先頭の一行（ヘッダ行）を削ります。そして、(※4) の部分で、結果をコンソールに出力します。

CSV の読み込みに関しては、少し説明が必要です。Shift_JIS で記述されたファイルを読み込んだために、プログラムの (※1) では、iconv モジュールを利用して、Shift_JIS を UTF-8 に変換しています。そして、comma-separated-values モジュールのオブジェクトを作成します。このとき、オプションを指定できるようになっています。ここでは {header:false} というオプションを指定しています。これを指定した時には、二次元配列変数でデータを戻してくれます。もし、{header:true} のオプションを指定すると、戻り値は、オブジェクトの配列となります。

```
var CSV    = require('comma-separated-values');
//
var txt = "..."; // CSV のテキストデータ
//
var csv = new CSV(txt, {header:false});
var records = csv.parse();
```

プログラムを少し改造して、この header オプションの動きを確認してみましょう。

● file: src/ch04/03-data-format/csv-read2.js

```
// CSV ファイルを読む for Node.js

var fs     = require('fs');
var CSV    = require('comma-separated-values');

// CSV データ
var txt = "id,name,price\r\n" +
  "1001,Banana,300\r\n" +
  "1002,Apple,230\r\n";

// CSV ファイルをパースする
var csv = new CSV(txt, {header:true});
var records = csv.parse();

// コンソールに出力
console.log(records);
```

プログラムを実行すると以下のようなデータが出力されます。二次元配列データではなく、オブジェクトの配列となっていることがわかりますね。

```
[
  { id: 1001, 'name': 'Banana', 'price': 300 },
  { id: 1002, 'name': 'Apple', 'price': 230 }
]
```

その他の形式

Web で公開されているその他の形式としては、Microsoft の Word/Excel 形式や、PDF 形式が多く見られます。それぞれ、専用のアプリを利用して確認することが多いので、ここでは扱いません。しかし、Node.js には、Excel の読み書きを行う「officegen」モジュールや、PDF の書き出しが可能な「PDFKit」が用意されていますので、これらのモジュールを利用することもできるでしょう。

この節のまとめ

- ここでは、Web で公開されているデータ形式について考察しました。
- それぞれ、Node.js でどのように扱えば良いか紹介しています。
- Excel や CSV、PDF のデータは Web 上にたくさん存在するので、実際にデータ解析する際には参考となるでしょう。

04

CoffeeScriptは必修項目

最近の JavaScript を語るときに、避けて通れない話題が「CoffeeScript」です。他人と違う、少し尖ったことを JavaScript で実現しようと思ったら、もはや CoffeeScript は必須となっています。ここでは、基礎的な部分に絞って紹介しましょう。

ここで学ぶポイント	ツールやライブラリの一覧
●CoffeeScriptとは? ●とりあえず読めるようになろう	●CoffeeScript

なぜ CoffeeScript ?

プログラミングが上達する早道は、自分なりにプログラムを改造してみることです。いまどきのプログラマーでしたら、改造のためには多少なりとも Web を調べるかと思います。ここまでのプログラムをいろいろ改造してみた方ならわかると思いますが、Node.js や PhantomJS/CasperJS、Electron（旧 atom-shell）などのキーワードを Web で検索してみると、JavaScript に似て非なるプログラミング言語を目にしたのではないでしょうか。それが、CoffeeScript です。

いまでは、多くの開発者が CoffeeScript を利用して JavaScript を書いています。CoffeeScript は実行されるときに JavaScript に変換されて実行されるのです。つまり、CoffeScript で書けば、結果的には JavaScript のコードを書いたことになるのです。しかも、CoffeeScript は、JavaScript に比べて、簡潔で、プログラムの可読性が向上しています。

また、JavaScript と比べてもパフォーマンスを下げることなく、より短いコードでプログラムを記述できるのです。そのために、多くの開発者が CoffeeScript を愛用しています。

```
CoffeeScript -(コンパイル)-> JavaScript -> (実行)
```

ところで、CoffeeScript って JavaScript と比べたら難しそうという印象の方もいるかもしれませんね。でも、安心してください。CoffeeScript は、JavaScript と別物というほどかけ離れたプログラミング言語ではありません。ここでは、プログラムを読むための CoffeeScript の基礎的な部分を紹介します。JavaScript とそれほど違うわけではないので、習得も困難ではないでしょう。

CoffeeScript のインストール

CoffeeScript のインストールは、npm を利用して行うことができます。以下のコマンドを実行するとインストールが完了します。

```
$ sudo npm install -g coffee-script
```

プログラムの実行の方法

簡単なプログラムを書いて、その実行方法を確認していきましょう。以下に 1 から 10 までの数を合計するプログラムを作ってみます。

● file: src/ch04/04-coffee/test-sum.coffee

```
v = 0
for i in [1..10]
  v += i
console.log(v)
```

実行するには、以下のようなコマンドを実行します。

```
$ coffee test-sum.coffee
```

また、明示的に JavaScript へ変換するには、次のように記述します。以下のコマンドを実行すると、同じディレクトリに「test-sum.js」が出力されます。

```
$ coffee -c test-sum.coffee
```

上記の CoffeeScript のソースコードは以下のような JavaScript にコンパイルされました。面白いですね。

● file: src/ch04/04-coffee/test-sum.js

```
// Generated by CoffeeScript 1.9.2
(function() {
  var i, j, v;

  v = 0;

  for (i = j = 1; j <= 10; i = ++j) {
    v += i;
  }

  console.log(v);

}).call(this);
```

CoffeeScript の基本文法をチェック

では、CoffeeScript の基本文法をチェックしていきましょう。まず、JavaScript と違って、文末にセミコロン「;」が不要です。

```
console.log "Hello"
```

ただし、複数の文を一行に記述したい場合には、セミコロンを使って文を区切る必要があります。

```
console.log "Hello"; console.log "World"
```

上記の例を見てもわかると思いますが、関数やメソッドの引数を指定する際に括弧 (...) が不要です。
下記の記述は関数を二つ並べて記述したものです。

```
console.log Math.floor 3.15
```

上記の記述は、以下のように記述したのと同じ意味になります。したがって「3」が表示されます。

```
console.log( Math.floor(3.15) )
```

括弧は書いても書かなくても良いのですが、引数の対応関係が不明になる場合には、明示的に括弧で囲っておくと良いでしょう。

CoffeeScript のコメント

CoffeeScript では、コメントが二種類あります。単一行コメントが「# xxx」の形式。複数行コメントは「### ... ###」の形式です。

```
# 単一行コメント

###
複数行の
コメント
###
```

単一行のコメントはコンパイル時に捨てられますが、複数行のコメントは、生成された JavaScript にそのまま残されます。そのため、ファイルの先頭にライセンスを埋め込みたい場合などは、複数行のコメントを使うと良いでしょう。

```
###
xx を oo するプログラム
作者： xxx (xxx@****.org)
Released under the MIT License
###
```

制御構文はインデントでレベルを明示する

CoffeeScript にも、JavaScript と同じように、if/switch/for/while などの制御構文があります。ただし、JavaScript と違って、インデント（字下げ）でブロックを表現することになっています。

● file: src/ch04/04-coffee/indent.coffee

```
name = "Joseph"
if name.length > 1
  console.log "Hello, #{name}!"
else
  console.log "empty name"
```

変数の宣言

JavaScript では変数を宣言する際、var キーワードが必要ですが、CoffeeScript では不要です。値を最初に代入した場所で変数が生成され、スコープが決定されます。

```
hoge = 30;
```

これを JavaScript にコンパイルすると以下のようになります。

```
var hoge;
hoge = 30;
```

文字列について

文字列リテラルは、ダブルクォーテーション ("...") あるいは、シングルクォーテーション ('...') で囲って表現します。ダブルクォーテーションで囲んだ文字列には、変数を埋め込んで記述することができます。

● file: src/ch04/04-coffee/str.coffee

```
# 変数の宣言
name = "John"
age = 24
console.log "私は、#{name}と申します。"
console.log "今年で、#{age}才になります。"
```

上記のプログラムを実行すると、文字列の中の「#{ 変数 }」が展開されて、以下のように表示されます。

```
$ coffee str.coffee
私は、John と申します。
今年で、24 才になります。
```

複数行の文字列を記述するのに便利な、ヒアドキュメントも用意されています。これは、文字列展開のある """...""" と、展開のない '...' の二種類があります。

● file: src/ch04/04-coffee/str-hdoc.coffee

```
eye = "目"
message = """
  体のともしびは #{eye} です。
    もし #{eye} が純一であれば
      あなたの体全体は明るいでしょう。
  """
console.log message
```

上記のプログラムを実行すると、以下のように表示されます。

```
$ coffee str-hdoc.coffee
体のともしびは目です。
  もし目が純一であれば
    あなたの体全体は明るいでしょう。
```

上記の結果を見るとわかりますが、ヒアドキュメントの中で行頭に空白があっても、その空白を無視するようになっています。また、文字列中の「#{ 変数名 }」もしっかり展開されます。

真と偽

JavaScript では真偽値として、true と false を利用できました。CoffeeScript では、これに加えて on と off、yes と no を指定することができます。意味としては、true と false と同じなのですが、文脈に応じて使い分けるとプログラムが読みやすくなるでしょう。

配列

配列やオブジェクトを指定するとき、JavaScript と同じく、配列を [...] で、オブジェクトを {...} で表現できます。

ただし、配列の区切り文字として、カンマ「,」だけでなく改行も利用できます。また、最後の要素がカンマで終わってもエラーにならないように工夫されています。

```
fruits = [
  "Orange"
  "Banana"
  "Apple"
]

info = {
  id: "Ba32103"
  price: 300
  origin: "Okinawa"
};

oslist = ["Windows", "OS X", "Linux",]
```

YAML と同じように、インデントで階層を付けると、入れ子のオブジェクトを表現することができます。

● file: src/ch04/04-coffee/like-yaml.coffee

```
fruits =
  Orange:
    price: 200
    origin: "Shizuoka"
  Banana:
    price: 240
    origin: "Okinawa"
  Apple:
    price: 400
    origin: "Aomori"

console.log JSON.stringify fruits
```

JSON で表現すると、以下のようになります。

```
{
  "Orange":{"price":200,"origin":"Shizuoka"},
  "Banana":{"price":240,"origin":"Okinawa"},
  "Apple":{"price":400,"origin":"Aomori"}
}
```

演算子

演算子は、JavaScript とほぼ同じですが、以下のような演算子も用意されています。

CoffeeScript	JavaSCript	利用例
is	===	a is "hoge"
isnt	!==	a isnt "hoge"
not	!	not true
and	&&	a and b
or	||	a or b

CoffeeScript で使える演算子

その他に気をつけなければならないのが、CoffeeScript で比較演算子「==」は、JavaScript の「===」に変換されるという点です。JavaScript で「===」は、厳密な比較を表し、変数の値だけでなく型が厳密に一致していないと false が返ります。

CoffeeScript にも対話式インタプリタの REPL が用意されていますので、REPL で結果を確認してみましょう。

```
$ coffee
coffee> 3 == "3"       # --- (※1)
false
coffee> 3 == 3
true
coffee> "0xFF" == 255 # --- (※2)
false
coffee> parseInt("0xFF") == 255
true
```

例えば、(※1) の部分ですが、JavaScript で、(3 == "3") と書けば true となるのですが、CoffeeScript では false となります。同様に、JavaScript で ("0xff" == 255) は等しく true になりますが、CoffeeScript では false となります。

範囲の比較を行う

JavaScript では「10 <= x && x <= 30」のように記述することがあります。これは、10 以上 30 以下を表す式です。CoffeeScript では、これを「10 <= x <= 30」と書くことができます。両者を比較してみると、CoffeScript のほうが直感的に記述できることがわかります。

変数の存在をチェックする

変数が未定義かどうか調べるのに「?」演算子を利用することができます。これは、変数が未定義で、かつ、null ではないかどうかを調べることができます。

```
obj = { name:"fuga", age:30 }
console.log obj.name?      # 結果 → true
console.log obj.weight?    # 結果 → false
```

この「?」演算子を利用することで、変数が未定義の場合にデフォルト値を指定することができます。

```
value = sval ? 100
```

関数やメソッドを呼び出す場合にも、存在演算子を利用することができます。例えば、関数が存在する場合のみ関数を実行するとか、オブジェクトが存在する場合のみ、そのプロパティにアクセスするといった場合です。

```
# 関数が存在すれば実行
console.log func?()

# オブジェクト obj が存在する場合のみ price にアクセス
console.log obj?.price
```

CoffeeScript では、「?」が存在演算子となるとなるため、JavaScript の参考演算子は利用できません。

連続する数値の表現ー範囲演算子「..」

範囲演算子「n..m」が利用できるできるので、数値配列を手軽に生成できます。例えば、1 から 10 の値を生成する場合には、以下のように記述できます。

```
range = [1..10];
```

また、範囲演算子には「n...m」(ドットが 3 つ) のものもあり、これを使う場合は、末尾の値が除かれます。例えば、[1...10] と書いた場合、1 から 9 までの値を指定した [1..9] と等しくなります。

制御構文

次に、CoffeeScript の制御構文を一つずつ確認していきます。CoffeeScript の制御構文ブロックはインデントで表現します。

if..else..

インデントを使うので、条件分岐はシンプルに記述できます。以下のように if 構文を連続でつなぐこともできます。

● file: src/ch04/04-coffee/if.coffee

```
age = 18
if age < 20
  console.log "少年"
else if age < 30
  console.log "青年"
else if age < 50
  console.log "中年"
else
   console.log "老年"
```

そして、後置形式の if も指定できます。この場合、条件式が真の時のみ、前方の式が実行されることになります。

● file: src/ch04/04-coffee/if2.coffee

```
x = 50
console.log "30" if x is 30
console.log "50" if x is 50
console.log "70" if x is 70
```

上記のプログラムを実行してみます。x が 50 なので、if x is 50 を指定した式が実行されることになります。

```
$ coffee if2.coffee
50
```

if..then..else

また、一行で if 構文を書くときには、if .. then .. else .. と「then」を使うことができます。JavaScript の三項演算子のように使うことができます。

● file: src/ch04/04-coffee/if3.coffee

```
age = 12
price = if age > 12 then 500 else 250
console.log "#{age}才は、#{price}円です"
```

```
$ coffee if3.coffee
12才は、250円です
```

unless

if 構文は、条件式が true のとき続くブロックを実行しましたが、unless 構文は、条件式が false の場合に続くブロックを実行します。

```
v = 20
unless v is 15
  console.log "#{v} != 15"
```

switch..when..else

式の値に応じて処理を分岐するのが、switch 構文です。

JavaScript の switch と異なり、case 句の代わりに when 句を使い、default 句の代わりに else 句を利用します。when 句には、カンマで区切って複数の値を指定できます。条件句を抜けるための break は不要で、次の when が来たらそこで処理を自動的に抜けます。

● file: src/ch04/04-coffee/switch.coffee

```
star = 5
switch star
  when 1
    console.log "格安ホテル"
  when 2
    console.log "安めのホテル"
  when 3
    console.log "普通のホテル"
  when 4
    console.log "良いホテル"
  when 5
    console.log "豪華なホテル"
  else
    console.log "判定不能"
```

while / until

条件式が true の間、繰り返しブロックを実行するのが、while 構文です。これに対して、条件式が false の間、繰り返すのが、until 構文です。

● file: src/ch04/04-coffee/while.coffee

```
i = 1
while i <= 5
  console.log i
  i++

console.log "---"

until i < 0
  console.log i
  i--
```

```
$ coffee while.coffee
1
2
3
4
5
---
6
5
4
3
2
1
0
```

for..in / for..of

配列から順に値を取り出して特定の処理を繰り返すのが、for..in 構文です。そして、オブジェクトから順に値を取り出して処理を繰り返すのが、for..of 構文です。

まずは、配列の各要素を表示する、for..in 構文の利用例から見てみましょう。

● file: src/ch04/04-coffee/for.coffee

```
fruits = [ "Banana", "Mango", "Apple", "Orange" ]
for name in fruits
  console.log "I like #{name}."
```

実行すると、以下のようになります。

```
$ coffee for.coffee
I like Banana.
I like Mango.
I like Apple.
I like Orange.
```

JavaScript の for..in 構文では、反復子の変数に要素番号が代入されるだけでしたが、CoffeeScript では、要素そのものが代入されるので便利です。要素番号自体が必要な場合には、「for value, index in 配列」のような書式で記述します。

● file: src/ch04/04-coffee/for2.coffee

```
fruits = [ "Banana", "Mango", "Apple", "Orange" ]
for name,index in fruits
  console.log "#{index}: #{name}"
```

このプログラムを実行してみます。

```
$ coffee for2.coffee
0: Banana
1: Mango
2: Apple
3: Orange
```

次に、オブジェクトの各要素を繰り返す、for..of 構文の利用例を見てみましょう。

● file: src/ch04/04-coffee/for_of.coffee

```
mail_info = {
  subject: "こんにちは"
  from: "test@example.com"
  body: "お久しぶりです。げんきですか？"
}

for key, value of mail_info
  console.log "#{key} : #{value}"
```

このプログラムを実行してみます。

```
$ coffee for_of.coffee
subject : こんにちは
from : test@example.com
body : お久しぶりです。げんきですか？
```

関数の記述

関数を記述するには、次の書式で書きます。ちなみに、return を記述する必要なく、最後の式が戻り値となります。

```
（引数1，引数2，...）-> 式
```

では、簡単にかけ算を行うだけの関数を定義してみます。以下を実行すると、2x3 で「6」が表示されます。

● file: src/ch04/04-coffee/func.coffee

```
multiply = (a, b) ->
  a * b

console.log multiply 2, 3
```

次に、二乗を計算する x2 と、四乗を計算する x4 という関数を定義する例を紹介します。

● file: src/ch04/04-coffee/func2.coffee

```
x2 = (num) -> num * num
x4 = (num) -> num * num * num * num

console.log( x2(3) + x4(2) )
```

3 の二乗で 9、2 の四乗で 16、これを足して、25 が答えです。

ところで、引数がない関数を記述する場合には、（引数）の部分を省略することができます。しかし、関数を呼び出す際には、関数名の後に () が必要になります。また、戻り値が不要の場合には、return 文を記述します。

```
# 引数のない関数を定義
func = ->
  console.log "hello"
  # 戻り値のない関数
  return

# 引数のない関数を呼び出す
func()
```

引数のデフォルト値

関数定義の際に、引数のデフォルト値を指定することができます。

● file: src/ch04/04-coffee/func-def.coffee

```
# 文字列を delimiter で区切って返す関数
splitStr = (str, delimiter = ",") ->
  str.split(delimiter)

# delimiter を省略
s1 = "1,2,3"
console.log splitStr(s1)

# 明示的に delimiter を指定
s2 = "a:b:c"
console.log splitStr(s2, ":")
```

```
$ coffee func-def.coffee
[ '1', '2', '3' ]
[ 'a', 'b', 'c' ]
```

可変長引数

関数定義の際に、引数の末尾に「...」を記述すると、可変長引数を指定できます。可変長引数は配列の形で取得できます。

● file: src/ch04/04-coffee/func-va.coffee

```
sum = (args...) ->
  total = 0
  for arg in args
    total += arg
  total

console.log sum 1,2,3
console.log sum 3,4,5
```

上記のプログラムを実行してみます。

```
$ coffee func-va.coffee
6
12
```

無名関数（匿名関数）

無名関数も同じように、「(引数)->」の書式で記述することができます。以下は、定期的にコールバック関数を実行する setInterval() 関数の利用例となっています。一秒ごとに無名関数の形で指定したコールバック関数が実行されます。

● file: src/ch04/04-coffee/func-nameless.coffee

```
counter = 5
setInterval ->
  console.log counter
  counter--
  if counter is 0
    process.exit()
,1000
```

実行すると、1 秒に一度数字がコンソールに出力されます。

```
$ coffee func-nameless.coffee
5
4
3
2
1
```

オブジェクト指向

JavaScript にもオブジェクト指向はあるのですが、プロトタイプベースのオブジェクト指向であり、わかりにくいものでした。CoffeeScript では、一般的なオブジェクト指向言語のようにして、クラスを定義することができます。これは大きなメリットです。

クラス定義

クラス定義の例を見てみましょう。Animal クラスを定義し、インスタンスを生成してみます。クラスのメソッド定義の中でプロパティを参照したい場合には「@ 変数名」のように記述します。

● file: src/ch04/04-coffee/class.coffee

```
# Animal クラスを定義
class Animal
  # プロパティの定義
  atype: "Animal"

  #コンストラクタを定義
  constructor: (@name) ->
    # name プロパティは自動的に設定される

  #メソッドを定義
  print: ->
    console.log "名前は #{@name}, 種類は #{@atype} です。"
```

```
# インスタンスを生成
taro = new Animal "Taro"
taro.print()
```

```
$ coffee class.coffee
名前は Taro，種類は Animal です。
```

継承

では、Animal を継承して Dog クラス、Cat クラスを定義してみます。

● file: src/ch04/04-coffee/class2.coffee

```
# Animal クラスを定義
class Animal
  # プロパティの定義
  atype: "Animal"

  # コンストラクタを定義
  constructor: (@name) ->
    # name プロパティは自動的に設定される

  # メソッドを定義
  print: ->
    console.log "名前は #{@name}，種類は #{@atype} です。"

class Dog extends Animal
  atype: "Dog"
  print: ->
    console.log "わんわん"
    super()

class Cat extends Animal
  atype: "Cat"
  print: ->
    console.log "にゃーにゃー"
    super()

# インスタンスを生成
jiro = new Dog "Jiro"
jiro.print()

mike = new Cat "Mike"
mike.print()
```

上記プログラムを実行してみましょう。

```
$ coffee class2.coffee
わんわん
名前は Jiro，種類は Dog です。
にゃーにゃー
名前は Mike，種類は Cat です。
```

静的メンバー

静的なクラスメソッドやプロパティを定義するには、「@ プロパティ」のように、名前の前に「 @」を記述します。

● file: src/ch04/04-coffee/class-static.coffee

```
class Calc
  # 静的プロパティ
  @pi: 3.1415
  # 静的メソッド
  @mul: (a, b) -> a * b
  @div: (a, b) -> a / b
  @mod: (a, b) -> a % b

console.log Calc.pi
console.log Calc.mul 2,3
```

このプログラムを実行してみましょう。

```
$ coffee class-static.coffee
3.1415
6
```

メンバーを動的に追加する

CoffeeScript では、既存クラスへ動的にメンバーを追加することができます。この場合「クラス名 :: メンバー = 値」のようにして追加できます。

```
Animal::food = "Pet food"
```

ちなみに、CoffeeScript の「@」は自分自身を表すキーワードとなっており、JavaScript の this と同じ意味です。

この節のまとめ

- この節では、CoffeeScript の基本的な構文を一通り紹介しました。
- CoffeeScript は最終的に JavaScript に変換されるので、多少意味のわからない CoffeeScript のコードでも、変換するとわかる場合があります。
- CoffeeScript が評価されている理由の一つが、コンパイル後のソースコードが見やすいという点にあります。
- もしも「この CoffeeScript 訳がわからない」と思ったら、JavaScript に変換してみてください。

05 データベースの使い方

第 3 節でさまざまなデータ形式について考察しました。ここでは、データベースについて紹介します。JavaScript から利用できるどんなデータベースがあるでしょうか。

ここで学ぶポイント

- データベースについて
- データベースの扱い方

ツールやライブラリの一覧

- Node.js
- SQLite (関係データベース)
- LevelDB (Key-Value型データベース)

データベースをなぜ使うのか？

「餅は餅屋」ということわざがあります。餅は餅屋のついたものがいちばんうまい、その道のことはやはり専門家が一番であるというたとえですが、これは、データの保存に関しても同じ事が言えます。

Web からさまざまなデータをダウンロードしてきて、それを上手に整理して保存したいと思った時に、データベースに保存していくのは自然なことでしょう。

関係データモデルと NoSQL

現在主流は、リレーショナルデータモデル（関係データベース）を採用したデータベースです。これは、データを複数の表として管理し、表同士が関係（リレーション）と呼ばれる構造で相互連結可能となっています。多くのリレーショナルデータベースでは、SQL と呼ばれる問い合わせ言語を用いて、データベースを操作します。

有名なものには、Oracle Database、Microsoft SQL Server、MySQL、PostgreSQL、DB2、FileMaker、H2 Database などがあります。

これに対して、関係データモデルを利用しないデータベースモデルを「NoSQL」と言います。関係データモデルではないデータストアの特徴として、固定されたスキーマに縛られないこと、関係モデルの結合操作を利用しないことなどが挙げられます。「キー」と「値」を組み合わせ、それを入出力するシンプルな Key-Value 型データベースもあります。これを KVS(Key-Value Store) と略すこともあります。

NoSQL が活躍する場面は、関係モデルを必要としないデータを扱う時や、大量のデータを扱う時です。用途は多様であり、数百万の key-value ペアを格納したり、数個程度の連想配列を格納したり、数百万の構造的データを格納したりとさまざまです。この構造は、大規模なデータを統計的に解析したり、増えつづける情報をリアルタイムに解析したりするのにも便利です。

有名なものには、Google の BigTable、Amazon DynamoDB などがあります。また、オープンソースの実装も数多く存在し、MongoDB、Redis、Apache HBase、Apache Cassandra などがあります。

関係データベースから「SQLite3」を使う

関係データベースに関して、Node.js では、MySQL や Oracle、PostgreSQL など有名なデータベースのライブラリが揃っています。JavaScript からこれらのデータベースを利用できるのは、非常に便利です。

本書では、データベースのセットアップの手間を省けること、また、手軽に使えることを考えて、組み込みデータベースの「SQLite」を使ってみることにします。

Node.js で SQLite を利用するために、npm を利用して「sqlite3」モジュールをインストールしましょう。

```
$ npm install sqlite3
```

データベースを作成し、そこにデータを挿入し、データを取り出して表示するという簡単なサンプルを見てみましょう。

● file: src/ch04/05-db/sqlite-test.js

```
// モジュールの取り込み
var sqlite3 = require('sqlite3').verbose();
// ローカル DB を開く
var db = new sqlite3.Database('test.sqlite');

db.serialize(function () {
  // SQL を実行してテーブルを作成 ---(※1)
  db.run('CREATE TABLE IF NOT EXISTS items(name, value)');

  // プリペアードステートメントでデータを挿入
  var stmt = db.prepare('INSERT INTO items VALUES(?,?)');
  stmt.run(["Banana", 300]);
  stmt.run(["Apple", 150]);
  stmt.run(["Mango", 250]);
  stmt.finalize();

  // データを取り出す
  db.each("SELECT * FROM items", function (err, row) {
    console.log(row.name + ":" + row.value);
  });
});
db.close();
```

では、このプログラムを実行してみましょう。

```
$ node sqlite-test.js
Banana:300
Apple:150
Mango:250
```

SQLite3 を使うサンプルは、これ以上でも、これ以下でもないでしょう。ここでは、SQL を知っているという前提で、プログラムを解説します。

最初に SQLite3 のモジュールを取り込み、データベースを開きます。serialize() メソッドの中では、テーブル作成やデータ挿入、抽出などの操作して、最後に close() メソッドでデータベースを閉じています。

次に db.run() メソッドを使うことで、任意の SQL を実行することができます。db.prepare() メソッドを使うと、プリペアードステートメントを利用することができます。これは、SQL を事前にコンパイルし、部分的にデータを差し込んで実行することができるというものです。SQL をコンパイルするので処理速度が速いことや、挿入するデータに含まれる特殊記号を自動的にエスケープしてくれるのでセキュリティ的にも安心で、便利な機能です。

また、db.each() メソッドを使うと、SQL を実行した結果、データベースから抽出したデータを得ることができます。コールバック関数で各レコードを一件ずつ取り出すことができます。

補足ですが、プログラム (※ 1) では、SQL の「CREATE TABLE」を実行しています。これはテーブルを新規作成するコマンドですが、すでに同名のテーブルが存在する時にはエラーとなります。もし、プログラムを複数回実行したい場合などには、ここで記述したような「CREATE TABLE IF NOT EXISTS」にします。この SQL は「もしテーブルが存在しなかったときだけ、テーブルを作成し、すでにテーブルがある場合には何も処理を行わない」という意味になります。

Web からダウンロードして SQLite に保存しよう

ここまでの部分で、SQLite の基本的な使い方がわかりましたので、もう少し実践的なプログラムを作ってみましょう。2 章で Web データの収集について扱いましたので、それを踏まえたものにしてみます。

ここでは、「青空文庫」の人気作品をダウンロードして、SQLite に格納するプログラムを作ってみます。ちなみに、「青空文庫」は日本国内において著作権が消滅した文学作品などのテキストを公開している Web サイトです。興味深いことに、定期的に文学作品の人気ランキングを公開しています。そこで、その人気ランキングを元にして、上位 30 件の作品をダウンロードしてみたいと思います。

ここでは、2014 年の通年ダウンロードランキングを元にしてみます。

青空文庫 > XHTML 版アクセスランキング
`http://www.aozora.gr.jp/access_ranking/2014_xhtml.html`

XHTML版 アクセスランキング (2014.01.01 - 2014.12.31)

ランキング	作品名 副題	著者名	アクセス数
1	こころ	夏目 漱石	205548
2	〔雨ニモマケズ〕	宮沢 賢治	179672
3	走れメロス	太宰 治	146161
4	吾輩は猫である	夏目 漱石	107807
5	ドグラ・マグラ	夢野 久作	103555
6	山月記	中島 敦	91861
7	やまなし	宮沢 賢治	83536
8	羅生門	芥川 竜之介	83327
9	人間失格	太宰 治	80719
10	銀河鉄道の夜	宮沢 賢治	77365
11	坊っちゃん	夏目 漱石	75596
12	夢十夜	夏目 漱石	74675
13	蜘蛛の糸	芥川 竜之介	70487
14	注文の多い料理店	宮沢 賢治	66067
15	セメント樽の中の手紙	葉山 嘉樹	63129
16	草枕	夏目 漱石	59571
17	戦争責任者の問題	伊丹 万作	57505

青空文庫のランキング

ダウンロードとデータベースの格納を同時に行うのも良いのですが、ここでは、敢えて、ダウンロードだけのプログラムと、ダウンロードしたファイルをデータベースに格納するという二つのプログラムに分けてみます。本来は、ダウンロードしたデータを直接データベースに挿入していく方が良いのですが、ここでは、それぞれのプログラムを簡単にするために分けることにします。

では、まず、青空文庫のアクセスランキングから上位作品をダウンロードするプログラムから見ていきましょう。

● file: src/ch04/05-db/dl-aozora-files.js

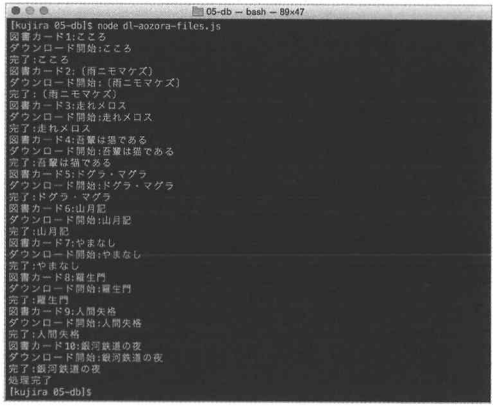
```
//
// 青空文庫の人気作品 30 をダウンロードする for Node.js
//
// 人気作品の 2014 年ランキングのページ
var URL_RANKING = "http://www.aozora.gr.jp/access_ranking/2014_xhtml.html";

中略

// 作品データ保存用のディレクトリを作る --- (※1)
if (!fs.existsSync(SAVE_DIR)) fs.mkdirSync(SAVE_DIR);

// ランキングページをダウンロード ---- (※2)
client.fetch(URL_RANKING, function (err, $, res) {
  if (err) { console.log("DL error"); return; }
  // ランキングのテーブル全行を取得
  var tr = $("table.list tr");

中略

    // 相対パスを絶対パスに変換 --- (※3)
    href = URL.resolve(URL_RANKING, href);
    cardlist.push([rank, name, href]);
  }
  downloadNextFile();
});

// 各作品をダウンロードする ---- (※4)
function downloadNextFile() {
  if (cardlist.length == 0) {
    console.log("処理完了");
    return;
  }
  // 遅延処理させる
  setTimeout(function(){
    var card = cardlist.shift();
    downloadCard(card);
  },1000);
}

// カードをダウンロードする --- (※5)
function downloadCard(card) {
  var index = card[0], name = card[1], link = card[2];
  console.log("図書カード" + index + ":" + name);
  client.fetch(link, function(err, $, res){
    if (err) { console.log("ERROR"); return; }
    // 全てのリンクを取得し作品ページを類推する
    var xhtml_link = "";
    $("a").each(function(idx){
      var text = $(this).text();
      var href = $(this).attr('href');
      if (text.indexOf("XHTML 版で読む") >= 0) {
```

```
        // 相対パスを絶対パスに変換
        href = URL.resolve(link, href);
        xhtml_link = href;
        return false; // これ以後 each しない
      }
    });
    if (xhtml_link == "") {
      console.log("作品リンクが見つかりません");
    }
    // 作品をダウンロードする
    var path = SAVE_DIR + "/" + index + ".html";
    console.log("ダウンロード開始:" + name);
    // 作品のダウンロード時 User-Agent の明示が必要 ---- (※6)
    client.setBrowser('chrome');
    client.fetch(xhtml_link, function (err, $, res, body){
      body = body.replace(/Shift_JIS/ig, "UTF-8"); // --- (※7)
      fs.writeFileSync(path, body, "utf-8");
      console.log("完了:" + name);
      downloadNextFile();
    });
  });
}
```

このプログラムを実行するには、以下のコマンドを入力します。

```
$ node dl-aozora-files.js
```

aozoraというディレクトリが作成され、そのなかに、1.html、2.html、3.html... と「(ランキング).html」のファイル名で、合計30個のHTMLファイルがダウンロードされます。



コマンドを実行したところ

上記のプログラムは、100行未満の小さなプログラムですが、プログラムを完成させるにあたって、いくつか難所がありました。まず、ランキングページからデータを取り出してみたところ、直接作品にリンクされているのではなく、作品の紹介ページ（図書カードのページ）にリンクされています。ですから、作品をダウンロードするためには「ランキングページ>図書カードページ>作品」と順にリンクをたどってダウンロードする必要があります。

さらに加えて、青空文庫の方では、サーバーの負荷軽減対策が施されているという点です。今回作ったプログラムはWebサイトのスクレイピングを行うものですが、一気にアクセスを行うと、サーバー負荷を

与えて、他のサーバーの管理者や他の閲覧者に迷惑をかけてしまいます。そこで、遅延を入れつつファイルを一つずつダウンロードを行うようにする必要があります。

さらに、User-Agent を設定しないとダウンロードがうまくいかないという問題もありました。

データのダウンロードに関する詳細は２章を参照してください。ここでは、ポイントだけを解説します。プログラムの (※ 1) では、作品をダウンロードするディレクトリを作成しています。ダウンロードが始まる前にディレクトリを作成しなくてはならないので、同期的に実行する fs.mkdirSync() を利用しています。(※ 2) では、ランキングページをダウンロードし、テーブルタグを解析して、そこからランキング情報を取り出します。ところで、HTML から取り出したリンクは一般的に相対パスで記述されてます。うっかりしがちですが、リンクを抽出したら絶対パスを得るのを忘れないようにしましょう (※ 3 の部分)。

さて、各作品をダウンロードする (※ 4) の部分ですが、一気に作品にアクセスしないようわざと遅延実行を行っています。ここでは、setTimeout() メソッドを利用して、作品をダウンロードしたら 1 秒 (1000 ミリ秒) 遅延するようにしました。また、作品一覧は配列変数 cardlist に記録しているのですが、配列の要素を一つずつ取り出してダウンロードを行うようにしました。そして、ダウンロードが完了するたびに、downloadNextFile() 関数が呼ばれるようになっています。

カードをダウンロードしている (※ 5) の部分ですが、まず図書カードのページを取得し、そのページ内のリンクを全部抽出します。リンクテキストを調べて「XHTML 版で読む」というものがあれば、それが作品へのリンクです。そして、一番悩んだ点の答えが (※ 6) です。作品をダウンロードする際には、User-Agent を明示しないと、アクセスが弾かれてしまいました。そこで、ここでは、Google Chrome と同じ User-Agent に変更しています。

青空文庫のテキストは Shift_JIS で保存されていますが、「cheerio-httpcli」モジュールを使ってデータを取得すると、Node.js でデータを扱いやすく UTF-8 へ自動的に変換してしてくれることもポイントです。そこで、プログラム (※ 7) では、HTML に記述されている「Shift_JIS」を「UTF-8」へ強制的に置換しています。

SQLite に作品を保存しよう

それでは、ダウンロードした文学作品をデータベースに保存してみます。既にダウンロード済みの HTML ファイルからも作者や作品名を取り出したいので、HTML を jQuery ライクに解析できるモジュール「cheerio」を利用してみます。

```
$ npm install cheerio
```

作品を登録するだけでは面白みがないので、作品の作者のランキングベスト 30 の中での登場回数を調べてみましょう。

● file: src/ch04/05-db/sqlite-aozora.js

```
// ダウンロードしたファイルを SQLite に流し込む for Node.js

中略

// DB に入れるファイル一覧を取得 --- (※1)
var files = fs.readdirSync(FILES_DIR);
// HTML ファイルだけ残す
files = files.filter(function(s){
  return s.match(/\.html$/);
});

// データベースを開く --- (※2)
var db = new sqlite3.Database(DB_PATH);
```

```
// データを登録
db.serialize(function(){
  // SQLを実行してテーブルを作成
  db.run("CREATE TABLE IF NOT EXISTS items(" +
    "item_id INTEGER PRIMARY KEY, " +
    "author TEXT, title TEXT, body TEXT)");
  // 挿入用プリペアドステートメントを準備
  var ins_stmt = db.prepare(
    'INSERT INTO items(author, title, body)' +
    'VALUES(?, ?, ?)');
  // 各HTMLファイルを処理
  files.forEach(function(file, i, ar) {
    var html = fs.readFileSync(FILES_DIR + "/" + file);
    // HTMLファイルから情報を得る
    var $ = cheerio.load(html);
    var title = $(".title").text();
    var author = $(".author").text();
    var body = $('body').text();
    // DBに挿入
    ins_stmt.run(author, title, body);
    console.log("+ " + title + " を登録");
  });
  ins_stmt.finalize();
});

// 作者の出現回数を調べる --- (※3)
console.log("集計結果:");
db.each("SELECT author,COUNT(author) as cnt "
  + "FROM items GROUP BY author "
  + "ORDER BY cnt DESC",
  function (err, row){
    console.log(row.cnt + "回:" + row.author);
  });
```

このプログラムを実行するには以下のコマンドを記述します。

```
$ node sqlite-aozora.js
```

実行したところ

試しに集計しただけですが、これを見ると、ベスト 30 の中の作品には、「夏目漱石」や「芥川龍之介」「宮沢賢治」の登場回数が多いことがわかりました。これらの作家は国語の教科書にも出てきますから、納得がいきますね。

プログラムの流れをつかんでおきましょう。

プログラムの (※ 1) では、DB に挿入するファイル一覧を取得しています。readdirSync() メソッドでは、ディレクトリにあるすべてのファイルを取得してしまいますので、filter() メソッドを利用して、HTML ファイルだけを抽出します。

次いで、プログラムの (※ 2) 以降の部分では、データベースを開いて、データを挿入します。ここでは、HTML ファイルに記述されている、作品名・著者情報を取り出しています。また、XHTML ファイルの全体を保存するのではなく、テキスト情報だけを取り出してデータベースに保存するようにしています。

最後に、プログラムの (※ 3) の部分では、データベース中に挿入した作品の作者の名前を数えて、多い順に表示しています。SQL を使うとこうした集計が手軽に行えるメリットがあります。

NoSQL から「LevelDB」を使ってみよう

次に、関係データベースモデルを利用しない、LevelDB データベースを利用してみたいと思います。

そもそも「LevelDB」とは、Key-Value 型のデータストアの一つで、Google の研究者が開発したものです。C++ で書かれていますが、多くのプログラミング言語から利用できるようになっています。ちなみに、HTML5 の仕様の一つに「IndexedDB」の仕様がありますが、Web ブラウザーの「Google Chrome」で IndexedDB を実装するために、LevelDB が開発されたのだそうです。「LevelDB」は組み込み用途で使えるデータベースで、非常に手軽に使うことができます。では、さっそく使ってみましょう。

モジュールのインストールは以下のように行います。

```
$ npm install level
```

LevelDB は Key-Value 型のデータベースなので基本的な使い方はとてもシンプルです。では、実際の利用例を見てみましょう。

● file: src/ch04/05-db/leveldb-test.js

```
// LevelDB の利用例 for Node.js

// モジュールの取り込みと DB を開く --- (※1)
var levelup = require('level');
var db = levelup('./testdb');

// 値を設定 ---- (※2)
db.put('Apple', 'red', function (err) {
  if (err) { console.log('Error', err); return; }
  testGet();
});

// 値を取得 ---- (※3)
function testGet() {
  db.get('Apple', function (err, value) {
    if (err) { console.log('Error', err); return; }
    console.log('Apple=' + value);
    testBatch();
  });
}
```

```
// 一括設定 ---- (※4)
function testBatch() {
db.batch()
.put('Mango', 'yellow')
.put('Banana', 'yellow')
.put('Kiwi', 'green')
.write(function(){ testGet2(); });
}
// 値を取得 その2
function testGet2() {
  db.get('Banana', function (err, value){
    console.log('Banana='+value);
    testKeys();
  });
}
```

プログラムを実行するには、以下のコマンドを入力します。二行目以降が実行結果です。

```
$ node leveldb-test.js
Apple=red
Banana=yellow
```

まず、プログラムの(※1)ですが、モジュールを取り込み、levelup()メソッドでデータベースを開きます。そして、(※2)のように、put()メソッドを実行すると、値を保存することができます。(※3)では、値を取得します。基本的な書式は次の通りです。

```
[書式] 値の設定
db.put(key, value, function(err) { ... })
```

```
[書式] 値の取得
db.get(key, function(err,value) { ... })
```

共に、Node.jsらしく、値が即時反映されるわけではなく、メソッドが成功したときに、コールバック関数に通知される仕組みとなっています。

プログラムの(※4)では、値の一括設定の方法について紹介したものです。db.batch()メソッドの後、write()メソッドまで、メソッドチェーンを利用して、連続で値を書き込むことができるようになっています。このとき、利用できるメソッドは、値を設定するput()のほか、キーを削除するdel()も利用できます。

LevelDBで検索したいとき

ところで、levelDBのようなKVS(Key-Value Store)では、どのようにして任意のデータを探したら良いのでしょうか。もう少し、詳細なlevelDBの使い方も確認しておきましょう。

以下は、levelDBで検索を行う例となっています。

● file: src/ch04/05-db/leveldb-test2.js

```
// levelDBの利用例 for Node.js

var levelup = require('level');
// データベースを開く(データはJSONで) --- (※1)
var opt = { valueEncoding:'json' };
```

```
var db = levelup('./testdb2', opt);

// 一括で値を設定 --- (※2)
db.batch()

中略

// キーの一覧を取得する ---- (※3)
function testKeys() {
  console.log("keys:")
  db.createKeyStream()
    .on('data', function (key) {
      console.log(" - " + key);
    })
    .on('end', testKeyValues);
}

// キーと値の一覧を取得する ---- (※4)
function testKeyValues() {
  console.log("\nkey-value-list:");
  db.createReadStream()
    .on('data', function (data) {
      var key = data.key;
      var o = data.value;
      console.log("+ key=" + data.key);
      console.log("| color=" + o.color);
      console.log("| price=" + o.price);
    })
    .on('end', testSearch);
}

// 検索を行う ---- (※5)
function testSearch() {
後略
```

このプログラムを実行するには、以下のコマンドを入力します。２行目以降が実行結果です。

```
$ node leveldb-test2.js
keys:
 - fruits!apple
 - fruits!banana
 - fruits!kiwi
 - fruits!orange
 - snack!choco
 - snack!poteto

key-value-list:
+ key=fruits!apple
| color=red
| price=300
+ key=fruits!banana
| color=yellow
| price=200
+ key=fruits!kiwi
| color=green
| price=220
+ key=fruits!orange
| color=orange
| price=180
+ key=snack!choco
```

```
| color=black
| price=220
+ key=snack!poteto
| color=brown
| price=340

range-search:
+ key=fruits!apple
+ key=fruits!banana
+ key=fruits!kiwi
+ key=fruits!orange
ok
```

では、プログラムを見ていきましょう。プログラムの冒頭(※1)の部分で、{valueEncoding:'json'} というオプションを付けてデータベースを開いています。これは、Key-Value 型の Value を常に JSON として保存することを意味します。つまり、値に JavaScript のオブジェクトを設定して、読み書きできるようにするということです。そして、(※2)では、実際に JavaScript のオブジェクトを設定しています。

続いて、プログラムの(※3)の部分では、キーの一覧を取得しています。このように、db.createKeyStream() メソッドを利用すると、キーの一覧を、data イベントで一つずつ取得することができます。

プログラムの(※4)では、キーと値の一覧を取得しています。ここで利用しているのは、createReadStream() メソッドです。このメソッドを使うと、key と value を含むオブジェクトを一気に取得することができます。

また、(※5)を見るとわかりますが、createReadStream() メソッドにオプションを付けると、キーの検索をすることができます。ここでは、「fruits!」から「fruits!\FF」までのキーを抽出します。これは、つまり、「fruits!」から始まるキーをすべて検索することになります。

もしも、この検索条件を、以下のようにするとどうなるでしょうか。

```
var opt = {
  start : "snack!",
  end   : "snack!\xFF"
};
```

そうです。この場合は、「snack!」から始まるデータを取り出すことができます。LevelDB のデータベースでは、特定のキーに合致するデータが取り出せるだけでなく、このように任意の範囲のデータを取り出すこともできます。

青空文庫のデータを LevelDB に保存

LevelDB の使い方がわかったところで、青空文庫のデータを LevelDB に保存してみましょう。このとき、工夫したいのは、何をキーにして保存するかという部分でしょう。できれば、作家で作品を検索できるようにしたいと思います。そこで、LevelDB のキーを「作者 : 作品名」として作成してみます。ただし、作品名でも検索したいと思いますので、検索用に作品名もキーに記録することにします。

それでは、先ほど HTML ファイルに保存した青空文庫の作品一覧を、LevelDB に保存し、簡単な検索をするプログラムを作ってみましょう。

● file: src/ch04/05-db/leveldb-aozora.js

```
// ダウンロードしたファイルを LevelDB に流し込む for Node.js

中略

// 各ファイルのデータを DB に入れる ---- (※1)
var count = 0;
files.forEach(function(file, i, ar) {
  // ファイルを開く
  var html = fs.readFileSync(FILES_DIR + "/" + file);
  // HTML ファイルから情報を得る
  var $ = cheerio.load(html);
  var title = $(".title").text();
  var author = $(".author").text();
  var body = $('body').text();
  // データベースに入れる ---- (※2)
  //「作者：作品名」で入れる
  var key = author + ":" + title;
  db.put(key, body, function () { count++; });
  // 作品名で検索できるようにも配慮 --- (※3)
  var key2 = "idx-title:" + title + ":" + author;
  db.put(key2, key);
  // console.log(key);
});

// 処理完了を待つ
var wait_proc = function () {
  if (files.length == count) {
    testSearch(); return;
  }
  setTimeout(wait_proc, 100);
};
wait_proc();

// 作者から作品一覧を検索 --- (※4)
function testSearch() {
  console.log("\n夏目漱石の作品一覧:");
  var opt = {
    start : '夏目漱石:',
    end   : '夏目漱石:\uFFFF'
  };
  db.createReadStream(opt)
    .on("data", function (data) {
      console.log(" - " + data.key);
    })
    .on("end", testSearch2);
}

// 作品名で検索する --- (※5)
function testSearch2() {
  var title = '注文の多い料理店';
  console.log("\n作品名 ["+title+"] で検索:");
  var opt = {
    gte: 'idx-title:' + title,
    lte: 'idx-title:' + title + "\uFFFF"
  };
  db.createReadStream(opt)
    .on("data", function (data) {
      console.log(" - " + data.value);
    })
```

```
    .on("end", testSearch3);
}

// 正規表現で作品を検索 --- (※6)
function testSearch3() {
  console.log("\n ひらがなの作品を検索 :");
  var opt = {
    gte: 'idx-title:',
    lte: 'idx-title:\uFFFF'
  };
  var hiragana_re = /^[ぁ - ん]+$/;
  db.createReadStream(opt)
    .on("data", function (data) {
      var params = data.key.split(":");
      var title = params[1];
      if (!hiragana_re.test(title)) return;
      console.log(" - " + data.value);
    })
}
```

プログラムを実行するには、以下のコマンドを入力します。

```
$ node leveldb-aozora.js
```

実行したところ

では、プログラムを見てみましょう。プログラムの(※1)では、青空文庫のHTMLファイルを読み出して、LevelDBに格納しています。LevelDBはKey-Value型なので、SQLのようにスキーマを定義する必要はありません。プログラムの(※2)の部分ですが、ここでは、キーを「作者:作品名」の形式で、値を本文にしてデータベースに保存します。

ただし、LevelDBの検索では、キーの前方一致検索のみできるので、作品から検索したい場合のことを考慮して、検索用のインデックスキーも一緒に格納しています。それが、プログラム(※3)の部分で、キーを「idx-title:作品名:作者」の形式で保存し、値を「作者:作品名」としています。こうしておけば、実際の本文データも、すぐに取り出せますね。

プログラムの(※4)では、作者から作品一覧を検索しています。利用例として、「夏目漱石」の作品一覧を検索しています。そのために、db.createReadStream()メソッドのオプションで、夏目漱石から始まるキーを抽出するように指定しています。

ここで注意が必要ですが、キーがアルファベットだけのときは、以下のように、書けばいいのですが、日本語のキーだとこれではうまくいきません。

```
var opt = {
  start : "key:",
  end   : "key:\xFF"
};
```

日本語キーでも正しく動くようにするには、Unicode 文字の埋め込み指定である「\uXXXX」形式で終端文字列指定する必要があります。そこで、以下のように指定しています。

```
var opt = {
  start : "夏目漱石:",
  end   : "夏目漱石:\uFFFF"
};
```

プログラムの (※ 5) では、作品名から作品を検索します。ここで、検索オプションを以下のように指定しています。先ほど (※ 4) の部分で、「idx-title:」から始まる検索用のキーを保存したので、これを利用することができます。ところで、ここでは開始文字を「gte」、終了文字を「lte」で指定しています。「gte」は greater than or equal の略で、「start」と同じ意味になります。同様に「lte」も less than or equal の略です。どちらかと言えば「start」と「end」で指定する方がわかりやすいのですが、マニュアルには、今後「gte」と「lte」を使うようにと指示があります。

```
var opt = {
  gte: 'idx-title:' + title,
  lte: 'idx-title:' + title + "\uFFFF"
};
```

そして、プログラムの (※ 6) では、さらに正規表現を使って、作品名がひらがなだけのものを検索しています。作品一覧を全部列挙しておいて、JavaScript でタイトルがひらがなかどうかを判定しているだけなので、それほど特殊なことをしているわけではありません。

この節のまとめ

- この節では、データベースの使い方について紹介しました。
- 組み込みデータベースエンジンだけで、手軽に扱える「SQLite」と「LevelDB」を取り上げました。
- 「青空文庫」のデータを、それぞれのデータベースへ保存してみましたので、実際の使い勝手を比較できました。
- どちらのデータベースを使うにしても、一度データベースに格納すれば、すぐに目的データを取り出すことができて便利です。

06 レポートの自動生成

Web からデータをダウンロードして、データベースに保存しただけでは意味がありません。保存したデータを解析し、レポートとして出力する必要があります。ここでは、さまざまなレポートの自動生成について紹介します。

ここで学ぶポイント

- レポートの生成について
- どんな出力形式を選ぶのか
- PDF形式について
- MS Excel形式について

ツールやライブラリの一覧

- node.js
- ライブラリ「PhantomJS」
- ライブラリ「PDFKit」
- ライブラリ「officegen」
- ApachePOI

レポートを自動的に生成する

Web ダウンロードしたデータを、そのままレポートとして出力することはまれでしょう。たいていの場合、なんらかの分析や集計を行った上で、レポートとして出力するケースが多いと思います。分析や集計の方法は後ほど紹介しますので、ここでは、どんなレポートを出力できるのか、その出力形式や、どんなライブラリを利用できるのかを解説します。

レポートを出力する目的をはっきりさせよう

さて、何のためにレポートを出力するのでしょうか。そもそも、レポートを書く目的は「自分の考えや主張、それらを形にして自分自身と誰かに報告するため」です。自分自身のなかではっきりさせたいことや、誰かに伝えたいことをレポートとしてまとめるのです。

もちろん、完全なレポートを JavaScript のプログラムだけで生成することはできません。プログラムでレポートを自動生成することを考えれば、すでに何かしらの形があるものを整形して出力したり、あるいは、レポートの参考資料を自動生成したりするというのがふさわしいでしょう。

したがって、レポートを自動生成するには、「誰に何を伝えるのか」という部分を意識して選定するのが良いと思います。

出力形式について

さて、ここでは「どんなレポート」を「誰のために出力するのか」の部分にフォーカスして考えてみましょう。

レポートを Web で公開する場合

まず、Web でレポートを公表するという前提があるのであれば、HTML や PDF で出力することになるでしょう。また、Excel や Word 形式でデータが配布されているのも見かけます。これらの形式では、Web ブラウザーからローカルへ一度ダウンロードした上で見てもらうことになるので、一段階、手間がかかりますが、Excel や Word であれば、誰もが持っているアプリを利用して見てもらうことができます。

他にも、特定の目的を持って配布されるデータ形式もあります。

例えば、デザインのサイトで、Adobe Illustrator ファイルが配布されたり、建築などのサイトで、CAD ファイルが配布されたりいった場合です。本書では扱わないものの、最近の Illustrator や CAD ソフトには、マクロ機能がついており、それらを利用して、レポートを自動生成することもできます。ちなみに、最近のアプリについているマクロ言語は、JavaScript を採用しているものが多いので、手軽に使えます。簡単にまとめてみます。

ファイル形式	特徴
HTML 形式	Web ブラウザーで見られる
PDF 形式	Web ブラウザーの中でビューワーが開いて見られる
Excel/Word 形式	ダウンロードしてローカルで見てもらう
専門分野のファイル	特定のアプリ向けだが、専門分野の人にダウンロードして見てもらう

Web で公開するレポート

自分用にレポートを出力する場合

次に自分用にレポートを出力することを考えてみましょう。自分用であれば、それほど、形式や出力品質にこだわる必要はないでしょう。分析や集計の結果が直感的にわかる形式で十分です。ですから、最低限のデータということで、テキストファイルで出力しても問題ないでしょうし、テキストデータに簡単に見出しや段落を付けた、HTML ファイルを生成するのも、ほとんど手間がかからないのでお勧めです。

加えて、自分用であれば、CSV 形式を出力するだけでも十分でしょう。CSV 形式はテキストデータをカンマと改行で区切っただけのものですから、データの作成は簡単です。しかし、Excel などの表計算アプリでデータを読み込むことで、手軽でありながら、リッチなレポートとして見ることができます。CSV 形式と同じように、XML 形式でも出力できるでしょう。多くの Web ブラウザーは、XML 形式のデータを綺麗に色分けし、タグの階層に分けて表示する機能を備えています。

また、HTML を書くのと、それほど手間は変わらないものの、WIKI 記法や Markdown 形式で書き出すことも検討できます。というのは、これらのデータは手軽に HTML や PDF などのファイル形式に変換することができるからです。

フォーマット	特徴
テキスト形式	手間がかからない
HTML 形式	簡単なタグ付けで見栄えもよくなる
CSV 形式	Excel などの表計算アプリで読み込めば見やすい
XML 形式	Web ブラウザーの中には綺麗に表示してくれるものがある
WIKI 記法 /Markdown 形式	手軽に HTML や PDF に出力できる

自分向けのレポート

レポートを印刷する場合

いろいろな分野で電子化が進んでいるというものの、やはり、紙に資料を印刷して出力する必要がある場合も、まだまだ多いものです。その理由としては、役所に紙で提出する必要があったり、上役などには紙で資料を読みたがる人がいたりすることがあるからです。

この場合には、用紙サイズやページ枚数を意識した作りのアプリ向けにデータを作成する必要があります。となると、PDF 形式や、Excel/Word 形式に限られるでしょう。

また、この場合でも、ゼロからデータを作ることは少ないでしょう。テンプレートを用意しておいて、そのテンプレートにデータを埋め込んだり、値を書き込んだりと、ある程度の枠組みがあった方が、品質の高いレポートが作成できます。

フォーマット	特徴
PDF 形式	プリンタに依存しない形式で出力できる
Word/Excel 形式	綺麗に印刷できる

印刷向けのレポート

PDF 形式ファイルの作成

PDF 形式はレポート出力の有力な候補となります。PDF で作成すれば、PC やタブレット、スマートフォンなど、多くの環境でデザインの崩れを気にすることなく閲覧できます。

そもそも、PDF とは、「Portable Document Format」の略で、Adobe 社が開発した電子文書フォーマットです。特定の環境に左右されずに全ての環境でほぼ同様の状態で文章や画像等を閲覧できるという特性を持っています。2008 年 7 月には国際標準化機構によって ISO 32000-1 として標準化されました。Adobe Reader をはじめ、多くの PDF ビューワーが存在します。Web ブラウザーとの親和性がよく、他の Web ページと同じように、ブラウザーの中で表示することもできます。

簡単に PDF を作る方法

自動的に作るわけではありませんが、簡単に PDF を作るには、Microsoft の Excel や Word の PDF 出力機能を利用する方法があります。メニューの「保存」から「PDF 形式で保存」を選ぶことで、PDF 形式で文書を出力することができます。

PhantomJS で HTML を PDF として出力する方法

まず、直接 PDF を作るのではなく、何か元のデータを作っておいて、それを PDF に変換する方法を考えてみます。PhantomJS/CapserJS を利用することで、HTML ファイルを PDF に変換することができます。やり方は、3 章で説明した画面のスクリーンショットを撮る方法と同じなのですが、印刷用の設定を少し加えます。

それでは、プログラムを見てみましょう。今回は、青空文庫から、宮沢賢治の小説「やまなし」のページを PDF に変換してみました。

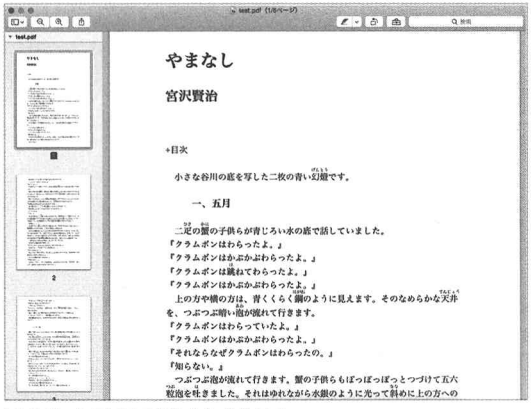
● file: src/ch04/06-output/html2pdf.js

```
// HTML を PDF として出力 for CasperJS

// 宮沢賢治 - やまなし
var url = "http://www.aozora.gr.jp/cards/000081/files/46605_31178.html";
var savepath = "test.pdf";

// CasperJS のオブジェクトを作成
var casper = require('casper').create();
casper.start();
// ページの設定 ----- (※1)
casper.page.paperSize = {
  width: '8.5in',
  height: '11in',
  orientation: "portrait",
  margin: '1cm'
};
casper.open(url);
casper.then(function () {
  casper.capture(savepath);
});
casper.run();
```

プログラムを実行するには、下記のようにコマンドをタイプします。

```
$ casperjs html2pdf.js
```

すると、次のような PDF が生成されます。綺麗にレンダリングされているのがわかるのではないでしょうか。



HTML を PDF で出力したところ

基本的にスクリーンショットなので、画像となっているため、文字の個別選択などはできませんが、さまざまな端末で綺麗に表示できるのがメリットです。

プログラムのポイントは、プログラムの(※1)の部分です。casperのオブジェクトを作成し、casper.start()メソッドを実行した後で、casper.page.paperSizeのプロパティを設定するのです。ページサイズの設定やページの向き、また、用紙のマージンを設定することができます。

また、CasperJSを使うなら、印刷用のCSS設定などを後から差し込むことができます。この場合、スクリーンショットを撮影する前に、以下のようなJavaScriptコードを差し込みます。

```
// CSSを書き換える
casper.then(function () {
  casper.evaluate(function () {
    var els = document.querySelectorAll('h4');
    for (var i = 0; i < els.length; i++) {
      var e = els[i];
      e.style.backgroundColor = "red";
      e.style.color = "white";
    }
  });
});
```

PDFKitを利用して出力する

PDFをゼロから作ることができる「PDFKit」というライブラリもNode.js用に公開されています。このライブラリも、npmを利用してインストールすることができます。

```
$ npm install pdfkit
```

では、まずどんな雰囲気でPDFファイルを作ることができるのか、サンプルを見て学びましょう。

● file: src/ch04/06-output/pdfkit-test.js

```
// PDFKitを使うテスト for Node.js

// モジュールの読み込み
var PDFDocument = require('pdfkit');
var fs = require('fs');

// ドキュメントを作る　(※1)
var doc = new PDFDocument();

// 出力ファイルを設定する　(※2)
doc.pipe(fs.createWriteStream('output.pdf'));

// フォントを埋め込む　(※3)
doc.font('sazanami-gothic.ttf');

// 文字を表示する　(※4)
doc.fontSize(30)
   .text('今日の格言', 90, 100);
doc.fontSize(20)
   .text("求めつづけなさい。そうすれば与えられます。\n\n" +
         "探しつづけなさい。そうすれば見いだせます。\n\n" +
         "たたきつづけなさい。そうすれば開かれます。",
         100,180);
```

```
// 図形を描画する　（※5）
doc.save()
   .moveTo(80, 80)
   .lineTo(250, 80)
   .lineTo(250, 150)
   .lineTo(80, 150)
   .lineTo(80, 80)
   .stroke('#0000FF');

// 改ページを行う　（※6）
doc.addPage();

// 図形を描画する
doc.save()
   .moveTo(100, 150)
   .lineTo(100, 250)
   .lineTo(200, 250)
   .fill('#FF0000');

// 描画を終了する　（※7）
doc.end();
```

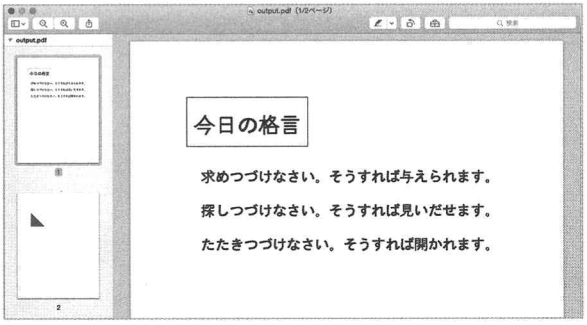
このサンプルでは、「さざなみフォント」というフォントファイルを利用して日本語を表示しています。以下のサイトよりフォントファイルをダウンロードして、プログラムと同じフォルダにコピーしてください（もちろん、TTF フォントであれば、どのフォントでも利用できます）。

```
さざなみフォント
http://sourceforge.jp/projects/efont/
```

プログラムを実行するには、コマンドラインより、以下を実行します。

```
$ node pdfkit-test.js
```

すると、「output.pdf」という PDF ファイルを生成します。



PDF を自動生成したところ *

プログラムを見ると、PDFKit を使うと、手軽に PDF を作成できることに驚いたかもしれません。先ほどの、PhantomJS を使う方法では、作成した PDF の中にある文字を選択することはできませんでしたが、PDFKit を使ったこの方法では、文字を選択してクリップボードにコピーすることもできます。

---

* もしも、プログラムを実行したときに、日本語が「・」に化けているなら、それは、正しいフォントが見つからなかったということです。正しいフォントのパスを指定する必要があります。

では、実際のプログラムを見ていきましょう。必要なモジュールを読み込んだ後で行う事は三つだけです。

まず、プログラムの(※1)にあるように、new PDFDocument() を呼び出します。そして、(※2)のように出力ファイルを指定します。最後に、(※3)のように、日本語フォントを指定して埋め込みます。これですべての準備が整います。

文字を表示させるには、プログラムの(※4)にあるように、doc.fontSize() メソッドでフォントサイズを指定して、doc.text() メソッドでテキストと表示位置を指定するだけです。左上座標が (0, 0) となっており、右下に行くほど数値が増えていきます。一行だけでなく、改行を含む数行を一緒に表示させることができます。テキストの長さが長い場合は、自動的に改行してくれるので、安心して長文を指定して表示させることができます。

プログラム(※5)にあるように、図形も表示できます。この場合、doc.save() メソッドに続けて、moveTo()、lineTo() メソッドを使って座標をして最後に、stroke() メソッドで任意の色の線を引くことができます。各メソッドの意味は以下の通りです。

メソッド	意味
moveTo(x, y)	図形の始点を指定する
lineTo(x, y)	前回の座標から (x, y) へ線を引く
stroke(color)	それまでに指定した座標に線を引く
fill(color)	それまでに指定した座標を塗りつぶす

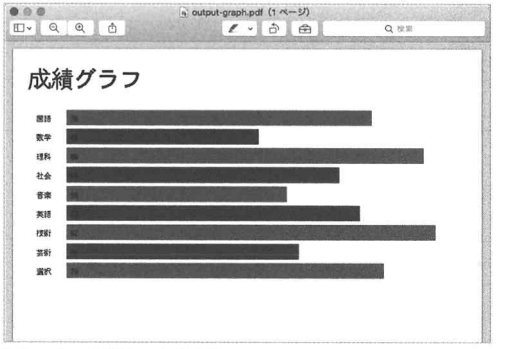
PDFKit の図形描画メソッド一覧

PDF で改ページを行う場合は、(※6)にあるように、doc.addPage() メソッドを使います。

また、重要な点として、(※7)のように、最後に、doc.end() メソッドを呼び出します。

PDFKit でグラフを描く

先ほどは、文字を描いたり簡単な図形を描いたりするだけでした。今度は、データに基づいて PDF に簡単な棒グラフを描画してみましょう。以下のようなグラフを作ってみます。



PDFKit でグラフを描画した

グラフを描くプログラムは、以下のようになります。

● file: src/ch04/06-output/pdfkit-graph.js

```
// PDFKit でグラフを描画 for Node.js

  中略
```

```
// ドキュメントを作る　（※1）
var doc = new PDFDocument();
var page_w = doc.page.width;
var page_h = doc.page.height;

// 出力ファイルを設定する　（※2）
doc.pipe(fs.createWriteStream('output-graph.pdf'));

// フォントを埋め込む　（※3）
doc.font('sazanami-gothic.ttf');

// タイトルを表示する　（※4）
doc.fontSize(30)
   .text('成績グラフ', 20, 20);

// グラフを描画する　（※5）
var margin = 20;
var g_w = page_w - margin * 2 - 50;
var g_x = margin + 50;
var y = 80;
var wpx = g_w / 100;
for (var i = 0; i < data.length; i++) {
  var value = data[i].value;
  var label = data[i].label;
  doc.save()
     .rect(g_x, y, wpx * value, 20)
     .fill( (i % 2) ? 'blue':'red');
  doc.fontSize(10)
     .fillColor("black")
     .text(label, 30, y + 5)
     .text(value, g_x + 5, y + 5);
  y += 20 + 5;
}

// 描画を終了する　（※6）
doc.end();
```

プログラムを実行するには、コマンドラインから次のコマンドを入力します。

```
$ node pdfkit-graph.js
```

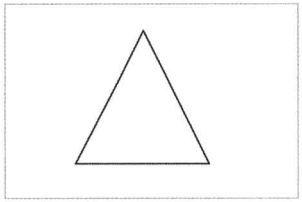
では、プログラムのポイントを確認してみましょう。プログラムの（※1）では、PDFDocument のオブジェクトを作成します。ページのサイズは、PDFDocument.page オブジェクトに設定されています。page.width と page.height を調べることでわかります。

プログラムの（※2）では、出力先ファイルを指定し、（※3）ではフォントの埋め込みを行います。（※4）では、タイトルを描画し、（※5）でグラフを描画します。

ここでは、doc.rect() メソッドを利用して各値の棒グラフを描画しています。図形を描画するのに便利なメソッドが用意されています。

メソッド名	説明
rect(x, y, width, height)	長方形を描画
roundRect(x, y, width, height, conerRadius)	角丸長方形を描画
ellipse(cx, cy, radiusX, radiusY)	楕円形を描画
circle(cx, cy, radius)	正円を描画
polygon(points...)	多角形を描画
path(pathData)	SVG の path 指定で描画

多角形を描画する polygon() メソッドは以下のように利用します。



多角形を描画した

```
doc.polygon(
  [150, 50], [100, 150], [200, 150]);
doc.stroke();
```

PDFKit のより詳しい情報

この他にも、PDFKit では、図形にグラデーションをかけて描画したり、直接画像データを埋め込んだりと、他にもさまざまな機能が利用できます。より詳しい情報は、PDFKit の Web サイトに API がまとめられています。

```
PDFKit
http://pdfkit.org
```

Excel 形式ファイルの作成

多くの人は、Microsoft Excel ファイルでの出力を望んでいます。これは、普段からオフィスで利用する機会が多いからでしょう。使い慣れたファイル形式であれば、安心して使えるというのは大きいでしょう。というのも、Excel であれば、そこからさらにデータの集計を行ったり、印刷したりもできます。

Node.js + Officegen を使う方法

しかし、Excel 形式のファイルは、基本的に Excel または Excel 互換オフィスツールを利用しないと作成できません。Excel 形式はそれほど単純なデータフォーマットではありませんが、Node.js 用に、ライブラリ「officegen」が用意されています。これを利用することで、Excel がインストールされていない環境でも、Excel ファイルを作成することができます。また、Word など Office のファイル形式も出力することができます。

それでは officegen をインストールしましょう。npm を利用してインストールできます。

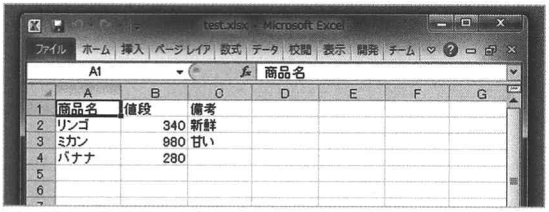
```
$ npm install officegen
```

まずは、新規シートにデータを書き込んでみます。見たとおりのプログラムです。それほど難しい点はないでしょう。

● file: src/ch04/06-output/officegen-test.js

```
// Excel ファイルを生成するテスト for Node.js

中略

// 新規シートを作成
var sheet = xlsx.makeNewSheet();
sheet.name = "test";

// 直接データを書き換え
sheet.data[0] = ["商品名", "値段", "備考"];
sheet.data[1] = ["リンゴ", 340];
sheet.data[2] = ["ミカン", 980];
sheet.data[3] = ["バナナ", 280];

// セル名を指定して書き換え
sheet.setCell('C2', '新鮮');
sheet.setCell('C3', '甘い');

// ファイルを書き出す
var strm = fs.createWriteStream('test.xlsx');
xlsx.generate(strm);
```

プログラムを実行すると、次のようになります。



プログラムを実行したところ

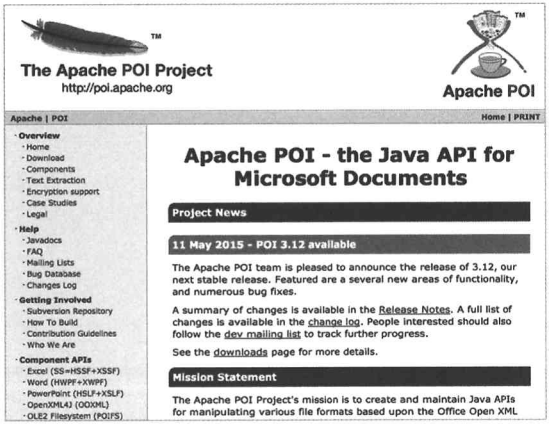
Rhino と Apache POI を使う方法

このように、比較的簡単に使える「officegen」です。しかし、それほど複雑なことができるわけではありません。新規シートを作成し、そこに値を書き込むことができるくらいのものです。

もう少し、凝ったことを JavaScript で実現したいという場合は、Java のライブラリの力を借りると良いでしょう。ここでは、Java のライブラリである「Apache POI」を使ってみましょう。Rhino/JavaScript では、Java のライブラリを利用できるのが良いところです。Apache POI は、Java のライブラリで、Microsoft の Excel や Word のファイルを Java から操作するためのライブラリです。この Apache POI を利用することで、Excel や Word をインストールしていない PC でも、Excel/Word のファイルを読み書きすることができるようになります。

Apache POI は以下の URL からダウンロードすることができます。

```
Apache POI
http://poi.apache.org/
```



Apache POI のページ

メニューの「download」から POI のアーカイブをダウンロードします。そして、Rhino の本体 (js.jar) と同じフォルダに、POI で利用するライブラリ (*.jar) の一式をコピーします。ここでは、原稿執筆時点の最新の安定版「3.12-20150511」を利用してみました。

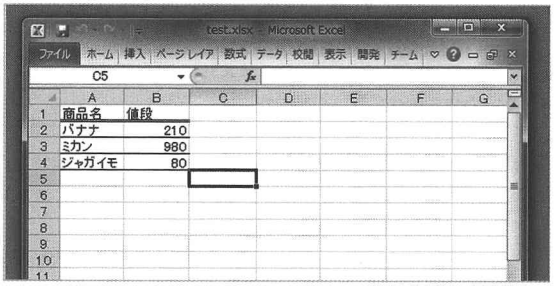
それでは、使い方を見てみましょう。

● file: src/ch04/06-output/poi-test.js

```
// Excel ファイルを書き出す for Rhino + Apache POI

// 今回 Excel に書き込むデータ
var list = [
  ["商品名", "値段"],
  ["バナナ", 210],
  ["ミカン", 980],
  ["ジャガイモ", 80]
];

// Java のクラスを宣言
var XSSFWorkbook = org.apache.poi.xssf.usermodel.XSSFWorkbook;
var XSSFCellStyle = org.apache.poi.xssf.usermodel.XSSFCellStyle;
var FileOutputStream = java.io.FileOutputStream;

// ワークブックを作成
var wb = new XSSFWorkbook();
// ワークシートを作成
var sheet = wb.createSheet("sheet-test");
// セルスタイルの作成
var style_u = wb.createCellStyle();
style_u.setBorderBottom(XSSFCellStyle.BORDER_THIN);
var style_head = wb.createCellStyle();
style_head.setBorderBottom(XSSFCellStyle.BORDER_DOUBLE);

// セルに値を設定
for (var i = 0; i < list.length; i++) {
  var row = sheet.createRow(i);
  var c1 = row.createCell(0);
  var c2 = row.createCell(1);
  c1.setCellValue(list[i][0]);
  c2.setCellValue(list[i][1]);
  var style = (i == 0) ? style_head : style_u;
  c1.setCellStyle(style);
```

```
  c2.setCellStyle(style);
}

// 値を出力
var out = new FileOutputStream("test.xlsx");
wb.write(out);
```

このプログラムを実行すると、以下のようにセルへ値を書き込んだ上で、セルに罫線を書き込むことができます。



セルにデータを書き込み罫線を引く

このプログラムを実行する方法なのですが、Apache POI はさまざまなライブラリに依存しており、非常に複雑な構造となっています。そこで、JAR ファイルを一つのフォルダにまとめた上で、それを利用するように、Rhino 実行時のクラスパスに JAR ファイルを指定するようにします。

例えば、Windows であれば、JAR ファイルを lib フォルダにまとめた上で、以下のようなコマンドを実行します。いやはや複雑ですね。とは言っても、これは、POI のダウンロードファイルに含まれていた JAR ファイルをすべてクラスパスに列挙しただけです。

```
java ^
  -cp ./lib/js.jar;./lib/commons-codec-1.9.jar;^
./lib/commons-logging-1.1.3.jar;./lib/junit-4.12.jar;^
./lib/log4j-1.2.17.jar;./lib/poi-3.12-20150511.jar;^
./lib/poi-examples-3.12-20150511.jar;^
./lib/poi-excelant-3.12-20150511.jar;^
./lib/poi-ooxml-3.12-20150511.jar;^
./lib/poi-ooxml-schemas-3.12-20150511.jar;^
./lib/poi-scratchpad-3.12-20150511.jar;^
./lib/xmlbeans-2.6.0.jar ^
  org.mozilla.javascript.tools.shell.Main poi-test.js %*
pause
```

Mac OS X や Linux でも同様ですが、パスの区切り記号が「;」ではなく「:」になります。

Apache POI は非常に高機能なので、ここでは概要だけ紹介するにとどめます。

その他の方法

また、ここでは扱いませんが、Windows に Excel がインストールされている環境であれば、WSH/JScript を利用して、COM 経由で Excel を遠隔操作するという方法も利用できます。この方法だと、Windows に Excel をインストールしているという前提でしか利用できませんが、Excel でできることは大抵できるのが魅力です。

Web API で取得した値を Excel に書き込む

それでは、ここでは、百人一種を取得する Web API である、「百人一首 API」を利用して書き出すプログラムを紹介します。ここでは、手軽に利用できる、Node.js+officegen の組み合わせで作ってみます。

```
百人一種 API
http://api.aoikujira.com/hyakunin/
```

この API の使い方は単純で、次の URL にアクセスすると、百人一首を JSON 形式で取得できるというものです。

```
http://api.aoikujira.com/hyakunin/get.php?fmt=json
```

ここで作るプログラムは、上記 API より、JSON 形式で百人一首の一覧を取得し、Excel ファイルに書き込むというものとなっています。それでは、さっそく作ったプログラムを見てましょう。

● file: src/ch04/06-output/hyakunin-excel.js

```
// 百人一首を Excel に書き出す for Node.js

var API = "http://api.aoikujira.com/hyakunin/get.php?fmt=json";

// モジュールを取り込む
var fs = require('fs');
var officegen = require('officegen');
var xlsx = officegen('xlsx');
var request = require('request');

// 百人一首をダウンロード
request(API, function(err, res, body){
  if (err) throw err;
  var list = JSON.parse(body);
  exportToExcel(list);
  console.log(list);
});

function exportToExcel(list) {
  // 新規シートを作成
  var sheet = xlsx.makeNewSheet();
  sheet.name = "百人一首";

  // 直接データを書き換え
  sheet.data[0] = [
    "番号","上の句","下の句"
  ];
  for (var i = 0; i < list.length; i++) {
    var r = list[i];
    sheet.data[i + 1] = [r.no, r.kami, r.simo];
  }
  // ファイルを書き出す
  var strm = fs.createWriteStream('hyakunin.xlsx');
  xlsx.generate(strm);
  console.log("ok");
}
```

プログラムを実行するには、以下のコマンドを実行します。

```
$ node hyakunin-excel.js
```

プログラムを実行したところ

少し、プログラムを見てみましょう。このプログラムでは、request() メソッドを利用して、Web 上のデータを取得します。取得したデータは、JSON 形式となっているので、JSON.parse() メソッドを利用して、JavaScript のオブジェクトに変換します。

その上で、Excel にデータを書き込みます。ここでは、作成したシートを表す配列変数 sheet.data に複数のデータを書き込んでいます。

Node.js と officegen を使うと手軽に Excel 形式でデータを出力できます。とは言え、あまり凝ったことができないので、もう少し凝ったことをしたい場合は、Java のライブラリを利用して、Rhino と Apache POI を利用することもできます。レポート出力において何が必要になるのか考えて、どの技術を使うのか選ぶと良いでしょう。

この節のまとめ

- **レポートの出力形式を決める時には、誰が何の目的で見るのかを明確にすると、最適な形式がわかります。**
- **PDF 出力には、PhantomJS や PDFKit などのライブラリを利用できます。**
- **Office 形式のデータの出力には、Officegen や Apache POI などのライブラリを利用できます。**

第5章

形態素解析で日本語を扱う

ここでは、実際に日本語の文章を扱う方法を紹介します。特に、形態素解析を利用して、日本語の文章を単語に区切る方法、また形態素解析を利用したプログラムを紹介します。日本語解析の基礎を確認していきましょう。

01

形態素解析について

日本語で書かれた情報を、何かしらの形で活用しよう思った時には、外すことのできない技術として「形態素解析」があります。ここでは、形態素解析とは何か、そして、何に活用できるのかを解説し、さらに具体的な利用方法を紹介します。

ここで学ぶポイント	ツールやライブラリの一覧
●形態素解析について	●MeCab

形態素解析とは何か？

「形態素解析（けいたいそかいせき、Morphological Analysis）」とは、自然言語処理の基礎技術のひとつです。簡単に言えば「自然言語で書かれた文章を、単語ごとに分割し、それぞれの品詞を判別する作業」のことです。

扱う言語が英語であれば、文章の中で単語と単語がスペースで明確に区切られているので、文章を単語に分けるのは容易です。これに対して、日本語は単語と単語の区切りがなく、単語ごとに分割する作業は骨の折れる作業となります。そこで、形態素解析という作業が必要になります。

以下に挙げるのは「MeCab」というツールを利用して文章を形態素解析した例です。

元の文章：

```
仕事と結婚は人生を左右する大切な事柄です。
```

解析結果：

```
仕事　名詞,サ変接続,*,*,*,*,仕事,シゴト,シゴト
と　助詞,並立助詞,*,*,*,*,と,ト,ト
結婚　名詞,サ変接続,*,*,*,*,結婚,ケッコン,ケッコン
は　助詞,係助詞,*,*,*,*,は,ハ,ワ
人生　名詞,一般,*,*,*,*,人生,ジンセイ,ジンセイ
を　助詞,格助詞,一般,*,*,*,を,ヲ,ヲ
左右　名詞,サ変接続,*,*,*,*,左右,サユウ,サユー
する　動詞,自立,*,*,サ変・スル,基本形,する,スル,スル
大切　名詞,形容動詞語幹,*,*,*,*,大切,タイセツ,タイセツ
な　助動詞,*,*,*,特殊・ダ,体言接続,だ,ナ,ナ
事柄　名詞,一般,*,*,*,*,事柄,コトガラ,コトガラ
です　助動詞,*,*,*,特殊・デス,基本形,です,デス,デス
。　記号,句点,*,*,*,*,。,。,。
```

このように、文章を最小単位の形態素に分割します。

この作業はどのように行われるのでしょうか？　必要になるのは文法ルールと、単語辞書です。日本語の形態素解析では、最初に文から切り出した単語が属する品詞を辞書を用いて調べていきます。そして、品詞の並びから文法的に正しい並びを探すという手法で行われます。これを実現するには、一定の規則を用いて解析する方法と、統計モデルを利用して解析する方法があります。

規則による分割方法では、文章を一文字ずつ辞書や文法ルールに合致するかどうかを調べていきます。これに対して、統計モデルを利用する方法では、大量の日本語文データから、出現頻度を調べて、頻度の高い方を選択します。実際にすべての単語と単語の組み合わせを調べるのは無理なので、品詞と品詞の組み合わせを元にする方法があります。現在主流なのは後者で、統計データを元にした解析を行います。

何に活用できるのか？

さて、形態素解析を何に活用できるのでしょうか。具体的な活用例としては、Google などの検索エンジンや音声認識、かな漢字変換、テキストマイニング、ニュース記事のレコメンド機能、重要メールの判定などといった文章の自動分類など、さまざまな場面で利用されています。

形態素解析を利用する方法

本書では形態素解析の具体的な解析手法を詳しく解説はしませんが、どんな形態素解析ツールがあるかを紹介し、実際に形態素解析を行ってみます。

それでは、まず日本語の形態素解析ツールをいくつか紹介します。以下は、フリーで入手可能な形態素解析エンジンの代表的なものです。

```
ChaSen（茶筌）
http://chasen-legacy.osdn.jp/
バージョン 2.4.5（2012-06-26）
```

```
MeCab（和布蕪）
http://taku910.github.io/mecab/
バージョン 0.996（2013-02-18）
```

```
KyTea
http://www.phontron.com/kytea/index-ja.html
バージョン 0.4.7（2014-10-18）
```

```
kuromoji - Java 形態素解析器
http://atilika.org/
https://github.com/atilika/kuromoji
```

```
Igo - Java 形態素解析器
http://igo.osdn.jp/
バージョン 0.4.3（2011-06-17）
```

この原稿を執筆している時点では、MeCab が iPhone のかな漢字変換エンジンにも採用されていて、最も安定して多くの環境で動作する形態素解析器という印象があります。そこで、本書ではこの MeCab を利用してプログラムを作っていきます。

MeCab のインストール

MeCab は誰でも自由に利用できるオープンソースライセンスのソフトウェアで、GPL/LGPL/BSD ライセンスの元で配布されています。

CentOS6 にインストールする場合は、ソースコードからインストールする必要があります。まずは、以下のコマンドを実行して、MeCab 本体をインストールします。

```
$ curl -L -o mecab-0.996.tar.gz "https://drive.google.com/uc?id=0B4y35FiV1wh7c
ENtOXlicTFaRUE&export=download"
$ tar zxf mecab-0.996.tar.gz
$ (cd mecab-0.996; ./configure --enable-utf8-only && make && sudo make
install)
```

続いて、辞書データをインストールしましょう。以下のコマンドでインストールします。

```
$ wget http://mecab.googlecode.com/files/mecab-ipadic-2.7.0-20070801.tar.gz
$ tar xvf mecab-ipadic-2.7.0-20070801.tar.gz
$ cd mecab-ipadic-2.7.0-20070801/
$ ./configure --with-charset=utf8
$ make
$ sudo make install
```

Mac OS X へ MeCab のインストール

Mac OS X では、Homebrew を利用してインストールします。このとき、MeCab 本体と辞書データをインストールします。コマンドを 2 回入力するだけで済んでしまうので、Homebrew を使う方法はかなり便利です。

```
$ brew install mecab
$ brew install mecab-ipadic
```

Windows へ MeCab のインストール

MeCab の Windows 版ではバイナリパッケージが用意されています。Web サイトにアクセスし、メニューの「ダウンロード > Binary package for MS-Windows」からダウンロードします。ダウンロードした実行ファイル (mecab-x.xx.exe。x.xx はバージョン) がインストーラーとなっているので、これを実行して何度か「次へ」ボタンをクリックすると設定が完了します。パスも自動的に通してくれます。ほとんど自動的にインストールできますが、セットアップ中に一度だけ文字コードを尋ねられるので「Shift_JIS」を選んでおきましょう。

MeCab の Web サイト
http://taku910.github.io/mecab/

MeCab の実行テスト

インストールすると、コマンドラインから MeCab を試すことができます。以下のように「mecab」コマンドを実行し、その後、日本語の文章を入力します。

```
>mecab
すももももももものうち
すもも　名詞 , 一般 ,*,*,*,*, すもも , スモモ , スモモ
も　　助詞 , 係助詞 ,*,*,*,*, も , モ , モ
もも　　名詞 , 一般 ,*,*,*,*, もも , モモ , モモ
も　　助詞 , 係助詞 ,*,*,*,*, も , モ , モ
もも　　名詞 , 一般 ,*,*,*,*, もも , モモ , モモ
の　　助詞 , 連体化 ,*,*,*,*, の , ノ , ノ
うち　　名詞 , 非自立 , 副詞可能 ,*,*,*, うち , ウチ , ウチ
EOS
```

MeCab を利用したところ

この節のまとめ

- この節では形態素解析について簡単に紹介し、MeCab をインストールしました。
- 簡単な実行例を紹介したので形態素解析の動作についても理解できたと思います。

02 文章にフリガナを振ろう

前節では形態素解析について紹介しました。この節では、Node.js を利用して実際に MeCab を利用するプログラムを作ります。形態素解析の利用例として文章にフリガナを振るプログラムを作ってみます。

ここで学ぶポイント

- MeCabをNode.jsから使う
- モジュールの作り方

ツールやライブラリの一覧

- MeCab
- Node.js

Node.js から MeCab を使う方法

はじめに、Node.js から MeCab を利用する方法を紹介しましょう。コマンドラインから MeCab を呼び出し、結果を取り込むという方法で行います。

簡単な手順ですが、Windows 環境コマンドラインから MeCab を実行すると、文字コードが Shift_JIS であるために、Unicode が基本となる Node.js で利用するには文字コードの変換が必要となります（MeCab のインストール時に Shift_JIS を選択していることを前提にしています）。

ここでは、Iconv-Lite モジュールを利用して、Shift_JIS を UTF-8 に変換するようにしています。npm を使ってインストールしておきましょう。もちろん、Mecab をインストールしていることが前提です。

```
$ npm install iconv-lite
```

● file: src/ch05/02-mecab/mecab-test.js

```
// MeCab を Node.js から使う

// モジュールの取り込み
var exec = require('child_process').exec;
var iconv = require('iconv-lite');
var fs = require('fs');
var platform = require('os').platform(); // OS 判定 ---- (※1)

// 形態素解析するテキスト
var srcText = "探しつづけなさい。そうすれば見いだせます。\n";

// 一時ファイル
var TMP_FILE = __dirname + '/__mecab_tmpfile';
// MeCab のコマンドライン
var MECAB = 'mecab';
var ENCODING = (platform.substr(0,3) == 'win')
             ? 'SHIFT_JIS' : 'UTF-8';
```

```
// 形態素解析を実行する関数 ----- (※2)
function parse(text, callback) {
  // 変換元テキストを一時ファイルに保存 ----- (※3)
  if (ENCODING != 'UTF-8') {
    var buf = iconv.encode(text, ENCODING);
    fs.writeFileSync(TMP_FILE, buf, "binary");
  } else {
    fs.writeFileSync(TMP_FILE, text, "UTF-8");
  }
  // コマンドを組み立てる
  var cmd = [
    MECAB,
    '"' + TMP_FILE + '"'
  ].join(" ");
  // コマンドを実行 -------- (※4)
  var opt = { encoding: 'UTF-8' };
  if (ENCODING != 'UTF-8') opt.encoding = 'binary';
  exec(cmd, opt,
    function (err, stdout, stderr) {
      if (err) return callback(err);
      var inp;
      // 結果出力ファイルを元に戻す ---- (※5)
後略
```

プログラムを実行するには、コマンドラインで以下のように入力します。

```
$ node mecab-test.js
```

プログラムを実行すると、以下のような結果が表示されます。

```
探し：動詞：サガシ
つづけ：動詞：ツヅケ
なさい：動詞：ナサイ
。：記号：。
そう：副詞：ソウ
すれ：動詞：スレ
ば：助詞：バ
見いだせ：動詞：ミイダセ
ます：助動詞：マス
。：記号：。
```

Windows 以外では、MeCab は UTF-8 で動作します。もちろん、Windows でもインストール時の辞書選択で、UTF-8 を選択すれば良いのですが、そうすると、コマンドラインで文字化けしてしまうので悩ましいところです。ここでは、Windows では Shift_JIS の文字コードを使うと仮定しています。

そのため、このプログラムでは OS で処理を分岐することにしました。プログラムの (※1) のように、os モジュールの platform() メソッドを利用すると、OS の種類を調べることができます。改めてプログラムを切り出してみましょう。他には、process.platform でも同じ結果が得られます。

```
// 方法1
var platform = require('os').platform();
console.log(platform);
// 方法2
console.log(process.platform);
```

Windows では、"win32" または "win64" と表示されます。Mac OS X では "darwin" と表示されます。他に、"freebsd"、"linux"、"sunos" などがあります。

プログラムの (※ 2) の部分、parse() 関数が形態素解析を行う関数となっています。MeCab をコマンドラインから実行して、その結果を受け取るというプログラムです。

ここで、ちょっと複雑なのは、Windows の場合、文字コードを Shift_JIS に変換する必要があるという点でしょう。プログラムの (※ 3) の部分で、テキストを一時ファイルに保存しますが、このときに、iconv-lite モジュールを利用して、文字コードの変換を行います。

プログラムの (※ 4) では、exec() メソッドを利用して、外部コマンドを実行します。このとき、mecab コマンドを実行するために、以下のような文字列を組み立てます。ここで入力ファイル名はカレントディレクトリに適当な名前の一時ファイルを作成します。これを実行すると、標準出力に形態素解析した結果が返ります。

```
$ mecab "( 入力テキストのファイル名 )"
```

プログラムの (※ 5) の部分ですが、Windows で動かす場合には、exec() メソッドでコマンドを実行した戻り値が Shift_JIS で戻ってくるため、UTF-8 への変換が必要となります。特に、Node.js の exec() メソッドは、標準出力が UTF-8 であることを想定しているので、バイナリデータでデータを受け取り、後から UTF-8 に変換するという作業が必要になります。そのために、exec() コマンドを実行する際のオプションに {encoding:"binary"} を指定しておきます。無事に、MeCab から結果が返されたなら、データをタブやカンマで区切って、二次元配列にして返します。

MeCab 利用上の注意

意外な盲点として、MeCab では改行を文章の最後と見なすのですが、そのため、末尾に改行がない文字列を与えると「input-buffer overflow」というエラーが出てしまいます。この点に注意しましょう。

プログラムを整理して形態素解析モジュールを作ろう

実は、原稿執筆時点でも、MeCab を利用するためのモジュールがいくつか公開されていたのですが、どれも、Windows/Mac OS X のどちらかでしか動かないか、対応していた Node.js のバージョンが古くて動かないというものでした。そこで、今回、両方で動くように考慮したプログラムを作ることになりました。

せっかくなので、先ほど作ったプログラムを改良して、MeCab を簡単に利用できるモジュールを作ってみましょう。Node.js でモジュールを作るには、module.exports オブジェクトに追加したいメソッドを記述するだけです。例えば、以下は、一番簡単な足し算を行うだけの「tasizan」モジュールです。このプログラムを「tasizan.js」という名前で保存します。

● file: src/ch05/02-mecab/tasizan.js

```
// 足し算の機能を持ったモジュール

module.exports = function (a, b) {
  return a + b;
};
```

そして、このモジュールを使うプログラムは、以下のようになります。ただし、npm を利用してモジュールをインストールしたわけではないので、モジュールを指定するときに、カレントディレクトリを表す「./」に続いてファイル名を指定します。

● file: src/ch05/05-mecab/tasizan-test.js

```
// tasizan モジュールの利用例

// モジュールの取り込み
var tasizan = require('./tasizan.js');

// モジュールの機能を利用する
console.log(tasizan(4, 5));
console.log(tasizan(1, 3));
```

プログラムを実行してみましょう。思い通りの結果が表示されました。

```
$ node tasizan-test.js
9
4
```

もし、複数のメソッドをサポートしたいときには、以下のようにします。二つの数を足す tasizan2() メソッドと、三つの数を足すことができる tasizan3 メソッドを定義してみました。

● file: src/ch05/02-mecab/tasizan2.js

```
// 複数の足し算の機能を持ったモジュール

module.exports.tasizan2 = function (a, b) {
  return a + b;
};

module.exports.tasizan3 = function (a, b, c) {
  return a + b + c;
};
```

● file: src/ch05/02-mecab/tasizan2-test.js

```
// モジュールを使うテスト

var tasizan2 = require('./tasizan2.js').tasizan2;
var tasizan3 = require('./tasizan2.js').tasizan3;

console.log(tasizan2(3, 5));
console.log(tasizan3(3, 5, 7));
```

どうでしょうか。思ったよりも、簡単にモジュールが作成できたのではないでしょうか。

MeCab モジュールを作成しよう

モジュール作成の要領が掴めました。では、MeCab を利用するモジュールを作ってみましょう。

● file: src/ch05/02-mecab/mecab-mod.js

```
// MeCab を利用するためのモジュール
module.exports = function () {

中略

  // 形態素解析を実行する関数
  this.parse = function (text, callback) {
    var encoding = this.ENCODING;
    text += "\n";
    // 変換元テキストを一時ファイルに保存
    if (encoding != 'UTF-8') {
      var buf = iconv.encode(text, encoding);
      fs.writeFileSync(this.TMP_FILE, buf, "binary");
    } else {
      fs.writeFileSync(this.TMP_FILE, text, "UTF-8");
    }
    // コマンドを組み立てる
    var cmd = [
      this.MECAB,
      '"' + this.TMP_FILE + '"'
    ].join(" ");
    // コマンドを実行
    var opt = { encoding: 'UTF-8' };
    if (encoding != 'UTF-8') opt.encoding = 'binary';
    exec(cmd, opt,
      function (err, stdout, stderr) {
        if (err) return callback(err);
        var inp;
        // 結果出力ファイルを元に戻す
        if (encoding != 'UTF-8') {
          iconv.skipDecodeWarning = true;
          inp = iconv.decode(stdout, encoding);
        } else {
          inp = stdout;
        }
        // 結果をパースする
        inp = inp.replace(/\r/g, "");
        inp = inp.replace(/\s+$/, "");
        var lines = inp.split("\n");
        var res = lines.map(function(line) {
          return line.replace('\t', ',').split(',');
        });
        callback(err, res);
    });
  };
};
```

このモジュールを利用するプログラム例は、以下のようになります。

● file: src/ch05/02-mecab/mecab-mod-test.js

```
// mecab-mod.js のテストプログラム

var Mecab = require('./mecab-mod.js');
var mecab = new Mecab();

var text = "すももももももものうち";
mecab.parse(text, function(err, items) {
  for (var i in items) {
    var k = items[i];
    if (k == "EOS") continue;
    console.log(k[0] + ":" + k[1]);
  }
});
```

コマンドラインから以下のようにしてプログラムを実行してみましょう。

```
$ node mecab-mod-test.js
すもも:名詞
も:助詞
もも:名詞
も:助詞
もも:名詞
の:助詞
うち:名詞
```

上記のように、単語とその品詞が画面に表示されたでしょうか。

文章にフリガナを振るプログラム

それでは、先ほど作成したMeCabのモジュールを利用して、文章にフリガナを振るプログラムを作ってみます。形態素解析をした時点で、フリガナの情報が取れるので、後は、フリガナをどう見せるかという部分になります。基本的にひらがなで書かれている句や句読点などの記号にフリガナは不要ですから、それら不要な部分を抜く処理が必要になるでしょう。

また、せっかくフリガナを振るプログラムですから、少し汎用的に作ってみましょう。引数にテキストファイルを指定すると、フリガナを振って出力するようにしてみます。

● file: src/ch05/02-mecab/furigana.js

```
// フリガナを振る for Node.js
// モジュールの取り込み
var fs = require('fs');
var Mecab = require('./mecab-mod.js');
var mecab = new Mecab();

// コマンドラインを調べる ---- (※1)
var args = process.argv;
args.shift(); // node を除去
args.shift(); // スクリプト名を除去

// 引数がなければプログラムの使い方を表示する --- (※2)
if (args.length <= 0) {
  console.log("[USAGE] furigana.js 入力テキスト");
  process.exit();
```

```
}

// 入力ファイルを読み込む --- (※3)
var inputfile = args.shift();
var txt = fs.readFileSync(inputfile, "utf-8");

// 形態素解析する --- (※4)
mecab.parse(txt, function (err, items) {
  var res = "";
  for (var i in items) {
    var k = items[i];
    var word = k[0];
    var kana = k[8];
    if (k == "EOS") continue;
    // フリガナが必要なときを判定 --- (※5)
    if (word == kana || isHiragana(word)) {
      res += word;
    } else {
      res += word + '(' + kana + ')';
    }
  }
  console.log(res);
});

// ひらがな判定
function isHiragana(s) {
  return (s.match(/^[あ-ん]+$/));
}
```

プログラムをテストしてみましょう。まず以下のようなテキストファイル「test.txt」を用意します。ファイルは、UTF-8 で保存してください。

● file: src/ch05/02-mecab/test.txt

```
自分が裁かれないために、人を裁くのをやめなさい。
あなた方が裁いているその裁きであなた方も裁かれることになるからです。
```

ではプログラムを実行します。以下のコマンドを入力しましょう。すると、結果がコンソールに出力されます。

```
$ node furigana.js test.txt
自分(ジブン)が裁か(サバカ)れないために、人(ヒト)を裁く(サバク)のをやめなさい。あなた方(カタ)
が裁い(サバイ)ているその裁き(サバキ)であなた方(カタ)も裁か(サバカ)れることになるからです。
```

結果を見てみると大筋では合っているものの、「あなた方(ガタ)」と出て欲しいところが「あなた方(カタ)」となってしまいました。このように必ずしも形態素解析の結果が完璧になるわけではありませんが、大体のところで合っています。

では、プログラムを確認していましょう。プログラムの(※1)の部分では、コマンドラインを調べています。例えば「node furigana.js test.txt」とコマンドを実行した場合、process.argv の内容は次のようになります。つまり、要素 [0] が node に、要素 [1] がスクリプトのフルパスに、要素 [2] 以降がコマンドライン引数となります。

```
// process.argv
[
  "node",
  __dirname + "/furigana.js", // フルパス
  "test.txt"
]
```

そして、プログラムの(※2)の部分では、コマンドライン引数が特に指定されていないときに、プログラムの使い方を表示します。(※3)では、入力ファイルを読み込んで、(※4)の部分でMeCabを実行します。
(※5)の部分では、フリガナが必要かどうかを簡単に判定して、それに応じて文章のつなぎを変更しています。

フリガナ振りプログラムの改良のアイデア

ところで、今回のプログラムで、フリガナの付け方が、若干不自然なので、もう少し手を加えて自然なフリガナにできたらと思います。現在は「裁く(サバク)」と送り仮名にもカナが表示されてしまっています。そこで、漢字の部分だけを検出して、うまく「裁(サバ)く」とフリガナを見せられると良いですね。また、HTML形式で出力することを考えると、<ruby>タグがあるのでルビを振ることもできます。腕試しにプログラムの改良に挑戦してみると良いでしょう。

この節のまとめ

- この節では、プログラムからMeCabを使う方法を紹介しました。
- モジュール化したので、別のプログラムからも使いやすくなりました。
- 以降の節では、このモジュールを使っていろいろなプログラムに応用してみます。

03 マルコフ連鎖で文章を要約する

形態素解析の一歩進んだ利用例として、ここではマルコフ連鎖を使って要約などの作業を手軽に行えるテクニックを見てみましょう。明治の文豪の文章の要約に挑戦してみます。

ここで学ぶポイント	ツールやライブラリの一覧
●マルコフ連鎖	●MeCab ●Node.js

マルコフ連鎖を使った文章の要約

マルコフ連鎖とは、確率過程の一種です。これは、ロシアの数学者マルコフによって研究されたもので、物理学や統計学の基本的なモデルに応用されています。ここでは、難しいことは抜きにして、マルコフ連鎖を利用して文章の要約に挑戦してみます。

マルコフ性（Markov property）とは、「次の状態が過去の状態に依存せず現在の状態のみによって決まる」という性質のことを言います。

マルコフ性が存在する場合、状態が {q0, q1, q2, q3, ..., q(n-1)} と、n 通りの状態を取り得ると考えます。現在の状態が qi であった時に、次の状態 qj へ遷移する確率は、次の状態と現在の状態のみで記述でき、これを P(qj | qi) で決定します。同様に、状態遷移した順に並べた順序列 {a0, a1, a2, …, a(m-1)} の生成確率は、Π i = 1m - 1P(ai | a(i-1)) と表すことができます。

これを利用すると、仕組みが単純でありながら、時に面白い文章を生成してくれます。そのため、要約のほかに、文章の自動生成にも利用されることが多いようです。最近では、機械的に文章を生成し、それを Twitter に自動で発言するようなプログラムも多く作られています。

作文の仕組みを箇条書きすると以下のようになります。

(1) **文章を単語に分解（形態素解析）する**
(2) **単語の前後の結びつきを辞書に登録する**
(3) **辞書を利用してランダムに作文する**

辞書の作り方にちょっと特徴があり、分かち書きした文章を、前後の結びつきで登録していきます。例えば、「彼の猫は可愛い」という文章ならば、「彼 | の | 猫 | は | 可愛い」と文章を分割します。そして、これを単語の前後に注目して、3 要素ごとに辞書へ登録していきます。辞書は次のように登録します。

```
彼 | の | 猫
の | 猫 | は
は | 可愛い
```

同様にして「あのホテルの朝食は美味しい」という文章であれば、次のように登録されます。

```
あの | ホテル | の
ホテル | の | 朝食
の | 朝食 | は
朝食 | は | 美味しい
```

このような辞書を元にして同じ組み合わせを持つ単語と単語をランダムに組み合わせると、文章ができあがるという仕組みです。ただし、前後の単語の関連性が薄く、また意味を考えて文章を作るわけではないので、組み合わせによっては、でたらめな文章になることもあります。

マルコフ連鎖の実装

では、本題となるマルコフ連鎖の実装部分のコードを紹介します。今回、辞書の生成に関して、JavaScript の Object を利用しています。つまり、単語の組み合わせ辞書をメモリー上に配置することになります。そのため、大きな文章を読み込ませると、メモリー不足でエラーになってしまいます。とはいえ、その分、プログラム自体は簡潔です。形態素解析をしてそれを辞書に登録し、その後、辞書を元に作文を行うというものです。

実際のプログラムは以下のようになります。このプログラムでは、前節で作成した MeCab のモジュール「mecab-mod.js」を利用します。「sample.txt」に記述したテキストファイルを読み込んで、要約した内容を画面に表示します。

● file: src/ch05/03-markov/markov.js

```
// マルコフ連鎖で文章を要約する
var SENTENCE_COUNT = 3; // 3文作文する
var Mecab = require('./mecab-mod.js');
var mecab = new Mecab();
var fs = require('fs');

// サンプルテキストファイルを読む
var text = fs.readFileSync("sample.txt", "utf-8");
// 形態素解析して作文する ---- (※1)
mecab.parse(text, function(err, items) {
  var dic = makeDic(items);
  makeSentence(dic);
});

// マルコフ連鎖用辞書の作成 ----- (※2)
function makeDic(items) {
  var tmp = ["@"];
  var dic = {};
  for (var i in items) {
    var t = items[i];
    var word = t[0];
    word = word.replace(/\s*/, '');
    if (word == "" || word == "EOS") continue;
    tmp.push(word);
    if (tmp.length < 3) continue;
    if (tmp.length > 3) tmp.splice(0, 1);
```

```
    setWord3(dic, tmp);
    if (word == "。") {
      tmp = ["@"];
      continue;
    }
  }
  return dic;
}
function setWord3(p, s3) {
  var w1 = s3[0], w2 = s3[1], w3 = s3[2];
  if (p[w1] == undefined) p[w1] = {};
  if (p[w1][w2] == undefined) p[w1][w2] = {};
  if (p[w1][w2][w3] == undefined) p[w1][w2][w3] = 0;
  p[w1][w2][w3]++;
}

// 辞書を元に作文を行う ---- (※3)
function makeSentence(dic) {

後略
```

サンプルとして、夏目漱石の「こころ」の一部を「sample.txt」という名前で保存しました。テキストは、UTF-8 で保存しました。

プログラムを実行するには、以下のコマンドを実行します。

```
$ node markov.js
```

プログラムを実行すると、例えば、以下のような文章が生成されます。

よそよそしい頭文字などはとても使う気にならない。
ある時は海の中に裹まれて、出掛ける事にした。
それで夏休みに当然帰るべきはずであったので、私を呼び寄せた友達からぜひ来いという端書を受け取ったので、生活の程度は私とそう変りもしなかったのであった。

ランダムに文章を作るので、実行するたびに表示される文章が変わります。

筆を執っても心持は同じ事である。
それに海へはいりに出掛けた。
暑中休暇を利用して海水浴に行った友達からぜひ来いという端書を受け取った。

もう一度、実行してみました。

ところが私が先生と知り合いになった。
それに肝心の当人が気に入らなかった。
学校の授業が始まるには長い畷を一つ越さなければ手が届かなかったのは鎌倉でも辺鄙な方角にあったけれども友達は中国のある資産家の息子で金に不自由のない男であったので、生活の程度は私とそう変りもしなかったのである。

「こころ」を読んだことがあれば、なんとなく、物語の雰囲気が伝わってくるものになっているのではないでしょうか。

しかし、ランダムに文章を選んで作成しているので、やはり、意味はよくわからないものになりがちです。

プログラムについて

それでは、プログラムについて見てみましょう。

プログラムの (※ 1) がメイン処理を記述している部分です。MeCab を利用して形態素解析をしています。解析ができたら、makeDic() 関数で分割した形態素を辞書に登録し、makeSentence() 関数で作文を行います。

プログラムの (※ 2) で作文用の辞書を作成しています。先ほど解説した通り、三語を一セットとして辞書に登録します。JavaScript の Object に単語の組み合わせを保存していくという仕組みです。ここでは、文頭を表す記号「@」から始めて辞書に登録しています。これによって、作文の際に適切な文頭を選ぶことができます。

そして、プログラムの (※ 3) ですが、辞書を元に作文をします。文頭を表す "@" から始まる語句をランダムに選び出し、続いて、第二、第三の語句を辞書から候補を選びます。候補から選んだ第二、第三の語を元にして、続く語句を辞書から選び、選んだ語を元にして、さらに続く語句を選びます。これを句点まで続けることで文章が生成されるというわけです。

この節のまとめ

- **マルコフ連鎖を利用した要約に挑戦してみました。**
- **要約と言ってもランダムに単語を組み合わせるだけなので、時には意味不明な文章になることもあります。**
- **正確な文章を作るには、各文章の重要度を測ったり、重複を避けて文字を取り出すなど、もう少し工夫すると良いでしょう。**

04

簡単な文章校正ツールを作ろう

形態素解析の利用例として、ここでは簡単な文章校正ツールの制作に挑戦してみます。単語数のカウントや助詞「の」の連続やカッコの非対応、文字種の不整合などを指摘するツールを作ってみます。

ここで学ぶポイント	ツールやライブラリの一覧
●形態素解析の利用例 ●文章構成ツールの作成	●Node.js ●MeCab

文章校正ツールについて

大なり小なり、私たちは日々の生活の中で文章を書く機会があります。業務の報告書や、商品の宣伝文や操作マニュアル、また、趣味のブログなど、多くの機会があります。そこで、活用したいのが文章の校正ツールです。Microsoft の Word や、ジャストシステムの一太郎には標準で備わっている機能です。ここでは、簡単な形態素解析の利用例として文章の校正ツールを作ってみたいと思います。

ここで作成するプログラム

ここで作成するのは、文章校正ツールです。文章の書き間違い、文法の間違い、表記の揺れをチェックします。テキストファイルをプログラムの引数として渡すと、テキストを解析して校正点を指摘するというものにします。

今回対応したのは、次の校正ポイントです。

- **助詞「の」の連続**
- **長すぎる文章**
- **並列助詞「～したり、～したり」が対応していない場合**
- **同じ接続詞が連続で使われている場合**
- **「いい訳」と「言い訳」など表記の揺れ**
- **「プログラマー」と「プログラマ」など音引きの有無の相違**

実際のプログラム

では、実際のプログラムを見てみましょう。前節同様に、本章で作成した「mecab-mod.js」を利用します。

● file: src/ch05/04-kousei/kousei.js

```
// 文章の校正ツール
var MAX_WORD = 40; // 最大単語数の警告

var Mecab = require('./mecab-mod.js');
var mecab = new Mecab();
var fs = require('fs');

// 引数をチェック ---- (※1)
var args = process.argv;
args.shift(); // 除去 'node'
args.shift(); // 除去 スクリプトのパス
if (args.length <= 0) {
  console.log('node kousei.js textfile');
  process.exit();
}
var filename = args.shift();

// ファイルを読み込む --- (※2)
var text = fs.readFileSync(filename, "utf-8");
// 形態素解析を行う
mecab.parse(text, function(err, items) {
  checkSentence(items);
});

// 文章をチェックする
function checkSentence(items) {
  checkJosiNo(items);
  checkTaiou(items);
}

// 助詞の「の」の連続と単語数の長さを確認する ------ (※3)
function checkJosiNo(items) {
  var cnt = 0; // 助詞「の」が出現した回数を数える
  var cur = []; // 現在読み込んでいる文を保存する
  var lineno = 1; // 行番号を数える
  for (var i in items) {
    var it = items[i];
    var w = it[0];
    if (w == "EOS") { // 改行
      lineno++; cur = []; cnt = 0;
      continue;
    }
    // 文末および句点の確認 ---- (※4)
    if (w == "。" || w == "、") {
      // 「の」の回数
      if (cnt >= 3) {
        console.log("[警告] 助詞「の」が" + cnt + "回連続しています。");
        console.log("\t(" + lineno + "行目)" + cur.join(""));
      }
```

```
      // 単語数を確認 --- (※5)
      if (cur.length >= MAX_WORD) {
        console.log("[警告] 一文が長すぎます。" + cur.length + "以上の単語です。");
        console.log("\t(" + lineno + "行目)" + cur.join("|"));
      }
      cnt = 0;
      if (w == "。") { cur = []; }
      continue;
    }
    //「の」があるか確認 ---- (※6)
    if (it[0] == "の" && it[1] == "助詞") cnt++;
    cur.push(w);
  }
}

// 対応チェック ----- (※7)
function checkTaiou(items) {
  var heiritujosi = 0, cur = [], lineno = 1;
  var meisi = {};
  var setuzokusi = {}, oldCur = [];
  for (var i in items) {
    var it = items[i];
    var w = it[0];
    if (w == "EOS") { // 改行
      lineno++;
      setuzokusi = {};
      oldCur = cur; cur = [];
      continue;
    }
    // 文末の処理 ----- (※8)
    if (w == "。") {
      if (heiritujosi == 1) {
        console.log("[警告] 並立助詞「?たり」が一度しか出現しません。");
        console.log("\t(" + lineno + "行目)" + cur.join(""));
      }
      oldCur = cur; cur = []; heiritujosi = 0;
      continue;
    }
    // 並立助詞「たり」のチェック --- (※9)
    if (it[2] == "並立助詞" && (w == "たり" || w == "だり")) {
      heiritujosi++;
    }
    // 接続詞のチェック(一行に同じ接続詞が出てこないようにする) --- (※10)
    if (it[1] == "接続詞") {
      if (typeof(setuzokusi[w]) == "undefined") {
        setuzokusi[w] = 1;
      } else {
        console.log("[警告] 一行に同じ接続詞「" + w + "」が複数回使われています。");
        console.log("\t(" + lineno + "行目)" + oldCur.join(""));
      }
    }
    // 表記の揺れチェック --- (※11)
    if (it[1] == "名詞" && w.length >= 2) {
      var kana = it[8];
      if (kana == undefined) kana = it[0]; // 辞書にない単語対策
      kana = kana.replace(/ー/g, ''); // カタカナ対策
      if (meisi[kana] == undefined) {
        meisi[kana] = w;
      } else if (meisi[kana] != w) {
        console.log("[確認] 表記の揺れ: " + meisi[kana] + " != " + w);
      }
    }
```

```
    cur.push(w);
  }
}
```

プログラムを実行するには、以下のようにします。

```
$ node kousei.js (text-file)
```

この校正ツールプログラムを動かすために、以下のような間違いだらけのサンプルテキストを用意しました。

● file: src/ch05/04-kousei/sample.txt

```
今日は朝早く起きました。天気は晴れです。

●「の」の連続
でも今日の朝食の卵焼きの醤油が辛かったのです。
学校で私の隣の席のジェニーはよく笑います。

●並立助詞の不整合
先週は、山に行ったり自然を堪能した日でした。
そこで食べたり飲んだり歌ったりしました。

●カタカナと漢字による表記揺れ
リンゴを食べたり、イチゴを食べたり、ミカンを食べたり、食べてばかりでした。
林檎は美味しい。苺も美味しい。

●「ー」による表記揺れ
Perl プログラマー対 PHP プログラマの対決を見よう。
サーバーの性能を確かめよう。あのサーバは快適に使える。

●長文のチェック
毎朝しっかり食べないと元気が出ないので、学校に行く前は必ずしっかり食べるようにと、母から言われていますので、朝ご飯に魚とご飯と味噌汁が必要です。

●接続詞の重複
鮭は寿司ネタである。しかし、鮭はムニエルにもする。しかし、鮭が好きである。
```

すると、以下のように間違いを指摘します。

```
[警告] 助詞「の」が 3 回連続しています。
  (4 行目) でも今日の朝食の卵焼きの醤油が辛かったのです
[警告] 助詞「の」が 3 回連続しています。
  (5 行目) 学校で私の隣の席のジェニーはよく笑います
[警告] 一文が長すぎます。40 以上の単語です。
  (20 行目) 毎朝 | しっかり | 食べ | ない | と | 元気 | が | 出 | ない | ので ...
[警告] 並立助詞「? たり」が一度しか出現しません。
  (8 行目) 先週は、山に行ったり自然を堪能した日でした
[確認] 表記の揺れ: リンゴ != 林檎
[確認] 表記の揺れ: プログラマー != プログラマ
[確認] 表記の揺れ: サーバー != サーバ
[警告] 一行に同じ接続詞「しかし」が複数回使われています。
  (23 行目) しかし、鮭はムニエルにもする
```

プログラムについて

では、プログラムの仕組みについて見ていきましょう。いずれも、その仕組みはそれほど難しいものではありません。形態素解析さえしてしまえば、各要素を一つずつ調べていくだけだからです。

プログラムの (※ 1) では、コマンドラインから指定されたファイル名を読み込んでいます。配列変数 process.argv にパスが格納されています。

プログラムの (※ 2) では、テキストファイルを読み込みます。テキストを読み込んだら、MeCab を利用して形態素解析を行います。解析が完了したら、文章のチェックを行う checkSentence() 関数を実行します。

ここから、実際の確認処理となります。プログラム (※ 3) の checkJosiNo() 関数では、助詞「の」の連続や、一文が長すぎる場合の確認を行います。

確認の仕方ですが、for 構文を利用して、形態素解析した各要素を一つずつチェックしていきます。助詞「の」が連続するのがよくないケースは、句読点で区切った中に複数の「の」が出現する場合です。そこで、プログラムの (※ 4) にあるように、句読点ごとに「の」の回数をチェックするという処理を行います。ここでは、3 回以上「の」が連続した場合に、警告を表示するようにしました。

また、(※ 5) の部分では、長すぎる一文を警告するようにしています。あまり、一文が長すぎると読んでいて疲れます。とは言え「長すぎる」というのがどの程度なのかは、人によって異なると思います。そこで、MAX_WORD という変数を宣言し、プログラムの冒頭でこの値を指定するようにしました。デフォルトは 40 語です。

プログラムの (※ 6) では、助詞「の」を数えています。ただの「の」を数えると助詞以外のものもカウントしてしまうので、形態素解析した結果に含まれる品詞情報を利用しています。

プログラム (※ 7) にある関数 checkTaiou() では、並立助詞の「〜たり、〜たり」が対応しているか、また、連続で同じ接続詞が使われていないか、表記の揺れがないかの確認をしています。ここでも同じように、改行ごと、句点ごとに確認処理を行っています。

プログラムの (※ 8) と (※ 9) では、並列助詞の確認を行います。(※ 9) の部分で、並列助詞が出てきた回数をカウントし、文末の (※ 8) で警告を出します。ちなみに「並列助詞」とは「食べたり、飲んだり」のように「〜たり、〜たり」と組み合わせで使うものです。そのため、この助詞が文に一度しか出てこない場合文法が間違っている可能性があります。

プログラム (※ 11) では表記の揺れをチェックします。ここでは、名詞句で二語以上のものを対象としています。例えば、文章の中に「林檎」と「リンゴ」のような不揃いの表現が出てくるのを指摘します。ただし、ヨミガナだけを利用して揺れをチェックしているので、まったく問題がなくても表記の揺れとして表示してしまう場合があります。そこで、ここでは控えめに「確認」というメッセージを表示しています。また、「サーバー」と「サーバ」のように音を伸ばすかどうかの違いを確認するため、ヨミカナを確認用辞書に登録するときに「ー」を削った状態で登録して、表記揺れがないかを確認します。

この節のまとめ

- この節では形態素解析利用した校正ツールを自作してみました。
- サンプルではそれほどたくさんのチェックは行いませんので、改良して精度の良いものにしてみてください。
- 自分がよく書き間違える癖があるならば、その文法チェックを加えて、自作校正ツールを強化するのもいいでしょう。

05 語句の出現頻度を調べよう

文章に出現する頻出語句がわかれば、その文章の特徴を浮き彫りにすることができます。ここでは、形態素解析を利用するもう一つの例として、語句の出現頻度を調べるプログラムを作ってみます。文章に出現する頻出単語を表示させてみましょう。

ここで学ぶポイント

- 形態素解析の利用例

ツールやライブラリの一覧

- Node.js
- MeCab

語句の出現頻度を調べる

実際のところ、形態素解析さえ行ってしまえば、頻出語句を調べるのはそれほど難しい問題ではありません。JavaScript の Object 型を利用すれば、手軽に単語の出現回数を数えることができます。

まずは、あまり深く考えず、出現頻度をざっと調べるプログラムを作ってみましょう。

● file: src/ch05/05-hindo/hindo.js

```
// 単語の出現頻度を調べる

中略

// 出現頻度を調べる
function checkHindo(items) {
  // 語句をオブジェクトに格納し頻度を調べる
  var words = {};
  for (var i in items) {
    var it = items[i];
    var w = it[0];
    if (words[w] == undefined) {
      words[w] = 1;
    } else {
      words[w]++;
    }
  }
  // 語句を出現頻度にソートするため配列にコピー
  var list = [];
  for (var key in words) {
    list.push({
      "word":key,
      "nums":words[key]
    });
  }
```

```
  // ソートする
  list.sort(function(a, b){
    return b.nums - a.nums;
  });
  // 画面に出力する
  for (var i = 0; i < list.length; i++) {
    var it = list[i];
    console.log((i + 1) + ":" + it.word + "(" + it.nums + ")");
  }
}
```

このプログラムを実行するには、コマンドラインで以下のようなコマンドを実行します。

```
$ node hindo.js (textfile)
```

例えば、夏目漱石の「こころ」で冒頭の一節の出現頻度を調べてみると、次のような結果となります。

```
$ node hindo.js sample.txt
1:に(47)
2:た(45)
3:は(44)
4:。(39)
5:の(34)
6:を(34)
7:で(27)
8:て(25)
9:、(24)
10:私(21)
11:が(17)
12:と(15)
13:も(12)
?以下省略?
```

実行結果を見てみると、助詞や句読点がたくさん抽出されているのがわかります。日本語の文章を形態素解析して、出現頻度順に並べると、助詞や句読点が多く列挙されてしまいます。そのため、意味のありそうな語句はそれほど上位に表示されません。

助詞や句読点を除外する

そこで、次に助詞や句読点を表示しないようにプログラムを少し改良してみましょう。

● file: src/ch05/05-hindo/hindo2.js

```
// 単語の出現頻度を調べる(その2)

中略

// 出現頻度を調べる ---- (※1)
function checkHindo2(items) {
  // 語句をオブジェクトに格納し頻度を調べる
  var words = {}; // --- (※2)
  for (var i in items) {
    var it = items[i];
    var w = it[0];
    var h = it[1];
```

```
    // 意味のない語句を除外する ---- (※3)
    if (h != "名詞" && h != "動詞" && h != "形容詞") continue;
    if (words[w] == undefined) {
      words[w] = 1;
    } else {
      words[w]++;
    }
  }
  // 語句を出現頻度にソートするため配列にコピー --- (※4)
  var list = [];
  for (var key in words) {
    list.push({
      "word":key,
      "nums":words[key]
    });
  }
  // ソートする --- (※5)
  list.sort(function(a, b){
    return b.nums - a.nums;
  });
  // 頻出上位の語句を画面に出力する
  for (var i = 0; i < 15; i++) {
    var it = list[i];
    console.log((i + 1) + ":" + it.word + "(" + it.nums + ")");
  }
}
```

コマンドラインで以下のコマンドを入力して結果を確かめてみましょう。

```
$ node hindo2.js sample.txt
1:私(21)
2:い(11)
3:の(9)
4:し(9)
5:こと(7)
6:人(6)
7:先生(5)
8:あっ(5)
9:事(5)
10:友達(5)
11:いる(5)
12:彼(5)
13:着(4)
14:海(4)
15:鎌倉(4)
```

今度は、特徴的な語句が抽出されました。出現語句の一位が「私」となりました。確かに、小説「こころ」は、「私」の視点で書かれており、冒頭の部分は、自己紹介的な部分を多く含んでいることがわかります。その他、気になる語句としては「先生」や「友達」「鎌倉」というものがあります。これらは、「こころ」を語る上で外せないキーワードと言えます。頻出語句を調べることで、文章の性質を浮き彫りにすることができました。

プログラムについて

少しプログラムの仕組みについて調べてみましょう。特にポイントとなるのは、プログラム (※ 1) の部分で、出現頻度を調べる checkHindo2() 関数です。出現頻度を調べるには、JavaScript の Object 変数を利用するのが簡単です。ここでは、プログラムの (※ 2) のように、オブジェクト変数 words を利用して、出現回数を調べています。形態素解析した各要素を for 構文で一つずつ調べ、オブジェクトの要素をカウントアップしていきます。例えば、オブジェクトの要素をカウントアップした時、変数 words は以下のようになっています。

```
{ '私': 21,
  '人': 6,
  '先生': 5,
  '呼ん': 1,
  'い': 11,
  'ここ': 7,
  '書く': 1,
  ...
}
```

しかし Object 型では、出現頻度順に並び換えるのが面倒です。そこで、プログラムの (※ 4) のようにして、一度、配列変数に移し替えておきます。そうすれば、プログラム (※ 5) のように sort() メソッドを使って、出現頻度順に並び換えることができます。ソートした後、変数 list の値は以下のようになっています。

```
[ { word: '私', nums: 21 },
  { word: 'い', nums: 11 },
  { word: 'の', nums: 9 },
  { word: 'し', nums: 9 },
  { word: 'ここ', nums: 7 },
  { word: '人', nums: 6 },
  { word: '先生', nums: 5 },
  { word: 'あっ', nums: 5 },
  { word: '事', nums: 5 },
  ...
]
```

Array.sort() メソッドに、関数を指定すると任意の項目を利用したソートが可能になります。以下は、プログラム (※ 5) から sort() メソッドを使う部分を抜粋したものですが、オブジェクト変数 a と b に対して、そのプロパティ nums の値を利用してソートを行うというものになっています。

```
list.sort(function(a, b){
  return b.nums - a.nums;
});
```

また、プログラム (※ 3) の部分で、名詞・動詞・形容詞以外の語句を除去しています。これにより助詞の「は」「の」、句読点など、無意味な語句を除外することができます。

この節のまとめ

- この節では文章から頻出語句を調べるプログラムを作ってみました。
- 形態素解析を利用すれば、意味のある単語を抽出することが容易になります。
- ここでの例では、助詞や記号などを除外することで、注目に値する単語を抜き出すことができました。
- いろいろな文章の頻出語句を調べてみると、文章の特徴がよくわかりますので試してみてください。

第6章

クローラーのためのデータソース

ネット上に公開されている有益なデータソースと、その扱い方を紹介していきます。SNS や Web サービスや無料データなど、一般公開されている役立つデータとデータの活用法、利用法を紹介していきます。

01

有益なデータソースの一覧

本章ではWebからのデータ収集を行う上で、欠かせない貴重で有益な情報を提供しているサイトを紹介します。をそして、この節では具体的にどのようなデータが公開されていて、何に役立てることができるのか紹介していきます。実際のデータ収集に役立ててみてください。

ここで学ぶポイント	ツールやライブラリの一覧
●どんなWebサイトがあるか	●なし

データソースについて

ここでは、Webに公開されているさまざまな有益なデータソースについて紹介します。多くの大手サービスでは、Web APIを提供しており、APIを通して開発者が情報をダウンロードして活用できるようにしています。

なぜ、わざわざAPIが提供されているのかを考えてみると、純粋に開発者を支援したいという目的もあるでしょうが、サイト全体をクローリングされて、無駄なリクエストによる負荷の発生を防ぎたいという気持ちも多分にあるでしょう。私たち開発者は、有益なデータを提供してくれるサーバーになるべく迷惑を掛けないように、クローリングを行いたいと思うのではないでしょうか。

SNSの活用

SNS(Social Networking Service)とは、インターネット上の交流を通して、社会的なつながりを作り出すサービスのことを言います。FacebookやLINE、Twitterなどがその代表です。誰かと友達関係を設定したり、日記を書いたり、日記にコメントをつけたりすることで、情報交換や会話を楽しむことができます。個人間のコミュニケーションを促進するものとなっています。企業や政府機関などさまざまな分野でSNSの利用が進んでいます。そのため、最新情報やトレンドをいち早くキャッチするには、SNSからのデータ収集が欠かせません。また、多くのSNSでは、Web APIを用意しており、API経由でデータを取得することができます。

名称	URL
Facebook	https://www.facebook.com/
Twitter	https://twitter.com/
Google+	https://plus.google.com/
mixi	http://mixi.jp/

代表的なSNS

ソーシャルブックマークの活用

ソーシャルブックマーク（Social Bookmark）は、オンラインブックマークサービスの発展形です。自分のブックマークをネット上に公開することができます。また、ブックマークにタグ付けすることができるので、同じ指向をもったコンテンツを見つけやすくなります。加えて、話題となっている人気のサイトを調べるのにも役立ちます。

名称	URL
はてなブックマーク	http://b.hatena.ne.jp/
delicious	https://delicious.com/

代表的なソーシャルブックマーク

商品情報の活用

一般に販売されている多くの商品を、Webサイトでオンライン購入することができます。世界中で面白い商品が販売されているので、商品情報も立派なデータと言えます。それぞれ、Web APIを通じて商品情報を検索できるようになっています。うまく利用すれば、商品を紹介して広告収入に結びつけることができます。

名称	URL
Amazon	http://www.amazon.co.jp/
楽天市場	http://www.rakuten.co.jp/
Yahoo! ショッピング	http://shopping.yahoo.co.jp/
価格 .com	http://kakaku.com/

商品情報が見られるサイト

オンライン辞書の活用

Wikipedia など、辞書・百科事典のサイトも有益情報の源です。読み物として面白いだけでなく、「日本の鉄道駅一覧」など各種情報の一覧を調べるのにも適しています。整形して取り出せば重宝すること間違いなしです。また、はてなキーワードは旬のキーワードが更新される傾向があるため、流行語を調べるのに最適です。

名称	URL
Wikipedia(日本語)	https://ja.wikipedia.org/
はてなキーワード	http://d.hatena.ne.jp/keyword/
Weblio	http://www.weblio.jp/
ピクシブ百科事典	http://dic.pixiv.net/
ニコニコ大百科	http://dic.nicovideo.jp/

インターネットで公開されているオンライン辞書

オフライン辞書データの活用

先ほど、オンライン上の百科事典を紹介しましたが、ダウンロードして利用するオフラインの辞書データにもさまざまなものがあります。Vector の各種辞書のコーナーではさまざまな無料の辞書データが配布されています。

種類	名称	URL
類似語辞書	日本語 WordNet	http://compling.hss.ntu.edu.sg/wnja/
英和辞書	英辞郎	http://www.eijiro.jp/
英和辞書	ejdic-hand	http://kujirahand.com/Web-tools/EJDictFreeDL.php
英英辞書	WordNet	http://wordnet.princeton.edu/
各種辞書	Vector > 各種辞書	http://www.vector.co.jp/vpack/filearea/data/writing/dic/

ダウンロードで入手できる辞書データ

ブログサービスの活用

多くのサイトで無料ブログが作成できますが、これを利用することで情報収集を行うことができます。特定のブログに注目するのも一つの方法ですが、無料ブログサービスのトップページでは、人気のブログを紹介しているので、そこから順に役立ちそうなブログをピックアップするのもいいかもしれません。また、ブログであれば、大抵、要約情報である RSS を配信しているので、これを利用することもできます。

名称	URL
Ameba ブログ	http://ameblo.jp/
livedoor Blog	http://blog.livedoor.com/
FC2 ブログ	http://blog.fc2.com/
JUGEM	http://jugem.jp/
Seesaa BLOG	http://blog.seesaa.jp/

天気予報・気象情報の活用

天気予報は、それ自体が非常に有益な情報です。しかし、気温や降雨量の情報を、他の要素と組み合わせることで、まったく別の価値を生み出すこともあるでしょう。例えば、猛烈に暑い日であることがわかれば、耐暑グッズの売り上げがよくなることが予想できます。それに合わせて、ブログで紹介する商品を自動で差し替えるなど、用途はいろいろ考えられます。多くの天気予報サイトが、Web API や RSS を配信しているので、それを活用できます。

名称	URL
Weather Hacks	http://weather.livedoor.com/weather_hacks/
Yahoo! 天気・災害	http://weather.yahoo.co.jp/weather/
OpenWeatherMap	http://openweathermap.org/
drk7.jp > 天気予報情報	http://www.drk7.jp/weather/

気象情報などのサイト

また、次の項で述べるオープンデータとも関連していますが、気象庁では、過去の気象情報を CSV 形式でダウンロードできるようになっています。

名称	URL
過去の気象データ・ダウンロード	http://www.data.jma.go.jp/gmd/risk/obsdl/

過去の気象データが CSV 形式で手に入る

オープンデータの活用

オープンデータ (Open Data) とは、自由に再利用再配布できるようにしたデータのことを言います。つまり、著作権や特許などの制限なしで自由に利用できるようにしたものです。著作権・特許、ライセンス、再利用、再配布が自由なデータは利用価値が高いものです。近年、政府が主導で収集したデータをオープン化して配布するオープンデータや、科学データのオープン化が行われています。

名称	URL
日本政府のオープンデータ	http://www.data.go.jp/
アメリカ政府のオープンデータ	http://www.data.gov/
イギリス政府のオープンデータ	http://data.gov.uk/

オープンデータの例

この節のまとめ

- ここではデータソース一覧を紹介しました。
- これ以降の節では、いろいろなデータソースからのダウンロード方法を解説します。
- ここで紹介したデータソースも、実際のプログラムと一緒に紹介していきます。

02 Twitterからのダウンロード

短文の投稿を共有するWebサービスのTwitterからデータをダウンロードする方法を紹介します。Twitterではリアルタイムに情報が更新されるので、話題になっているトピックスを抽出することができるでしょう。

ここで学ぶポイント	ツールやライブラリの一覧
●Twitter	●Node.js ●「twit」モジュール

Twitterとは何か？

Twitterは、ユーザーが誰でも手軽に情報を発信できるコミュニケーションサービスです。さまざまなシーンで活用されるようになりました。ユーザーは「いま、どうしてる？(What's happening?)」に答える形で140文字以内の短文を投稿していきます。

Twitter APIを使う準備

Twitterのデータをダウンロードするには、無料で使えるTwitter APIを使う方法が推奨されています。Twitter APIを使うには、開発者向けのWebサイトで登録する必要があります。登録には、Twitterのアカウントが必要です。具体的に、APIを使うためには、以下のデベロッパーセンターにアクセスして、アプリケーションを登録します。

```
Twitter Developers
https://dev.twitter.com/
```

ページ右上にある「Sign In」というリンクをクリックして、Twitterアカウントでログインします。そして、ページの下にあるフッター部分の「Manage Your Apps」をクリックします。あるいは、以下のURLにアクセスします。

```
Application Management > Twitter Apps
https://apps.twitter.com/
```

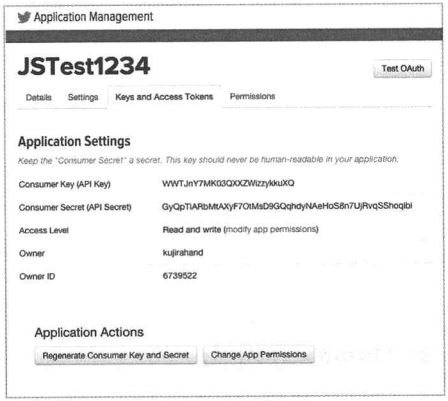
そして、ページの情報にある「Create New App」というボタンをクリックします。すると以下のような画面が出ますので、Name、Description、Website を記入します。Website がない場合は適当なサイトを指定しておいて後で変更します。

Twitter アプリの登録画面

入力したら、「Developer Agreement」の「Yes, I agree」にチェックを入れて、「Create your Twitter application」のボタンをクリックします。

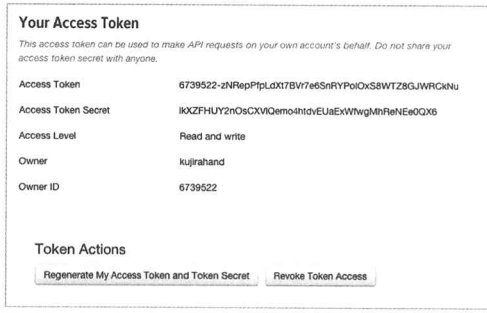
このとき、アプリの名前 (Name) の欄には、他のアプリと被らないユニークな名前を入力しなければなりません。適当な名前を指定してください。また、Description の項目には、10 文字以上の説明を入力する必要があります。加えて、Twitter のアカウントで、モバイルの電話番号を登録していないと、アプリの作成ができないようになっています。Twitter のページ「設定＞モバイル＞自分の携帯電話」で番号を登録し SMS 認証をしておいてください。

Twitter アプリが無事に登録できると「Twitter Apps」の画面に戻ります。そこで、先ほど作成したアプリを選んでクリックします。すると、アプリごとに割り振られたパラメーターの一覧が表示されます。そして、アプリ名のすぐ下にある「Keys and Access Tokens」のタブボタンをクリックします。この中の「Consumer Key (API Key)」と「Consumer Secret (API Secret)」の二つが必要ですのでメモしておきましょう。



Twitter アプリのキー情報

それから、同じページの下の方に「Create my access token」というボタンがあるので、このボタンをクリックして、アクセストークンも取得します。アクセストークンを作成すると、「Access Token」および「Access Token Secret」の情報が得られますので、この値もメモしておきましょう。



アクセストークンの情報

twit モジュールをインストール

次に、Node.js から Twitter API を手軽に使うためのモジュールをインストールしましょう。Twitter API を利用するモジュールはいろいろ公開されていますが、ここでは「twit」モジュールを利用してみます。モジュールのインストールは以下のようにします。

```
$ npm install twit
```

Twitter API を使ったプログラム

それでは、Twitter API を使ったプログラムを作ってみます。プログラムは以下のようになります。このプログラムは、Twitter に投稿された「JavaScript」という単語を含むつぶやきを画面に表示するというものになっています。

● file: src/ch06/02-twitter/tw-api-test.js

```
// Twitterを使うテスト for Node.js
var Twit = require('twit');

// 以下、正しいキーを設定してください★ -----(※1)
var T = new Twit({
  consumer_key: '051dEbfwOFYrQwTaifmeB3AVP',
  consumer_secret: 'ARcMIhyRM8ToY5uRllRVQsOE3TwZoG6lvDxOEeLWzq8HaCn1t6',
  access_token: '6739522-zNRepPfpLdXt7BVr7e6SnRYPolOxS8WTZ8GJWRCkNu',
  access_token_secret:'ikXZFHUY2nOsCXVlQemo4htdvEUaExWfwgMhReNEe0QX6'
});

// JavaScriptに関するつぶやきを表示する ---- (※2)
var stream = T.stream('statuses/filter', {track: 'JavaScript'} );
// つぶやきがあったときに呼ばれるイベントを設定 --- (※3)
stream.on('tweet', function (tw) {
  var text = tw.text;
  var user_name = tw.user.name;
  console.log(user_name + "> " + text);
});
```

プログラムの(※1)の部分では、適当な値が設定されています。プログラムを実際に利用するときは、読者の皆さんがTwitterのデベロッパーセンターにて作成したキーに設定し直してから実行してみてください。

プログラムを実行するには、コマンドラインから次のコマンドを実行します。

```
$ node tw-api-test.js
```

しばらく待っていると、JavaScriptに関連するつぶやきが次々と表示されていきます。これを利用すれば、リアルタイムにJavaScriptの情報を得ることができます。今回は、Streaming APIを利用しているので、次々とデータを取得できるものの過去の情報を取得することはできません。

Twitter APIを使ってJavaScriptの情報を続々と表示させる

それほど難しいプログラムではありませんが、少し解説を入れておきます。プログラムの(※1)では、Twitter APIを使うためのキーを設定しています。

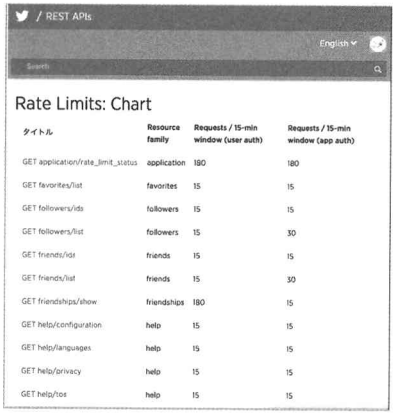
そして、プログラムの(※2)では、Streaming APIを使うために、stream()メソッドを呼び出しています。ここでは、全ユーザーによるつぶやきから特定のキーワードを取得するAPI「statuses/filter」を利用しています。引数で、trackパラメーターに「JavaScript」を指定しています。そのため、JavaScriptに関するつぶやきを取得することができます。もちろん、このtrackパラメーターの値を書き換えることで、任意のキーワードを含む情報を取り出すことができます。

プログラムの(※3)では、つぶやきがあったときに実行される「tweet」イベントを設定しています。このイベントで得られるデータには、実際にユーザーがつぶやいたテキストのほか、ユーザー情報、つぶやいた場所などが含まれています。とは言え、プログラムのテストをするには、情報が多すぎるので、ここでは素直に、つぶやいたテキストと、ユーザーの名前だけをコンソールに出力するようにしています。

Twitter APIの制限

Twitter APIの利用には制限があることが昔から有名ですが、APIの種類ごとに、どんな制限があるのかが以下のページにまとめられています。15分ごとにリクエストをいくつ投げられるかのリストとなっています。

```
Twitter > Rate Limits: Chart
https://dev.twitter.com/rest/public/rate-limits
```



/ REST APIs

English

Search

Rate Limits: Chart

タイトル	Resource family	Requests / 15-min window (user auth)	Requests / 15-min window (app auth)
GET application/rate_limit_status	application	180	180
GET favorites/list	favorites	15	15
GET followers/ids	followers	15	15
GET followers/list	followers	15	30
GET friends/ids	friends	15	15
GET friends/list	friends	15	30
GET friendships/show	friendships	180	15
GET help/configuration	help	15	15
GET help/languages	help	15	15
GET help/privacy	help	15	15
GET help/tos	help	15	15

Twitter API の制限一覧

twit モジュールについて

ここでは、特定のつぶやきを表示するだけのプログラムを紹介しました。しかし「twit」モジュールのページには、つぶやいたり (statuses/update)、フォロワーの一覧を取得したり (followers/ids) する方法などが紹介されています。これを参考にすることで、大抵の操作方法がわかるでしょう。

```
GitHub > twit
https://github.com/ttezel/twit
```

この節のまとめ

- リアルタイムに取得した情報を、次々とデータベースに入れておけば、後から情報を解析することが可能です。
- リアルタイムに流れてくる情報を元に、情報を返信するなど、アイデア次第でいろいろと面白いことができそうです。
- このように、Twitter の API を使えば、何かしらの情報を元に、自動で発言を行う Twitter ボットなども簡単に作れます。

03 Facebookからのダウンロード

SNS の代表格である Facebook からデータをダウンロードするプログラムを紹介します。ここでは、Facebook API を利用して、タイムラインを取得したり、書き込みを行ったりするプログラムを紹介します。

ここで学ぶポイント	ツールやライブラリの一覧
●Facebook APIの使い方	●Node.js ●「fb」モジュール

Facebook とは何か？

Facebook（フェイスブック）は、Facebook, Inc. が運営するインターネット上のソーシャル・ネットワーキング・サービス（SNS）です。「FB」と略されることもあります。2004 年にマーク・ザッカーバーグを中心にした、ハーバード大学の同級生で開発されたのがきっかけです。その後、世界中で使われるようになりました。2012 年には、Facebook のアクティブユーザー数は 10 億人を超えました。13 歳以上であれば無料で参加でき、実名登録制となっています。

Facebook API

Facebook API を利用することで、Facebook の各種機能を利用できるようになっています。API を使うためには、Facebook 開発者ページからアプリを新規作成して、設定キーを取得する必要があります。

```
Facebook 開発者ページ
https://developers.facebook.com/
```

アプリケーションを新規作成

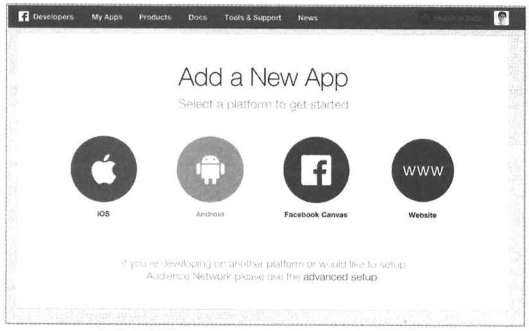
まずは、開発者ページで、新規アプリケーションを登録しましょう。Facebook のアカウントで Facebook にログインし、上記の開発者サイトを訪れましょう。そして、ページ上部にあるメニューから「My Apps ＞ Register as Developer」をクリックします。

開発者ページにアクセス

続いて、プライバシーポリシーに同意するかなど質問が出るので、「はい」に設定して「登録」ボタンをクリックして登録します。

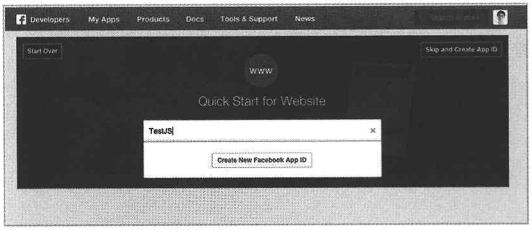
開発者登録する

続いて、ページ上部のメニューから「My Apps ＞ Add New App」をクリックして新規アプリケーションを作成します。すると、プラットフォームを選択するようにというページが出るので、「Website」を選んでクリックします。



Website を選択

続いて、アプリの名前を設定する画面が出るので、「TestJS」など適当なアプリ名を指定します。

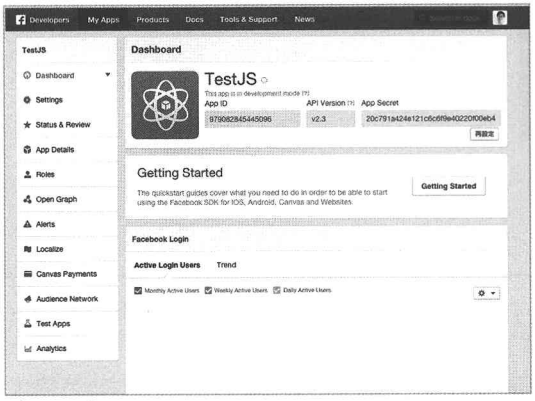


アプリの名前を設定

次に、アプリのカテゴリを選ぶように言われるので、適当なカテゴリを選んで「Create App ID」をクリックします。

カテゴリを設定

すると、Facebookアプリの管理画面（ダッシュボード）が表示されます。



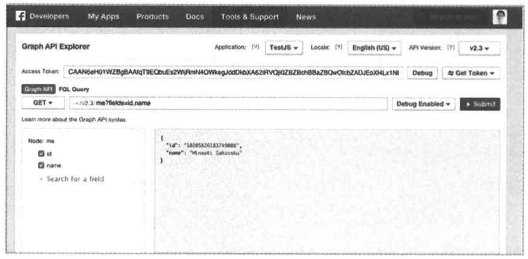
ダッシュボード

この画面にある「App ID」「App Secret」をメモしておきましょう。

次に、Graph API Explorerにアクセスしましょう。ページ上部の「Tools & Support ＞ Graph API Explorer」をクリックするか、以下のURLにアクセスします。

```
Graph API Explorer
https://developers.facebook.com/tools/explorer/
```

ここで、ページ上方にある「Graph API Explorer」の Application で先ほど作成したアプリを選択します。そして、そのすぐ下にある「Get Token」のボタンをクリックします。



Graph API Explorer の画面

すると「アクセス許可を選択」の画面が出ますので、「user_post」など取得したい情報を選んでチェックしておきます。最後に、「Get Access Token」のボタンをクリックします。すると、ポップアップウィンドウが表示され、許可を求められるので OK ボタンをクリックします。その後、Access Token が作成され、API Explorer に表示されます。この値もメモしておきましょう。

アクセストークンを取得

「fb」モジュールのインストール

Node.js で Facebook を操作するモジュールも、いくつか公開されていますが、ここでは「fb」モジュールを利用してみます。以下のコマンドを実行して、インストールしましょう。

```
$ npm install fb
```

Facebook で自分の投稿を表示するプログラム

それでは、プログラムを作ってみましょう。本当は、Facebook で公開されている投稿を検索して取得するようなものを考えていたのですが、Facebook の API バージョンが v2.0 に上がった段階で、そうした検索 API は廃止されてしまったようです。そこで、ここでは、自分の投稿を表示するプログラムを作ってみます。

● file: src/ch06/03-facebook/fb-api-test.js

```
// Facebookを使う for Node.js

// モジュールの取り込み
var FB = require('fb');
// 以下、設定を書き換えること ------ (※1)
FB.setAccessToken('CAAN6eH01WZBgBAHCDgGAxkUFxXknVT5ZCGZA9nPZC9K9GBdOZBVxfT
HZC9KdQdsnbFmjEUlebEdKJGvPi3qyFcvlwZCKCgE8EbUBjHaJIMCeZB16mFQYgwZBhVs3plKU
WVGL1NZASjCfMB8SfyZBLzD4ifg4RUfyQT4YLIukwIO2ZBMkV6M5qo5CLr2rC0uyUcdFAFcKoH
m1CY6fuMIRsUawdWQ1');

// 自分の投稿を取得して表示 ----- (※2)
FB.api('me/feed', 'get', {}, function(feed) {
  if (!feed) {
    console.log("error"); return;
  }
  var data = feed.data;
  for (var i in data) {
    var row = data[i];
    console.log(row);
    console.log("-----------------");
  }
});
```

プログラムの(※1)の部分は、Facebook開発者ページで取得した自分のトークンに書き換えて使ってください。

プログラムを実行するには以下のコマンドをタイプします。

```
$ node fb-api.test.js
```

うまく実行できると、過去に自分が投稿した一覧を取得することができます。

Facebook API を使ったプログラム

プログラムを確認してみましょう。プログラムの(※1)のように、fbモジュールでは、アクセストークンを設定することで利用できるようになります。

プログラムの (※ 2) の部分では、api() メソッドを呼び出しています。このメソッドの引数は、API のエンドポイント、その API に対する操作、パラメーター、API の実行が完了した時の処理の順に指定します。ここでは、自身のタイムラインに表示されているフィードを取得する「me/feed」を指定しています。API の戻り値の data を参照すると、そこにフィードの一覧が配列変数の形式で入っています。

フィードに任意のメッセージを投稿するプログラム

自分の投稿を表示するだけでは、ちょっと物足りなさがありますね。そこで、ここでは、Facebook に任意のメッセージを投稿するプログラムを作ってみました。

● file: src/ch06/03-facebook/fb-api-post.js

```
// Facebookを使う for Node.js

// モジュールの取り込み
var FB = require('fb');
// 以下、設定を書き換えること ------ (※1)
FB.setAccessToken('CAAIZBtT5klAsBAIvKUGp8qQengmKs6Ch2f9K58ZA3mJs0pNBfftOttZ
AZBCcVEkdSFOLuChmEa4P7bHWAm2eE3sJRMHgQQ5fVQD7NeusibRDri8LBBe5GwAZCvxZCZAn
OVpCUVP6Qh2DZCZBHlW8JyqSL6pFmz6ovv3DZB0wQ3gFMmF75tVW21gWFG4VZCjzxcXuLZBkcu
o3nlELMlJeExLvJIkc');

// 任意のメッセージをポストする -------- (※2)
var msg = "API を使ってメッセージを投稿するテストです。";
FB.api('me/feed', 'post', {message: msg}, function(res) {
  if (!res) {
    console.log("error"); return;
  }
  console.log(res);
});
```

もちろん、プログラムの (※ 1) にあるアクセストークンは自分のものに書き換えてください。
以下のコマンドを実行するとプログラムを実行できます。

```
$ node fb-api-post.js
```

うまく実行されると、タイムラインに書き込んだメッセージが表示されます。

Facebook に API から書き込み

プログラムを確認してみましょう。ほとんど、先ほどのプログラムと同じです。異なる部分は、プログラムの (※ 2) の api() メソッドの引数です。ここでは、自身のタイムラインにメッセージを投稿しています。

この節のまとめ

- Facebook のタイムラインの読み込みと、自分のタイムラインへの任意メッセージの書き込みプログラムを紹介しました。
- 実際のところ、Facebook ではログインしないと見られない情報が多く、Facebook API もそのアプリをインストールしたユーザーの情報だけが取得できるようになっています。

04 はてなブックマークからのダウンロード

ソーシャルブックマークサービスの「はてなブックマーク」から有益な情報を取得する方法を紹介します。Web APIが公開されているので、これをうまく活用することで効率的な情報収集ができます。

ここで学ぶポイント

- はてなブックマークを扱う方法

ツールやライブラリの一覧

- Node.js
- 「request」モジュール
- 「cheerio-httpcli」モジュール

はてなブックマークとは何か？

はてなブックマークは、ソーシャルブックマークサービスです。2005年にリリースされており、日本最大のブックマークサービスです。人気のあるブログやWebサイトを知るのに便利です。その機能ですが、タグやコメントをつけて保存、公開が可能なので、ブックマーク数からサイトの人気を知るだけでなく、はてなユーザーのコメントを見ることができます。

```
はてなブックマーク
http://b.hatena.ne.jp/
```

はてなブックマークのページ

はてなブックマークに関連するAPI

はてなブックマークでは、Web APIが公開されています。自分のブックマークを投稿したり、ブックマーク数を取得したり、全文検索一覧を取得するなどのAPIがあります。以下のURLで一覧を確認することができます。

はてなブックマークのAPI
`http://b.hatena.ne.jp/help/entry/api`

はてなブックマークのAPI

ブックマーク数を取得する方法

さて、はてなブックマークのAPIの基本は、AtomAPIが採用されていますが、もっとライトに使えるAPIも存在します。それが、特定のURLがいくつブックマークされているのかを調べるAPIです。

[書式] はてなブックマーク件数取得API
`http://api.b.st-hatena.com/entry.count?url=(URL)`

上記の(URL)の部分に、任意のURLを記述すれば、ブックマーク数をテキストで得ることができるというシンプルなものになっています。

任意のURLのブックマーク数を調べるプログラムは以下のようになっています。なお、「request」モジュールを利用しているので「npm install request」でインストールしておく必要があります。

● file: src/ch06/04-hatenabookmark/bookmark-count.js

```
// はてなブックマークのブックマーク数を取得

// 取得対象URLの指定
var targetURL = "http://www.hatena.ne.jp/";
var COUNT_API = "http://api.b.st-hatena.com/entry.count?url=";

// モジュールの取り込み
var request = require('request');

// APIの指定
var url = COUNT_API + escape(targetURL);
```

```
request(url, function(err, res, body) {
  console.log("ブックマーク数： " + body);
});
```

プログラムを実行するには、以下のコマンドをタイプします。

```
$ node bookmark-count.js
ブックマーク数： 5516
```

複数 URL のブックマーク数を取得する方法

ところで、ブックマーク数を取得したい場合、一気にいくつものサイトの人気を知りたい場合が多いのではないでしょうか。そこで、複数の URL に対するブックマーク数の取得 API も用意されています。以下は書式ですが、実際は改行やインデントはありません。

```
[書式] 複数 URL のブックマーク数を得る
http://api.b.st-hatena.com/entry.counts
  ?url=(URL1)
  &url=(URL2)
  &url(URL3)
```

すると、次のような JSON 形式のレスポンスが得られます。

```
{
  (URL1):(URL1 のブックマーク数 ),
  (URL2):(URL2 のブックマーク数 ),
  (URL3):(URL3 のブックマーク数 )
}
```

それでは、実際の JavaScript のプログラムで試してみましょう。

● file: src/ch06/04-hatenabookmark/bookmark-count2.js

```
// はてなブックマークのブックマーク数を取得

// 取得対象 URL の指定
var url_list = [
  "http://www.hatena.ne.jp/",
  "https://twitter.com/",
  "http://www.amazon.co.jp/"
];
var COUNTS_API = "http://api.b.st-hatena.com/entry.counts";

// モジュールの取り込み
var request = require('request');

// リクエスト用 URL の作成
var params = [];
for (var i in url_list) {
  params.push("url=" + escape(url_list[i]));
}
var url = COUNTS_API + "?" + params.join("&");

// API から結果の取得
request(url, function(err, res, body) {
```

```
  var obj = JSON.parse(body);
  for (var key in obj) {
    console.log(key + "のブックマーク数： " + obj[key]);
  }
});
```

プログラムを実行するには、以下のコマンドを実行します。

```
$ node bookmark-count2.js
http://www.amazon.co.jp/のブックマーク数： 5858
https://twitter.com/のブックマーク数： 8802
http://www.hatena.ne.jp/のブックマーク数： 5516
```

注意する点としては、ブックマーク数が0の場合、0ではなく空データが返ってきます。ここに気をつけましょう。

人気エントリーを取得する

はてなブックマークで、人気のページの情報を取得することで、流行をいち早くキャッチすることができます。そのために、人気ページ一覧のRSSを取得するAPIが用意されています。どんなRSSが取得できるかの一覧を以下のURLから見ることができます。（下記のページは古い情報と記されていますが、RSSの一覧は新しい「Hatena Developer Center」に載っていないので、こちらを紹介します。）

```
はてなウェブサービス
http://www.hatena.ne.jp/info/Webservices
```

注目エントリーを得るには、次のRSSを参照します。

```
注目エントリーのRSS
http://b.hatena.ne.jp/hotentry?mode=rss

最近登録されたエントリー フィード元のページ
http://b.hatena.ne.jp/entrylist?mode=rss

コンピューターカテゴリーの注目のエントリー
http://b.hatena.ne.jp/hotentry?mode=rss&cname=elec
```

それでは、このRSSを利用して、人気ページのタイトル一覧を表示するプログラムを作ってみます。「cheerio-httpcli」モジュールを利用しているので「npm install cheerio-httpcli」でモジュールをインストールしておく必要があります。

● file: src/ch06/04-hatenabookmark/bookmark-hotentry.js

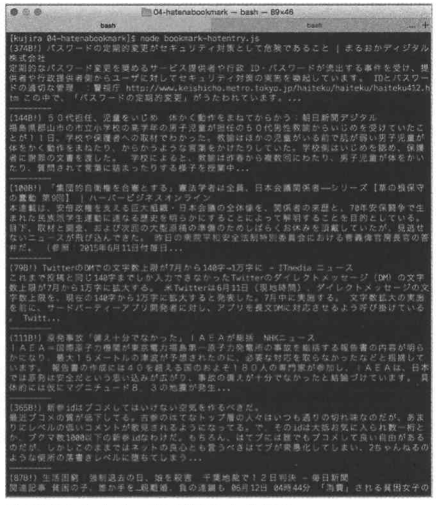
```
// はてなブックマークの注目のエントリーを表示 for Node.js

// 対象RSS
var RSS_URL = "http://b.hatena.ne.jp/hotentry?mode=rss";

// モジュールの読込
var client = require('cheerio-httpcli');

// ダウンロード
```

```
client.fetch(RSS_URL, {}, function(err, $, res) { //----(※1)
  if (err) { console.log("error"); return; }
  // エントリーを抽出して表示 --- (※2)
  $("item").each(function(idx, item) {
    // タイトルと説明とブックマーク数を取得
    var title = $(this).children('title').text();
    var desc = $(this).children('description').text();
    var count = $(this).children("hatena¥¥:bookmarkcount").text();
    console.log("(" + count + "B!) " + title);
    console.log(desc);
    console.log("---------");
  });
});
```

プログラムを実行するには、以下のコマンドを実行します。

```
$ node bookmark-hotentry.js
```

すると、注目のエントリー一覧が表示されます。



注目のエントリーを表示したところ

プログラムについて解説します。プログラムの (※ 1) では、Node.js のモジュール「cheerio-httpcli」を使って RSS を取得します。このモジュールを使うと、ページの取得から XML の解析まで一通りの処理を手軽に行うことができますね。

プログラムの (※ 2) の部分では、RSS から <item> タグの一覧を取り出して、each() メソッドで各タグを一つずつ反復処理します。反復処理の中では、<item> タグの子要素である <title> タグ、<description> タグ、<hatena:bookmarkcount> タグをそれぞれ抽出し、コンソールに出力しています。

ここでちょっと注意しないと生けないのは、「<hatena:bookmarkcount>」のようにコロン「:」を含むタグの指定です。これは、XML 名前空間に基づいたものです。cheerio ライブラリは、jQuery を模して作られていますが、jQuery では「:」はフィルタの働きをします。そのため「:」をエスケープするために「¥¥:」と記述する必要があります。

この節のまとめ

- この節では、はてなブックマークについて、ブックマーク数を調べる方法、そして、注目エントリーの一覧をダウンロードする方法を紹介しました。
- 日本語の情報で、今、話題となっているトピックを調べるのに、はてなブックマークは有益です。

05

Amazonからのダウンロード

Amazonから商品情報を抽出するプログラムを作ってみます。Amazon Product Advertising APIを利用して、商品を検索し、商品広告用のリンクを取得するところまで作ってみます。

ここで学ぶポイント

- AmazonのAPIを使う方法

ツールやライブラリの一覧

- Node.js
- 「apac」モジュール
- 「cheerio」モジュール

Amazonの商品情報

Amazonにはさまざまな商品が陳列されています。インターネット書店からはじめて、今では、電化製品からファッション・食品までいろいろな商品を扱っています。

```
Amazon
http://www.amazon.co.jp/
```

Amazonの Web サイト

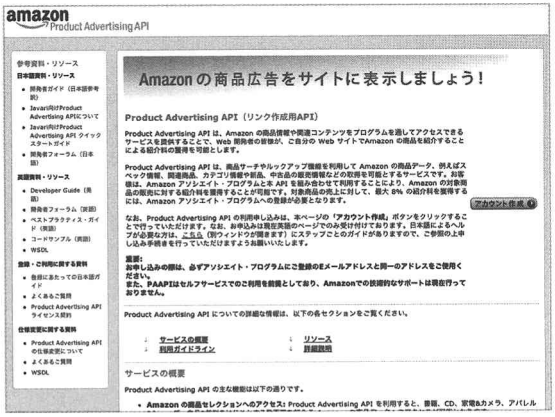
Amazonでは商品情報を、Amazon Product Advertising APIを使って取得できるようにしています。Amazonでは、さまざまなWeb APIを公開しており、Product Advertising APIもその中の一つです。

Product Advertising APIを使うと、商品サーチやルックアップ機能を利用して、Amazonの商品データの取得が可能になります。取得できる情報としては、商品のスペック情報だけでなく、関連商品、カテゴリ情報や新品、中古品の販売情報などがあります。また、情報を検索できるだけでなく、広告リンクを取得できるので、そのリンクを通して商品を買ったお客さんがいれば、紹介料を得ることができます。

API用開発者アカウントを取得する

Product Advertising APIを使うには開発者用のアカウントを取得する必要があります。以下のページにアクセスして、「アカウント作成」のボタンをクリックしましょう。

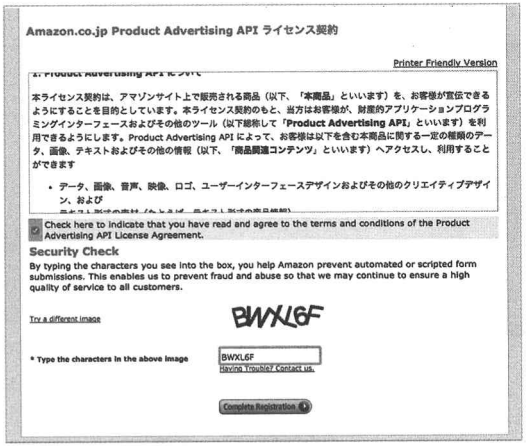
```
Product Advertising API（リンク作成用API）
https://affiliate.amazon.co.jp/gp/advertising/api/detail/main.html
```



Amazon APIのWebサイト

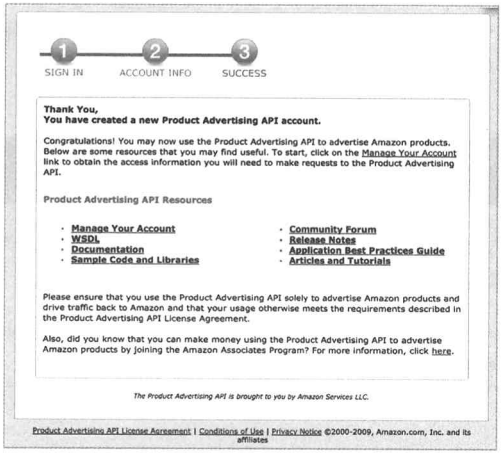
アカウント作成のボタンを押すと、英語に切り替わりますが、それほど難しい手順が必要ではないので安心して進みましょう。

アカウント作成の画面では、住所や電話番号の入力が必要となります。そして、最後ライセンスに同意して、最下部にある[Complete Registration]のボタンをクリックします。



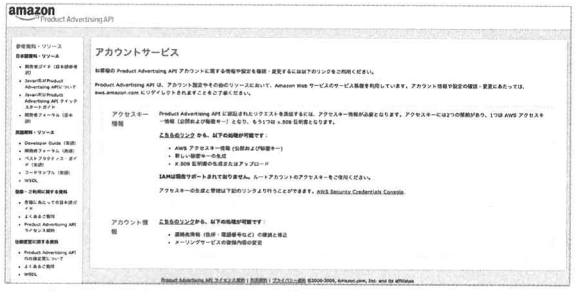
情報を入力してライセンスに同意する

登録が終わったら「Manage Your Account」のリンクをクリックします。



Manage Your Account をクリック

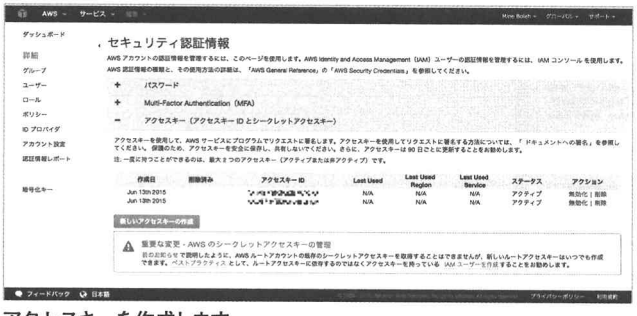
すると、次のような画面が表示されます。ここから日本語に戻ります。そこで「アクセスキー情報」を取得します。「こちらのリンク」というリンクをクリックします。



アカウントサービスのトップページ

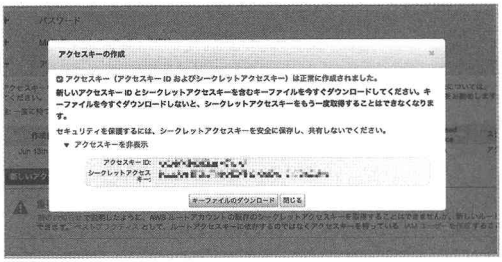
「セキュリティ認証情報」のページが表示されます。そのとき、IM ユーザーを作るようにというダイアログが表示されますが、「セキュリティ認証情報に進む」のボタンを押してください。

そして、「+ アクセスキー」のバーをクリックし、「新しいアクセスキーの作成」ボタンをクリックします。



アクセスキーを作成します

すると、アクセスキー ID とシークレットアクセスキーが作成され表示されます。この値をメモしておきましょう（あるいは、ダウンロードできるので大切に保存しておきましょう）。



アクセスキーを作成します

Amazon の書籍情報を検索する

準備が整いましたのでプログラムを作っていきましょう。まず、Product Advertising API を使うために、「apac」モジュールをインストールします。

```
$ npm install apac
```

宮沢賢治関連の本を Amazon で検索するプログラムを作ってみます。以下のプログラムは、書籍を検索する例です。

● file: src/ch06/05-amazon/amazon-search.js

```
// Amazonの書籍を調べる for Node.js

// モジュールの取り込み
var OperationHelper = require('apac').OperationHelper;

// 以下、アカウント情報を書き入れる --------- (※1)
var opHelper = new OperationHelper({
    awsId:     '**********',  // アクセスキーID
    awsSecret: '**********', // シークレットアクセスキー
    assocId:   'text2musiccom-22', // アソシエイトID
    endPoint: 'ecs.amazonaws.jp',  // 日本のendPointを指定
});

// 書籍を検索する -------------(※2)
opHelper.execute('ItemSearch', {
  'SearchIndex': 'Books',
  'BrowseNode': 465610, // 日本の書籍ID
  'Keywords': '宮沢賢治',
  'ResponseGroup': 'Small,OfferSummary',
  'Sort': 'salesrank',
  'MinimumPrice': 10,
  'MaximumPrice': 8000,
  'Sort': 'salesrank'
}, function(err, results, xml) {
  if (err) { console.log("error"); return; }
  // 結果を表示 --- (※3)
  var Items = results.ItemSearchResponse.Items;
  console.log(Items);
});
```

プログラムを実行するには、以下のコマンドを入力します。

```
$ node amazon-search.js
```

Amazon で書籍検索したところ

プログラムの (※ 1) では、アカウント情報を指定して、OperationHelper オブジェクトを作成しています。アクセスキー ID などの情報を書き換えてから実行してみてください。ちなみにアソシエイト ID(assocId) の指定ですが、これは、別途 Amazon アソシエイトに登録したときに得られる ID です。検索だけ行う場合には、指定する必要はありません。

プログラムの (※ 2) の部分では、execute() メソッドを実行することで、Web API にアクセスしますす。第一引数から、API の種類、オプション、API の戻り値を受け取る処理の順に指定します。

プログラム (※ 3) の部分では、API からの結果をコンソールに出力します。API の結果は、XML で出力されますが、それを、Node.js のモジュール「xml2js」でパースした結果が、変数 results に格納されます。そのため、この results 変数から任意の結果を取り出します。

検索結果を手軽に取り出す

しかしながら、xml2js の結果は非常に複雑なものとなっています。Amazon から返された XML は次のようになっています。

API の結果として戻された XML

結果を取得するには、変数 results から、results.ItemSearchResponse.Items[0].Item[0]... と非常に長い階層をたどる必要があります。

そこで、2章で紹介した「cheerio」モジュールを利用して、jQuery ライクに情報を取り出すようにしてみましょう。以下のコマンドを実行して、cheerio をインストールしてみてください。

```
$ npm install cheerio
```

それでは、このモジュールを利用して検索結果から書籍情報を表示するようにしてみましょう。書籍のタイトル、筆者、ASIN(Amazon の商品番号) を列挙してみます。

● file: src/ch06/05-amazon/amazon-search2.js

```
// Amazon の書籍を調べる for Node.js

// モジュールの取り込み
var OperationHelper = require('apac').OperationHelper;
var cheerio = require('cheerio');

// 以下、アカウント情報を書き入れる ---------- (※1)
var opHelper = new OperationHelper({
    awsId:     '*********',  // アクセスキー ID
    awsSecret: '*********', // シークレットアクセスキー
    assocId:   'text2musiccom-22', // アソシエイト ID
    endPoint: 'ecs.amazonaws.jp',  // 日本の endPoint を指定
});

// 書籍を検索する
opHelper.execute('ItemSearch', {
  'SearchIndex': 'Books',
  'BrowseNode': 465610, // 日本の書籍 ID
  'Keywords': '宮沢賢治',
  'ResponseGroup': 'Small,OfferSummary',
  'Sort': 'salesrank',
  'MinimumPrice': 10,
  'MaximumPrice': 8000,
  'Sort': 'salesrank'
}, function(err, results, xml) {
  if (err) { console.log("error"); return; }
  // XML を解析する ---- (※2)
  var $ = cheerio.load(xml);
  // 商品情報を抽出する ----- (※3)
  $("Items > Item").each(function(idx, item) {
    var ASIN = $(item).children("ASIN").text();
    var author = $(item).find("Author").text();
    var title = $(item).find("Title").text();
    console.log(title + " - "+ author + " : " + ASIN);
  });
});
```

プログラムを実行するには、以下のコマンドを入力します。

```
$ node amazon-search2.js
```

実行すると次のような結果になります。

Amazon から書籍の検索結果をわかりやすく表示

先ほどと同じように、プログラム (※ 1) のアカウント情報は書き換えてください。

プログラムの (※ 2) では、cheerio モジュールを利用して、XML をパースします。そして、(※ 3) では商品情報を抽出してコンソールに出力しています。

この節のまとめ

- 本節では、Amazon から書籍情報を取得する方法を紹介しました。
- Web API を使う場合には開発者登録などが必要で、若干手間がかかりますが、Amazon から情報を詳細に取得できるので活用できそうです。

06

Flickrから写真のダウンロード

米 Yahoo! の写真共有サービスである「Flickr」が提供している Web API を利用して、画像の検索およびダウンロードに挑戦してみます。

ここで学ぶポイント	ツールやライブラリの一覧
●Flickrで写真の検索とダウンロード	●Node.js

Flickr とは何か？

「Flickr」は米 Yahoo! の写真共有サービスです。投稿した画像をブログなどに貼り付けることができる機能があり急速に広まりました。無料のアカウントでも大容量の写真の投稿が可能と話題になりました。

```
Flickr
https://www.flickr.com/
```

Flickr の Web サイト

APIキーを取得しよう

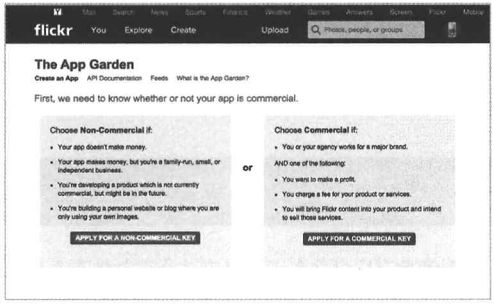
ここまで、Web APIを提供している大手のサービスでは、開発者登録しAPIキーの取得が必要でした。Flickrも同様で開発者登録をしてAPIキーを取得する必要があります。以下、その手順を紹介します。

Flickrのサイトのページ下部（フッター）にある「Get an API Key」のリンクをクリックします。

```
Flickr > Get an API Key
https://www.flickr.com/services/api/keys/apply/
```

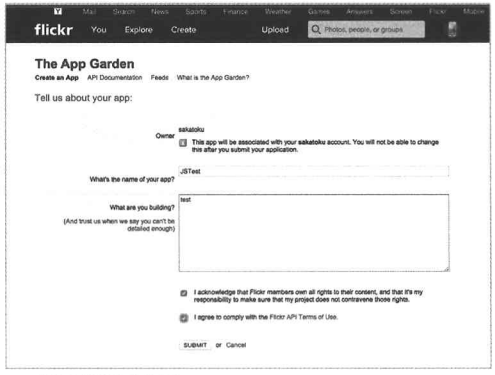
そして、Yahoo!のアカウントでログインします。（このとき、Yahoo!Japanのアカウントは利用できません（Yahoo!とYahoo!Japanのアカウントには互換性がないのです）。

はじめに商用かそうでないかを質問されます。ここでは非商用「APPLY FOR A NON-COMMERCIAL KEY」を選択します。



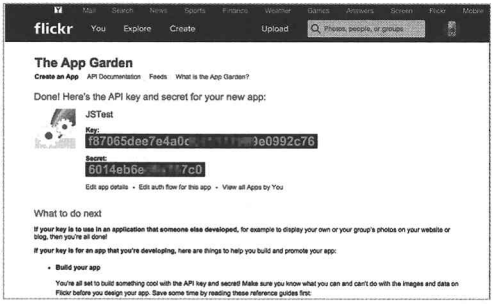
商用かどうかを選択

次に作成するアプリの説明を入力するように求められます。そこで、適当に説明を入力し、利用規約に同意した上で、「SUBMIT」ボタンをクリックします。



アプリの説明を記入

すると、APIキー（KeyとSecret）が表示されます。このキーをメモしておきましょう。



API キーを入手

Flickr API を使ったプログラムを作ろう

それでは、プログラムを作っていきましょう。まずは、Flickr API を使う Node.js のモジュールをインストールします。その後、以下のコマンドを実行します。

```
$ npm install node-flickr
```

Flickr で猫を検索するプログラム

続いて、プログラムを紹介します。ここでは「cat」というテキストを含む写真を検索してみます。

● file: src/ch06/06-flickr/flickr-search.js

```
// Flickrで写真検索 for Node.js

// APIキーを指定 以下を書き換えてください ----- (※1)
var keys = {
  api_key: 'f87065dee7e4a0db71177989e0992c76'
};

// Flickrオブジェクトを作成 ------- (※2)
var Flickr = require('node-flickr');
var flickr = new Flickr(keys);

// 画像を検索 --- (※3)
flickr.get("photos.search", {
  "text": "cat",
  "sort": "interestingness-desc",
  "per_page": 5
}, function (result) {
  // 写真数の情報 ----- (※4)
  var page = result.photos.page;
  var pages = result.photos.pages;
  var perpage = result.photos.perpage;
  var total = result.photos.total;
  console.log("total:", total);
```

```
  // 各写真の詳細情報を取得する
  var photo_list = result.photos.photo;
  for (var i in photo_list) {
    var p = photo_list[i];
    // URLを生成 ---- (※5)
    var url = "https://farm" + p.farm + ".staticflickr.com/" +
      p.server + "/" + p.id + "_" + p.secret + ".JPG";
    console.log(p);
    console.log("URL:" + url);
  }
});
```

プログラムを実行してみましょう。以下のコマンドを実行します。

```
$ node flickr-search.js
```

写真を検索したところ

まず、プログラムの (※1) では、Flickr の API キーを指定しています。自身で取得したキーに書き換えてご利用ください。

プログラムの (※2) では Flickr オブジェクトを作成して、API が利用できるようにします。

プログラムの (※3) では、API を利用して画像を検索します。get() メソッドの引数は、API の種類、API の引数、結果を処理するコールバック関数の順に指定します。ここで利用した API は「photos.search」です。いろいろな条件で写真を検索することができます。詳しいオプションなどの情報は、以下の API マニュアルに記されています。

```
Flickr > The App Garden > flickr.photos.search
https://www.flickr.com/services/api/flickr.photos.search.html
```

さて、API の結果を処理するコールバック関数 (※4) を見てみましょう。コールバック関数で得られる変数 result には、API から返された結果が格納されています。result.photos.total が検索結果の総数です。そして、各写真は、result.photos.photo の配列に入っています。

ところで、API を実行してみると、以下のような実に素っ気ないデータが戻ってきていることがわかります。これでは、どこに画像があるのかわかりません。

```
{ id: '5114665665',
  owner: '41018986@N02',
  secret: 'e55b2c2169',
  server: '1318',
  farm: 2,
  title: 'cat',
  ispublic: 1,
  isfriend: 0,
  isfamily: 0 }
```

Flickrのサイトを見てみると、いろいろな画像が次のようなURLで提供されていることがわかります。

```
実際のURL:
https://farm2.staticflickr.com/1318/5114665665_e55b2c2169.JPG
```

このアドレスを先ほどの写真データの戻り値と照らし合わせてみたところ、次のような書式になっていることがわかりました。

```
https://farm{farm}.staticflickr.com/{server}/{id}_{secret}.JPG
```

プログラムの(※5)では、上記の規則に基づいて、URLを生成しています。

URLをダウンロードしてみよう

さて、以上でAPIの使い方とURLがわかったので画像をどんどんダウンロードしてみましょう。以下は、Flickrで猫を検索し、次々と画像をダウンロードするプログラムです。node-flickrのほか、requestメソッドを利用するので、npmを使ってインストールしておきましょう。

● file: src/ch06/06-flickr/flickr-cat-download.js

```
// Flickrで猫画像をダウンロード for Node.js

// モジュールの取り込み
var Flickr = require('node-flickr');
var fs = require('fs');
var request = require('request');

// APIキーを指定 以下を書き換えてください ----- (※1)
var keys = {
  api_key: 'f87065dee7e4a0db71177989e0992c76'
};
// 何の写真を検索するか ----- (※2)
var KEYWORD = "猫";
// ダウンロード先の指定
var PHOTO_DIR = __dirname + "/photo";

// Flickrオブジェクトを作成
var flickr = new Flickr(keys);

// ダウンロードフォルダを作成 ---- (※3)
if (!fs.existsSync(PHOTO_DIR)) fs.mkdirSync(PHOTO_DIR);

// 画像を検索 ----- (※4)
flickr.get("photos.search", {
  "text": encodeURIComponent(KEYWORD),
```

```
  "sort": "interestingness-desc",
  "per_page": 20,
  "license": 4 // creativecommons by
}, function (result) {
  // 各写真の詳細情報を取得する
  var photo_list = result.photos.photo;
  for (var i in photo_list) {
    var p = photo_list[i];
    // URLを生成
    var url = "https://farm" + p.farm + ".staticflickr.com/" +
      p.server + "/" + p.id + "_" + p.secret + ".JPG";
    // ファイル名を生成
    var fname = PHOTO_DIR + "/" + p.id + ".JPG";
    console.log("+ " + p.title);
    console.log("| URL:" + url);
    // 保存 ----- (※5)
    request(url).pipe(fs.createWriteStream(fname));
  }
});
```

プログラムを実行するには、コマンドラインから、以下を入力します。

```
$ node flickr-cat-download.js
```

実行すると以下のような結果になります。

猫の画像をダウンロードしたところ

プログラムを確認してみましょう。先ほどと同様に、プログラム(※1)の部分、APIキーは自身で取得したものに書き換えてください。

プログラム(※2)の部分では、何の写真を検索するのかキーワードを指定します。その後でどこに画像をダウンロードするのかダウンロード先も指定します。ここではプログラムと同じディレクトリ(__dirname)以下「/photo」にダウンロードするよう指定しています。そして、実際にディレクトリを作成しているのが(※3)の部分となります。

プログラムの (※ 4) では画像を検索しています。ここでは、get() メソッドの第二引数に注目してください。「node-flickr」モジュールでは日本語が考慮されていないようで、うまく日本語のキーワードが検索できないので、encodeURIComponent() メソッドを利用して日本語をエンコードしておきます。

また、検索時にライセンス (license) を 4 番 (クリエイティブコモンズの by) を指定して検索しています。このようにライセンスが指定できるのも Flickr の特徴です。これを利用すれば、素材画像を自動的に検索することも可能になります。

プログラムの (※ 5) では、request モジュールを利用して、画像をダウンロードしています。一行書くだけでダウンロードできるのはいいですよね。これに関しては、2 章でも詳しく紹介しています。

この節のまとめ

- **この節では Flickr API を利用して画像をダウンロードするプログラムを紹介しました。**
- **ここで紹介したように、Flickr の画像は、いろいろな条件で検索することができるので、うまく利用することで、画像の有効活用ができるでしょう。**

07

YouTubeから動画のダウンロード

YouTube から動画を検索したり、ダウンロードする方法を紹介します。YouTube にはさまざまな動画が公開されており、また期間限定のものもあり、効率的にダウンロードすることができれば、必要な時に見返すことができます。

ここで学ぶポイント

- YouTube動画のダウンロード

ツールやライブラリの一覧

- Node.js
- youtube-dl
- youtube-node モジュール

YouTube とは何か？

YouTube は動画共有サービスです。さまざまな動画が投稿されています。最近では、有名企業が CM をアップしたり、ミュージシャンが公式にプロモーションビデオをアップすることも普通になっています。ですから、必要に応じてローカルにダウンロードしておくことができると何かと便利です。

```
YouTube
https://www.youtube.com/
```

YouTube の Web サイト

ここでは、YouTube から動画をダウンロードする方法を紹介します。YouTube では簡単に動画をダウンロードできるような機能は提供されていませんから、ダウンロード用のツール「youtube-dl」を利用します。

youtube-dl のインストール

「youtube-dl」は Python で作られているコマンドラインのプログラムです。そのため、Windows/Mac OS X/Linux で同じように動かす事ができます。

```
youtube-dl
http://rg3.github.io/youtube-dl/
```

Windows 用に実行バイナリファイルが用意されているので、それをダウンロードすることもできます。ダウンロードは、トップページから「Download」のボタンをクリックし、「Windows exe」というリンクをクリックします。

Mac OS X であれば、Homebrew を利用してインストールすることができます。ターミナルを起動して以下のコマンドを実行します。

```
$ brew install youtube-dl
```

また、Python のパッケージ管理システムである「pip」を利用して、インストールすることもできます。

```
$ sudo pip install --upgrade youtube_dl
```

youtube-dl の使い方

「youtube-dl」の使い方は簡単です。コマンドラインから次のように入力します。

```
[書式] youtube-dl の基本的な使い方
$ youtube-dl (YouTube 動画の URL)
```

いろいろ細かくオプションを指定できます。

オプション	意味
-u (ユーザー名)	YouTube のユーザーを指定
-p (パスワード)	YouTube のパスワードを指定
-o (ファイル名)	ダウンロードする動画のファイル名を指定
-s	実際にはダウンロードしないでシミュレーションを行う
-t	ダウンロードする動画のファイル名を動画のタイトルにする
-g	動画の最終的な URL を表示する

youtube-dl のオプション一覧

youtube-dl のアップデート

youtube-dl は、YouTube の公認アプリではありません。そのため、YouTube 側の構造が変更になったときに、youtube-dl が使えなくなってしまいます。しかし、youtube-dl は、GitHub でオープンソースとして開発されており、素早くアップデートされます。うまく動画がダウンロードできなくなったときには、youtube-dl をアップデートしてみると良いでしょう。

youtube-dl 自体にアップデートの機能もありますので、これを利用できます。

```
$ sudo youtube-dl -U
```

これでうまく行かない場合、Windows 版であれば、再度ツールをダウンロードし直してください。Homebrew や pip を使ってインストールしている場合は、以下の方法でアップデートできます。

```
# Homebrew の場合
$ brew update
$ brew upgrade youtube-dl
# pip の場合
$ sudo pip install youtube_dl -U
```

YouTube を検索する

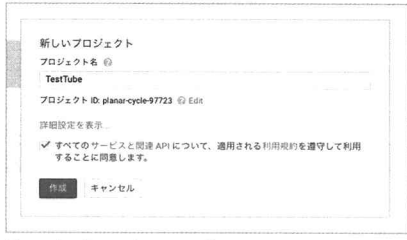
さて、YouTube も動画の検索を行う Web API を公開しています。ここでは、Node.js で動画を検索するプログラムを作ってみます。

YouTube API のキーを取得する

YouTube API を使うには、API キーが必要となります。そのため、Google Developers Console にアクセスして、プロジェクトを新規作成し、YouTube Data API を有効にした上で、API キーを取得します。

```
Google Developers Console
https://console.developers.google.com/
```

以下にその手順を紹介します。まず、上記のサイトに Web ブラウザーでアクセスし、Google アカウントでログインします。そして、新規プロジェクトを作成します。

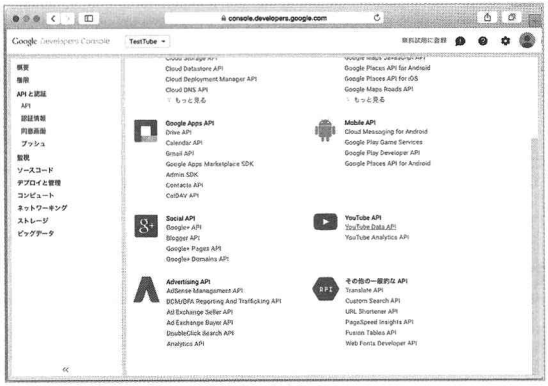


Google Developers Console

プロジェクト名を適当に入れたら「作成」ボタンをクリックします。

プロジェクト名を指定

続いて、画面左側メニューにある「API と認証> API」から「YouTube Data API」をクリックし、API を有効にします。



YouTube Data API を有効にします

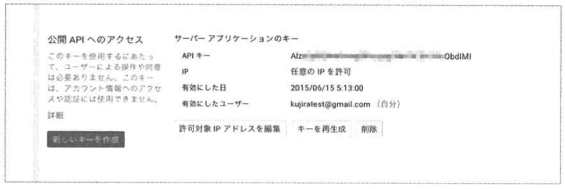
画面左側メニューにある「認証情報」をクリックし、公開 API へのアクセスの項目にある「新しいキーを作成」ボタンを押します。

新しいキーを作成

キーの種類を尋ねられますので「サーバーキー」を選びます。続いて、IP アドレスの設定がでますので、必要なら指定して「作成」ボタンをクリックします。

サーバーキーを作成

すると、API キーが生成されますので、これをメモしておきます。



キーの情報

Node.js のモジュール「youtube-node」をインストール

ここでは、Node.js のモジュール「youtube-node」を利用してみます。npm を利用してインストールしましょう。

```
$ npm install youtube-node
```

YouTube を検索するプログラム

それでは、YouTube でネコ動画を検索するプログラムを見てみましょう。

● file: src/ch06/07-youtube/youtube-search.js

```
// YouTube を検索する for Node.js

var Youtube = require('youtube-node');
var youtube = new Youtube();

// API キーを設定する（以下を書き換え） --- (※1)
youtube.setKey('AIzaSyBSAwRxvrqIBRscyapCMY9CSYU5uObdIMI');
```

```
// 検索を実行 ----- (※2)
var keyword = 'ネコ';
var limit = 3;
youtube.search(keyword, limit, function(err, result) {
  if (err) { console.log(err); return; }
  // 結果を表示
  console.log(JSON.stringify(result, null, 2));
});
```

プログラムの(※1)の部分では、YouTubeのAPIキーを指定していますので、自身で取得したキーに書き換える必要があります。

実行するには、コマンドラインから以下をタイプします。

```
$ node youtube-search.js
```

実行すると画面は以下のようになります。

YouTubeを検索したところ

このように、モジュール「youtube-node」を使えば、それほど難しくなくYouTubeの検索を行うことができます。youtube-nodeの使い方は、次の通りです。

```
// youtube-node のオブジェクトを作成
var Youtube = require('youtube-node');
var youtube = new Youtube();

// 検索 API を実行
var keyword = "***"; // キーワード
var limit = 5; // 件数
youtube.search(keyword, limit, function(err, result) {
  console.log(result); // 結果を表示
});
```

検索オプションを指定する

YouTube の検索 API の機能は、公式リファレンスを見ると、詳しく書かれています。検索オプションを指定することで、さまざまな条件のもと、動画を検索できます。

```
YouTube Data API (v3)
https://developers.google.com/youtube/v3/docs/search/list
```

いろいろありますが、ここでは代表的なオプションを紹介します。

オプション	意味
order	検索結果の順番を指定 (date/rating/relevance/title/videoCount/viewCount)
regionCode	国のコード指定 (日本ならば JP を指定)
safeSearch	検索結果に制限コンテンツを含めるかどうかを指定 (none/moderate/strict)
type	検索対象のタイプを指定 (channel/playlist/video)
videoDuration	動画の長さを指定 (any/long/medium/short)
videoLicense	動画のライセンスを指定 (any/creativeCommon/youtube)

代表的な YouTube の検索オプション

videoLicense オプションを指定するときは、type オプションで video を指定する必要があります。

それでは、オプションを指定して、プログラムを作ってみましょう。以下のプログラムでは、ライセンスが「クリエイティブコモンズ」に設定されている「ネコ」に関係する動画を検索してみます。

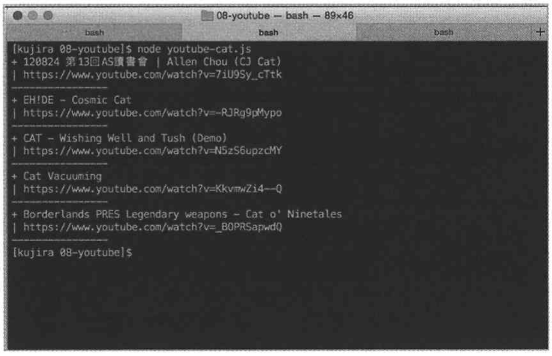
● file: src/ch06/07-youtube/youtube-cat.js

```
// YouTube を検索する for Node.js

var Youtube = require('youtube-node');
var youtube = new Youtube();

// API キーを設定する ( 以下を書き換え ) --- ( ※ 1)
youtube.setKey('AIzaSyBSAwRxvrqIBRscyapCMY9CSYU5uObdIMI');

// 検索を実行
var keyword = 'cat';
var limit = 5;
// オプションを指定 ----- ( ※ 2)
youtube.addParam('order', 'rating'); // 評価順に検索
youtube.addParam('type', 'video');
youtube.addParam('videoLicense', 'creativeCommon'); // クリエイティブコモンズ
youtube.search(keyword, limit, function(err, result) {
  if (err) { console.log(err); return; }
  // 動画の URL を表示する -------- ( ※ 3)
  var items = result["items"];
  for (var i in items) {
    var it = items[i];
    var title = it["snippet"]["title"];
    var video_id = it["id"]["videoId"];
    var url = "https://www.youtube.com/watch?v=" + video_id;
    console.log("+ " + title);
    console.log("| " + url);
    console.log("----------------");
  }
});
```

次のコマンドを入力して、プログラムを実行してみましょう。

```
$ node youtube-cat.js
```

実行すると次のような結果になります。



YouTube で cat（猫）を検索したところ

プログラムについて説明します。先のプログラムと同様に、プログラムの (※ 1) の部分は書き換えてください。

プログラムの (※ 2) の部分では、検索オプションを指定しています。addParam() メソッドを使います。ここでは、order オプションを「rating」にする事で評価の高い動画順に検索します。そして、videoLicense オプションを「creativeCommon」に設定して、クリエイティブコモンズに準拠した動画を検索します。このとき、type オプションで「video」を指定しないとエラーになるので注意が必要です。

プログラムの (※ 3) では、YouTube API の結果を見て、動画の URL を表示します。API の結果は、コールバック関数の引数である result 変数に格納されるので result 変数の内容を確認します。items プロパティに配列形式で動画の一覧が入っています。そのため、items プロパティの各要素を for 構文で一つずつ取り出して表示しています。

ちなみに動画の ID がわかると、YouTube の動画表示ページの URL もわかります。基本的に以下のような形式になっているからです。そこで、動画 ID を得て、コンソールに URL を出力しています。

```
https://www.youtube.com/watch?v=( 動画 ID)
```

動画を検索してダウンロードしよう

ここまでの部分で、YouTube で動画を検索する方法がわかりました。それでは、総まとめとして、動画を検索して、そのままダウンロードするプログラムを紹介します。

● file: src/ch06/07-youtube/youtube-download.js

```
// YouTube を検索する for Node.js

// モジュールの取り込み
var exec = require('child_process').exec;
var Youtube = require('youtube-node');
var youtube = new Youtube();
```

```
// APIキーを設定する（以下を書き換え） --- （※1）
youtube.setKey('AIzaSyBSAwRxvrqIBRscyapCMY9CSYU5uObdIMI');

// 検索を実行
var keyword = 'ネコ';
var limit = 2;
// オプションを指定 ----- （※2）
youtube.addParam('order', 'viewCount'); // ダウンロード順に
youtube.addParam('type', 'video');
youtube.addParam('videoLicense', 'creativeCommon'); // クリエイティブコモンズ
youtube.addParam('videoDuration', 'short'); // 短い動画で
youtube.search(keyword, limit, function(err, result) {
  if (err) { console.log(err); return; }
  // 動画のURLを表示する
  var items = result["items"];
  for (var i in items) {
    var it = items[i];
    var title = it["snippet"]["title"];
    var video_id = it["id"]["videoId"];
    console.log(title, video_id);
    // ダウンロード実行
    downloadVideo(video_id);
  }
});

// youtube-dlを使ってダウンロード ------ （※3）
function downloadVideo(video_id) {
  var url = "https://www.youtube.com/watch?v=" + video_id;
  exec('youtube-dl ' + url, function(err, stdout, stderr) {
    if (err) { console.log(err); return; }
    if (stdout) console.log(stdout);
    if (stderr) console.log(stderr);
  });
}
```

次のコマンドを入力して、プログラムを実行してみましょう。

```
$ node youtube-download.js
```

実行すると次のような結果になります。

動画を検索してダウンロードするプログラムを実行した

プログラムの (※ 1) は自分の Key に書き換えます。プログラムの (※ 2) では、検索オプションを指定しています。ここでは、クリエイティブコモンズの動画に加えて、短い動画を検索するよう、videoDuration のオプションを追加しました。

そして、動画をダウンロードするのが (※ 3) の部分です。本節の冒頭で紹介した「youtube-dl」を使ってダウンロードします。動画 ID から YouTube の動画 URL を作成し、それを、youtube-dl のコマンドに付けて実行するというものになっています。

この節のまとめ

- **この節では、YouTube の検索と動画のダウンロードについて紹介しました。**
- **ここで紹介したプログラムを少し修正して、cron などに登録しておけば、好きなアーティストの動画が投稿されるたびに自動で動画をダウンロードするといった使い方もできます。**

08

Yahoo!Financeから為替や株のダウンロード

定期的にダウンロードしたいデータには、為替や株の情報があります。ここでは、Yahoo!Finance のデータを利用して、為替や株のデータをダウンロードするプログラムを作ってみます。

ここで学ぶポイント

- 為替・株の情報をダウンロードする方法

ツールやライブラリの一覧

- Node.js

Yahoo! ファイナンス

Yahoo! ファイナンスの Web サイトを見ると、株式、FX・為替の速報に加えて、株価予想やニュースなど、さまざまな金融情報が掲載されています。ここから、為替や株価の情報を取得するプログラムを作ってみましょう。

```
Yahoo! ファイナンス
http://finance.yahoo.co.jp/
```

Yahoo! ファイナンスの Web サイト

FX・為替情報を取得する

為替情報は、Yahoo! ファイナンスの以下のページで取得できます。URL 末尾に code がついており、ここに 3 文字の通貨コードの組み合わせを指定することで、任意の通貨ペアを見ることができます。

```
Yahoo! ファイナンス > FX・為替 > 米ドル / 円
http://info.finance.yahoo.co.jp/fx/detail/?code=USDJPY
```

FX・為替のページ

以下が、為替コードの例ですが、例えば、人民元と日本円の組み合わせであれば、?code=CNHJPY というページにアクセスすれば情報を見ることができるように工夫されています。

通貨コード	説明
JPY	日本円
USD	米ドル
EUR	ユーロ
AUD	豪ドル
GBP	イギリスポンド
NZD	ニュージーランドドル
CAD	カナダドル
CHF	スイスフラン
ZAR	ランド
CNH	人民元

通貨コードの例

FX・為替情報を取得するプログラム

では、Node.js を利用して為替データを取得してみます。HTML を見てみると、売値 (Bid) と買値 (Ask) に id が割り振られているので、素早くデータを取り出すことができます。

HTML のダウンロードを行うために、Node.js の「cheerio-httpcli」モジュールを利用していますので、npm を利用してインストールしてから実行してください。

● file: src/ch06/08-finance/fx-USDJPY.js

```
// 為替情報を取得 for Node.js

// モジュールの取り込み
var client = require('cheerio-httpcli');

// HTML をダウンロード
var code = 'USDJPY'; // 通貨の指定
var url = "http://info.finance.yahoo.co.jp/fx/detail/";
// ページの取得
client.fetch(url, {"code":code}, function(err, $, res) {
  if (err) { console.log(err); return; }
  // 値を取得
  var bid = $("#USDJPY_detail_bid").text();
  var ask = $("#USDJPY_detail_ask").text();
  // 結果を表示
  console.log("Bid=" + bid);
  console.log("Ask=" + ask);
});
```

プログラムを実行するには、コマンドラインから以下のコマンドを入力してください。

```
$ node fx-USDJPY.js
Bid=123.587
Ask=123.59
```

せっかくなので、これを改良して、対円の為替情報を一気に取得するようにプログラムを作ってみましょう。ここでは、円と 9 種類の通貨 (米ドル、ユーロ、豪ドルなど) を個別に表示するものを作ってみました。

● file: src/ch06/08-finance/fx-JPYa.js

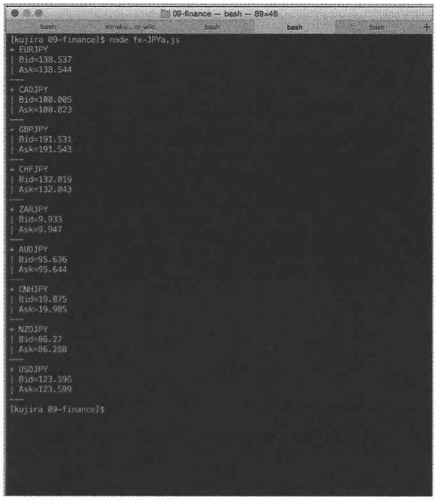
```
// 対円の為替情報を取得 for Node.js

// モジュールの取り込み
var client = require('cheerio-httpcli');

// 基本通貨の指定
var baseCode = "JPY";
var codeList = [
  "JPY", "USD", "EUR", "AUD", "GBP",
  "NZD", "CAD", "CHF", "ZAR", "CNH"
];
// Yahoo! ファイナンスの基本 URL
var baseUrl = "http://info.finance.yahoo.co.jp/fx/detail/";

// 一気にいろんな為替情報を取得
for (var i in codeList) {
  var a = codeList[i];
  if (a == baseCode) continue; // JPYJPY はスキップ
  var code = a + baseCode;
  getFX(code);
}

// ページの取得
function getFX(code) {
  client.fetch(baseUrl, {"code": code}, function(err, $, res) {
    if (err) { console.log(err); return; }
```

```
    // 値を取得
    var bid = $("#" + code + "_detail_bid").text();
    var ask = $("#" + code + "_detail_ask").text();
    // 結果を表示
    console.log("+ " + code);
    console.log("| Bid=" + bid);
    console.log("| Ask=" + ask);
    console.log("---");
  });
}
```

プログラムを実行するには、コマンドラインから以下のコマンドを入力してください。円といろいろな通貨の組み合わせが表示されます。

```
$ node fx-JPYa.js
```

実行すると以下のような結果になります。



円の為替レートを一気に取得するプログラム

株価を取得する

今度は、Yahoo! ファイナンスから株情報を取得してみましょう。為替のときと同じく、特定の URL に証券コードを指定することで、その情報を見られるように工夫されています。証券コードは、上場している会社ごとに割り振られているコードです。例えば、以下の（株）みずほフィナンシャルグループには 8411 というコードが割り振られています。

```
Yahoo! ファイナンス > 株式 > （株）みずほフィナンシャルグループ
http://stocks.finance.yahoo.co.jp/stocks/detail/?code=8411
```

株式情報

この証券コードは、以下のページより一覧を取得できます。ただし、Excel 形式のファイルなので、別途 CSV ファイルなどに変換して使うと良いでしょう。

日本取引所グループ > その他統計資料 > 東証上場銘柄一覧
`http://www.jpx.co.jp/markets/statistics-equities/misc/01.html`

	A	B	C	D	E
1	日付	コード	銘柄名	33業種コード	33業種区分
2	20150529	1301	極洋	50	水産・農林業
3	20150529	1332	日本水産	50	水産・農林業
4	20150529	1333	マルハニチロ	50	水産・農林業
5	20150529	1352	ホウスイ	6050	卸売業
6	20150529	1377	サカタのタネ	50	水産・農林業
7	20150529	1379	ホクト	50	水産・農林業
8	20150529	1414	ショーボンドホールディングス	2050	建設業
9	20150529	1417	ミライト・ホールディングス	2050	建設業
10	20150529	1419	タマホーム	2050	建設業
11	20150529	1420	サンヨーホームズ	2050	建設業
12	20150529	1514	住石ホールディングス	1050	鉱業
13	20150529	1515	日鉄鉱業	1050	鉱業
14	20150529	1518	三井松島産業	1050	鉱業
15	20150529	1605	国際石油開発帝石	1050	鉱業
16	20150529	1606	日本海洋掘削	1050	鉱業
17	20150529	1662	石油資源開発	1050	鉱業
18	20150529	1663	K＆Oエナジーグループ	1050	鉱業
19	20150529	1712	ダイセキ環境ソリューション	2050	建設業
20	20150529	1719	安藤・間	2050	建設業
21	20150529	1720	東急建設	2050	建設業
22	20150529	1721	コムシスホールディングス	2050	建設業
23	20150529	1722	ミサワホーム	2050	建設業
24	20150529	1762	高松コンストラクショングループ	2050	建設業
25	20150529	1766	東建コーポレーション	2050	建設業
26	20150529	1780	ヤマウラ	2050	建設業
27	20150529	1801	大成建設	2050	建設業
28	20150529	1802	大林組	2050	建設業
29	20150529	1803	清水建設	2050	建設業

証券コード一覧

株価を取得するプログラム

それでは、Yahoo! ファイナンスの株式ページから株価を取得するプログラムを作ってみましょう。ここでも、Node.js とモジュール「cheerio-httpcli」を利用します。

● file: src/ch06/08-finance/kabu.js

```
// 為替情報を取得 for Node.js

// モジュールの取り込み
var client = require('cheerio-httpcli');
```

```
// HTMLをダウンロード
var code = '8411'; // 証券コードの指定 ----- (※1)
var url = "http://stocks.finance.yahoo.co.jp/stocks/detail/";
// ページの取得 ---- (※2)
client.fetch(url, {"code":code}, function(err, $, res) {
  if (err) { console.log(err); return; }
  // 値を取得 ---- (※3)
  var price = $("td.stoksPrice").text().replace(/¥s/g, "");
  var name = $("th.symbol > h1").text();
  // 結果を表示
  console.log("+ code=" + code);
  console.log("| name=" + name);
  console.log("| price=" + price);
});
```

プログラムを実行するには、以下のコマンドを入力します。すると、会社名と株価を取得して表示します。

```
$ node kabu.js
+ code=8411
| name=(株)みずほフィナンシャルグループ
| price=263.4
```

プログラムの(※1)では、証券コードを指定しています。この値を書き換えることで、任意の会社の株価をチェックすることができます。

プログラムの(※2)では、HTMLを取得します。(※3)の部分では、HTML中から、CSSセレクタを利用して値を取得します。為替のようにidが指定されていれば便利だったのですが、ここでは、CSSのクラス名「stoksPrice」が指定されていたので、この値を取得しています。また、データを取得した際、前後に空白文字が入っていたので、replace()メソッドを使って空白文字を削除しています。

HTML抽出の小ワザ

HTMLからの任意の部分を抽出するには、cheerioモジュールを使ったCSSセレクタを利用した抽出が簡単です。2章で「cheerio」の使い方を紹介していますので、参考にしてみてください。

この節のまとめ

- この節では為替情報と株価情報を取得する方法を紹介しました。
- いずれも、HTMLファイルを解析する方法での取得となりました。そのため、HTMLの構造が変わってしまうと使えなくなってしまいます。
- しかし、CSSセレクタを書き換えることで、値を取得することができると思います。
- こうした方法でトレードの参考になるデータを上手に収集できます。

09 Wikipediaからのダウンロード

オンライン上で誰でも編集できるフリーな百科事典である、Wikipediaには、日本語版だけで97万項目以上が掲載されています。その膨大なコンテンツをコピーレフトで利用できます。ここでは、ダウンロードの方法を紹介します。

ここで学ぶポイント

- Wikipediaからのダウンロード
- GZipファイルの解凍
- データベース「LevelDB」の活用

ツールやライブラリの一覧

- Node.js
- 「zlib」モジュール

Wikipediaとは何か？

Wikiというシステムによって、誰でも編集できる、フリーな百科事典プロジェクトが、Wikipedia(ウィキペディア)です。米国のNPOであるウィキメディア財団が運営しています。英語、日本語をはじめ150言語以上が使用可能となっています。

本書執筆時点で、97万項目以上が掲載されています。これらの記事をクリエイティブコモンズ(CC-BY-SA)あるいはGFDLというライセンスで利用できます。

なお、参考までにクリエイティブコモンズの(CC-BY-SA)とは、次のようなライセンスをいいます。

ライセンス	説明
共有	どのようなメディアやフォーマットでも資料を複製したり、再配布できます。
翻案	資料をリミックスしたり、改変したり、別の作品のベースにしたりできます。どのような目的でも利用できます。

ただし、著作権表示が必要であり、もし改変した場合には、改変部分を元の作品と同じライセンスで配布する必要があるというものです。それでも、複製・再配布・改変や営利目的での使用までもが可能なので、かなり自由なライセンスであると言えます。

Wikipedia からのダウンロード

Wikipedia では、コンテンツが自由にダウンロードできるように配慮されています。そのため、サイト自体へのクローリングは禁止されています。

名称	URL
Wikipedia 日本語版のデータダウンロード	http://download.wikimedia.org/jawiki/

ただし、Wikipedia の膨大なデータが一つのファイルにまとまっているため、非常にファイルサイズが大きく、普通のテキストエディターなどでは開くことができません。ダウンロードのガイドページにも、その旨が記されています。ガイドページでは、ライセンスや各種データファイルの説明が記されています。

名称	URL
Wikipedia: データベースダウンロード (ガイド)	https://ja.wikipedia.org/wiki/WP:DD

データベースダウンロードのページ

データの形式について

Wikipedia のデータは、XML 形式 (あるいは XML を圧縮した形式) で配布されています。

ファイル名は「jawiki-20150602-abstract.xml」のような形式になっており、「(言語コード)-(日付)-(種類ファイル名)」の形式となっています。次のような種類のファイルが用意されています。

種類ファイル名	説明
pages-articles.xml.bz2	ノートページ、利用者ページを除く最新版のダンプ
pages-meta-current.xml.bz2	全ページの最新版のダンプ
pages-meta-history.xml.7z	全ページの全ての版のダンプ
all-titles-in-ns0.gz	全項目のページ名一覧 (標準名前空間)
abstract.xml.gz	ページの最初の段落とリンクのみを抽出した XML データ
upload.tar	無圧縮で全画像ファイル

Wikipedia から配布されているデータの形式一覧

要約データの利用

なお、ページの最初の段落とリンクのみを抽出したデータ「abstract.xml」をダウンロードする場合には、「jawiki-20150602-abstract.xml」を選んでダウンロードします。日付部分はその時の最新版、あるいは「latest」のものを利用すると良いでしょう。

要約データだけなのに、1.6GB もあるのは驚きます。そのためか、この 1.6GB を複数のファイルに分割したファイル「0150602-abstract1.xml」「0150602-abstract2.xml」「0150602-abstract3.xml」... も提供されています。

これをエディターで見ると、次のような形式となっています。

要約情報の XML

```
<feed>
  <doc>
    <title> タイトル </title>
    <url>Wikipedia への URL</url>
    <abstract> 最初の段落 </abstract>
    <links>
      <sublink linktype="nav">
        <anchor> 名前 </anchor><link> リンク </link>
      </sublink>
      ...
    </links>
  </doc>
  <doc>
    ...
  </doc>
</feed>
```

タイトル一覧の取得

Wikipediaのページタイトルの一覧が提供されています。膨大な項目名のデータを利用することで、文章中の意味のあるキーワードを抽出することが可能となるでしょう。

タイトル一覧は、「all-titles-in-ns0.gz」というファイル名で提供されています。このファイルは、G拡張子「.gz」であり、GZIP形式で圧縮されていることを表しています。

Mac OS X/Linuxのコマンドラインで以下のようにして解凍できます。Windowsでは、Lhacaなどの解凍ソフトを利用して解凍できます。

```
# gz形式を解凍
$ gunzip jawiki-latest-all-titles-in-ns0.gz
```

このファイルは、テキストファイルにタイトルが一件一行で記されています。

タイトル一覧のデータ

Wikipediaのタイトルデータベースを作る

さて、Wikipediaではダウンロードファイルが用意されていましたので、今回は、自動で最新のタイトルデータベースをダウンロードして、データベースに挿入するところまでを作ってみたいと思います。つまり、以下の動作を行います。

(1) **タイトル一覧データをダウンロード**
(2) **GZipファイルを解凍**
(3) **データベースに挿入**

ちなみに、DBには手軽に使える「LevelDB」を利用してみます（LevelDBについては、4章のデータベースの項で詳しく紹介しています）。以下のコマンドを実行して、LevelDBが使えるようにしておきましょう。

```
$npm install level
```

以下がプログラムです。　ダウンロードしたデータを解凍し、LevelDB に挿入します。

● file: src/ch06/09-wikipedia/make-wikipedia-title-db.js

```
// Wikipedia のタイトル DB を作成する for Node.js

// モジュールの利用
var request = require('request');
var fs = require('fs');
var zlib = require('zlib');
var levelup = require('level');

// Wikipedia の最新タイトルデータ --- (※1)
var titleName = "jawiki-latest-all-titles-in-ns0";
var titleUrl = "http://dumps.wikimedia.org/jawiki/latest/" +
      titleName + ".gz";
var local_gz  = __dirname + "/" + titleName + ".gz";
var local_txt = __dirname + "/" + titleName;

// データベースの指定 ---- (※2)
var db = levelup('./wikidata');

// テスト用
titleUrl = "http://localhost/" + titleName + ".gz";

// データをダウンロード ----- (※3)
request
  .get(titleUrl)
  .on('end', goGunzip)
  .pipe(fs.createWriteStream(local_gz));
console.log("ダウンロードします");

// ファイルの解凍 ----- (※4)
function goGunzip() {
  console.log('解凍します');
  // ファイルを読む
  var gz_data = fs.readFileSync(local_gz);
  // 解凍
  zlib.gunzip(gz_data, function (err, bin) {
    if (err) { console.log(err); return; }
    // 結果をファイルに書き込む(念のため)
    fs.writeFileSync(local_txt, bin);
    var txt = bin.toString('utf-8'); // ---(※5)
    insertDB(txt);
  });
}

// LevelDB に結果を書き込む ----- (※6)
function insertDB(txt) {
  console.log("データベースに書き込みます。");
  var lines = txt.split("¥n");
  lines.shift(); // 一行目は捨てる
  var t = db.batch();
  for (var i in lines) {
    var it = lines[i];
    if (it == "") continue;
    t.put(it, 1);
    if (i % 1000 == 0) console.log(i + "件目:" + it); // --- (※7)
  }
  t.write(function() {
    console.log("書き込み完了しました:" + lines.length);
  });
}
```

プログラムを実行するには、以下のコマンドを入力します。

```
$ node make-wikipedia-title-db.js
```

実行すると以下のような結果になります。

Wikipedia のタイトルデータベースを作成

プログラムの (※ 1) では、Wikipedia の最新タイトルデータの URL、また保存ファイル名を指定しています。ファイル名が長いので、ここでは、変数に置き換えて指定しています。

プログラムの (※ 2) では、LevelDB のデータベースオブジェクトを作成しています。「wikipedia」というディレクトリにデータベースを作成します。

プログラムの (※ 3) の部分では、データをダウンロードします。パイプの仕組みを利用してダウンロードをしたファイルを直接ファイルに書き込みますが、今回は、ダウンロードした後で、解凍処理を行いたいので、on('end',...) と記述してダウンロードが完了したタイミングで、関数 goGunzip() が実行されるように指定しています。

プログラムの (※ 4) のでは、ファイルの解凍処理を行います。解凍処理には、zlib.gunzip() メソッドを利用します。ここでは、解凍した結果をそのまま LevelDB に書き込んでいますが、念のためファイルにも解凍した結果を保存しています。ちなみに、zlib.gunzip() で解凍したデータは、バイナリデータとなっています。そのため、プログラムの (※ 5) の部分で UTF-8 に変換し、これをもとにデータベースへの挿入処理を行っています。

Gunzip 解凍を行う、zlib.gunzip() メソッドは以下のように使います。

```
var file_gz = "test.gz";
var file_raw = "test.txt";
// GZIP ファイルを読む
var gz_data = fs.readFileSync(file_gz);
// 解凍する
zlib.gunzip(gz_data, function (err, bin) {
  if (err) { console.log(err); return; }
  // ファイルに書き込む
  fs.writeFileSync(file_raw, bin);
});
```

プログラムの(※6)では、LevelDBへの書き込み処理を行っています。db.batch()メソッドからdb.write()メソッドまでの一連の流れですが、put()メソッドでデータベースにタイトルを書き込みます。このとき、db.batch()メソッドを使わず、db.put()メソッドを使って書き込むことができます。しかし、db.batch()メソッドを使う方が書き込み速度が速くなります。

なお、プログラムの(※7)のところですが、書き込みには時間がかかるので、1000件書き込むごとに、進捗を表示しています。

タイトルデータベースの活用

作成したWikipediaのタイトルデータベースを活用してみましょう。ここでは、「猫」から始まる項目を列挙するプログラムを作ってみます。

● file: src/ch06/09-wikipedia/read-title.js

```
// Wikipediaのタイトルを調べる for Node.js

// モジュールの利用
var levelup = require('level');

// データベースの指定
var db = levelup('./wikidata');

// 検索キーを指定 ----- (※1)
var opt = {
  start : "猫",
  end   : "猫¥uFFFF"
};
// 検索 ------ (※2)
db.createReadStream(opt)
  .on('data', function (data) {
    console.log(data.key);
  })
  .on('end', function () {
    console.log('ok.');
  });
```

プログラムを実行するには、以下のコマンドを入力します。

```
$ node read-title.js
```

すると猫から始まる項目がたくさん列挙されます。

猫から始まる項目を列挙

プログラムの (※1) の部分で、検索キーを指定しています。文字コードの範囲で検索しますので、「猫」から「猫 ¥uFFFF」とすると、猫からはじまる語句を調べることができます。

実際に検索をするのは (※2) の部分です。db.createReadStream() メソッドを使うと、オプションに合致するデータを検索して、data イベントにて、検索結果を通知します。検索が完了するときに、end イベントが実行されます。

正規表現でデータベースを検索

次に正規表現を使ってデータの検索を行ってみましょう。例えば、「『なんとかの女』って歌があったと思うんだけど、なんだったかなぁ。四文字以上だと思うけど・・・」と友達同士で話題になったとき、正規表現を利用して「/.{4,}の女 $/」で検索することができます。

● file: src/ch06/09-wikipedia/read-title-regex.js

```
// Wikipedia のタイトルを調べる for Node.js

// モジュールの利用
var levelup = require('level');

// データベースの指定
var db = levelup('./wikidata');

// 正規表現で項目を検索する ----- (※1)
var search_re = /.{4,}の女 $/;

// すべての項目を検索する
var cnt = 0, result = [];
db.createReadStream()
  .on('data', function (data) {
    // 検索経過の表示
    if (cnt % 50000 == 49999) {
      console.log(cnt+" 件を検索 :" + data.key);
    }
    // 正規表現マッチ ---- (※2)
```

```
    if (search_re.test(data.key)) {
      result.push(data.key);
      console.log("発見:" + data.key);
    }
    cnt++;
  })
  // 最終結果を表示
  .on('end', function () {
    console.log("発見:¥n" + result.join("¥n"));
    console.log('ok.');
  });
```

コマンドラインから以下を実行します。

```
$ node read-title-regex.js
```

正規表現で検索するためには、すべてのキーを巡回して、合致するデータを探す必要があります。そのため、検索にかなり時間がかかります。I 今回は、以下のような 79 件の結果を得ることができました。

```
がちょう番の女
さそり座の女
たまゆらの女
ぬかるみの女
ふたりの男とひとりの女
アンクルの女
エイリアス _2 重スパイの女
カフェ・タンブランの女
...(後略)
```

プログラム (※ 1) で、正規表現のパターンを指定し、(※ 2) の部分でマッチさせています。このとき、createReadStream() メソッドでオプションを何も指定しないと、すべてのキーについて調べることができます。

この節のまとめ

- Wikipedia はデータが豊富でライセンス的にも自由なので、うまく使いこなせば、さまざまな用途に利用できることでしょう。
- 本文ではページタイトル一覧のデータベースも作成してみました。ページタイトルを複雑な条件で調べるだけでも楽しいものでした。
- これらのタイトルを、文章中のキーワード一覧辞書とすることも可能でしょう。活用してみてください。

第７章

データの分類と予測と機械学習

本章ではデータの活用方法として、データの分類や予測などに関連する技術を紹介します。その中で人工無能（会話ボット）や機械学習を用いた文字認識などのプログラムを作ってみます。

01

データの活用法について

クロールして得たデータを活用するときに、考えられるのが「予測」と「分類」と「関連」です。この節では、データを幅広く活用する方法について考察していきます。そして次の節以降で実際にデータを活用する方法を紹介します。

ここで学ぶポイント	ツールやライブラリの一覧
●データマイングについて ●データの活用について	●なし

データをどのように活用するか？

7 章では、収集したデータをどのように活用していくのかを紹介します。データの活用で参考になるのが「データマイニング」の手法です。はじめに、データマイニングについて考察します。

データマイニングしよう！

「データマイニング（Data mining）」とは、大量のデータを調べて、そこから価値のある情報を抽出することを言います。特に今まで知られていなかった情報を、大量のデータの中から抽出すること、またその技術体系を指しています。データを調べる方法には、統計学、パターン認識、人工知能など、データ解析の技法を利用します。

そもそも、「マイニング（mining）」とは「採鉱」という意味です。広い鉱山から隠された金脈を採掘するのと同じように、大量のデータを解析することで価値のある情報を見つけることが、データマイニングです。

データマイニングの中でとくにテキストを対象とするものを「テキストマイニング」、Web ページを対象にしたものを「Web マイニング」と呼びます。

データマイニングの基本は「予測」「分類」「関連」

データマイニングが難しいのは、ただデータを入れれば結果が出るというものではないという点にあります。大量のデータの中から知識の発見を行う必要があります。そのために必要なのが、仮説を立てることです。マーケティングで言えば、その仮説は「予測」「分類」「関連」から考えることができます。

「予測」について

そもそも「予測」とは、将来の出来事や有様を何らかの根拠に立って推し測ることです。似た言葉に「予想」という語がありますが、予測が予想と違うのは、それが当てずっぽうではなく、何かしらの根拠に基づいているという点です。

例えば、その週に台風が来るかどうかを知りたい時、適当に「なんとなく台風が来そうだ」と言うのは「予想」であり、過去数十年の台風情報を元にして、「毎年、この週には 20% の確率で台風が来ているので、今週は台風が来る確率が高い」と考えるのは「予測」と言えます。一般的に言って、予測をするときには、その事象が発生する確率を算出することになります。

「分類」について

「分類」とは、複数の事物や現象を、何らかの基準に従って区分することによって体系づけることを言います。分類をすることで、情報を整理することができ、物事を把握しやすくなり多くのメリットがあります。

例えば、図書館における蔵書の整理を考えてみましょう。膨大な蔵書を分類し整理することで、利便性があがり読者は本を見つけやすくなります。家庭で衣類の分類をする場合にも、夏服と冬服、色や形で分けておくことで、着たい服をすぐに見つけることができますし、服の組み合わせを考えるのも容易になります。また、夏には冬服を奥にしまっておくなど効率的にスペースを使うこともできます。

正しく分類を行うことで、さまざまな仮説を考えることが可能になります。例えば、目の前に漠然とした顧客データがあるとします。どんな分類ができるでしょうか。

- **男性と女性に分ける**
- **２０代、３０代、４０代など年代で分ける**
- **年収で分ける**
- **職種で分ける**
- **購入した商品の色で分ける**
- **購入した金額で分ける**

このように、同じ顧客データにしても、さまざまな分類が可能です。違う分類を組み合わせて赤色の商品を買う人は、年収が高く、当社の高級な商品をよく買っているなどと、仮説を立てることができます。

「関連」について

「関連」とは、ある事柄と他の事柄との間につながりがあることを言います。データベースに蓄積されたデータを調べることで、頻繁に同時に起きる事象を見つけることができる可能性があります。

本書の冒頭でも紹介したように、スーパーの販売データを調べたところ、オムツを買う人はビールも同時に購入する、雨の日は肉の売り上げが上がるなど、データを調べることで思わぬ発見があります。こうした情報を活かしていくことで、売り上げを伸ばすことができます。

データマイニングの手順

データマイニング（ここでは、特にテキストを解析するテキストマイニング）のやり方の手順をまとめると、以下のようになります。

(1) **対象データを収集**
(2) **形態素解析などしてデータを分割**
(3) **データをクリーニング (外れ値、欠損値、ノイズ除去)**
(4) **データを要約 (次元縮約、属性選択など)**
(5) **データマイニング (統計や他のデータと組み合わせるなど)**
(6) **評価と検証**

(1) 対象データを収集したら、(2) 形態素解析などして、日本語を利用しやすいように最小の単位（形態素）に分割します。その後、(3) クリーニングをして外れ値や欠損値、ノイズなどを除去します。

そして、(4) どのように活用するかを考慮してデータを要約します。これは、例えば、数値化したり、意味のある集合でまとめるなどします。それから、(5) 統計や他のデータと組み合わせたりして、データマイニングを行います。最後に、(6) 評価や検証を行います。

代表的なデータマイニング手法

データマイニングを行う際には、以下のような手法が考えられます。

(1) **相関ルール：X が起きるときには Y も起きやすい**
(2) **回帰分析：X の属性から，数値変数 Y を予測**
(3) **クラス分類：X の属性から，そのクラス C を予測**
(4) **クラスタリング：似ているもの同士をまとめる**

もう少し、詳しく見ていきましょう。(1) 相関ルール抽出は、アソシエーションルール抽出とも言います。代表的な用途は、小売業の POS システムです。例えば、コンビニで「パンと野菜ジュースを買う人は、ヨーグルトも一緒に買う」などの分析を行うことを言います。これは、買い物バスケットの中身の組み合わせを漏れなく抽出し、その中から興味深い結果を探し出すことを目的としています。

(2) 回帰分析は、相関関係や因果関係があると思われる 2 つの変数のうち、一方の変数から将来的な値を予測するための予測式を求めるための手法です。データの傾向を分析することで、予測を行うことができます。

(3) クラス分類とは、特定の分類法をより正確に再現するモデルを作成することが目的です。方程式や、ルール、確率、マッチングなど、さまざまな手法が考えられます。

(4) クラスタリングは、データの集合を部分集合（クラスタ）に切り分けて、それぞれの部分集合に含まれるデータがある共通の特徴を持つようにすることです。この特徴は多くの場合、類似性や、ある定められた距離尺度に基づく近さで示されるものです。

この節のまとめ

- **データマイニングの方法や手順をまとめてみました。**
- **この節では、概念的な話が中心でしたが、次節以降で実際のプログラミングを見れば、理解を深めることができるでしょう。**

02 ベイジアンフィルタによる分類

ここでは、ベイジアンフィルタによる分類について解説します。ナイーブベイズ分類を利用して、自動的に文章を任意のカテゴリーに分類するプログラムを作ってみましょう。難しいように思えますが、実は簡単に自動分類を行うことができます。

ここで学ぶポイント	ツールやライブラリの一覧
●ベイジアンフィルタによる分類	●Node.js ●「bayes」モジュール ●「mecab-lite」モジュール

ベイジアンフィルタとは

ベイジアンフィルタ (Bayesian Filter) は、「ナイーブベイズ分類」を応用したもので、対象となるデータを解析・学習し分類するためのフィルタです。身近なところでは、迷惑メールを識別するフィルタの仕組みとして広く知られている機械学習処理のアルゴリズムです。学習量が増えるとフィルタの分類精度が上昇するという特徴を持っています。

例えば、迷惑メールかどうかを判定するプログラムは、迷惑メールに使われている語彙を学習し、新たに来たメールでどのような語彙が使われているかを調べて、迷惑メールかどうかを判定します。もし、判定を間違えた場合には、ユーザーが迷惑メールであることを学習させることで、より精度が高くなります。迷惑メールかどうかを判定する部分に使われているアルゴリズムが、ベイジアンフィルタです。

従来型のキーワード指定によるフィルタでは、ユーザーが一つ一つNGワードを入力することで、迷惑メールを判定していました。しかし、ベイジアンフィルタは、対象データの内容をフィルタが学習して自動的に分類するため、ユーザーが煩雑なキーワード指定を行う必要がありません。

このように、迷惑メールの判定で有名になった「ベイジアンフィルタ」ですが、他にも文章のカテゴリー分けに利用することが可能です。

ナイーブベイズ分類のアルゴリズム

それでは、ベイジアンフィルタ、つまり、ナイーブベイズ分類 (Naive Bayes classifier) のアルゴリズムについて簡単に紹介します。

ナイーブベイズ分類器というのは、ベイズの定理を利用した分類手法です。ベイズの定理というのは、次のようなものです。

ベイズの定理

入力 X が与えられたときに、出力 Y が得られる確率を、P(Y|X) と表します。そして、これをベイズの定理で表すと以下のようになります。

$$P(Y|X) = \frac{P(Y)P(X|Y)}{P(X)}$$

このとき、X には何らかの起こった事象や入力データを、Y にはそこから推論したい事柄などを当てはめます。例えば、入力 X は入力テキストになり、出力 Y は決定されるカテゴリーであるということになりますし、X が受信したメールであれば、Y は「迷惑メール」か「普通メール」のようになります。

X や Y ではわかりにくいので、具体的な例で考えていきましょう。例えば、迷惑メール (以後、スパムと略す) について考えると、ベイズの定理がわかりやすくなります。

ここでは、入力テキスト X を、メールの特徴 F とし、Y をスパムである確率と考えましょう。すると、少しベイズの定理が読みやすくなります。

$$P(\text{スパム} \mid \text{メールの特徴 F}) = \frac{P(\text{スパム}) \times P(\text{メールの特徴 F} \mid \text{スパム})}{P(\text{メールの特徴 F})}$$

もう少し詳しく見ていきましょう。P(スパム) というのは、あるメールがスパムである確率です。メールの特徴 F を観察する前の確率を表しているため、事前確率です。そのため、P(スパム) の決め方としては、それまでに学習したメールのうち、何割がスパムメールであったかを指定することができるでしょう。もし、事前に何も情報がなければ、50% を指定することもできます。

そして、P(スパム | メールの特徴 F) とは、ここで求めたい結果、つまり、あるメールの特徴 F を見たとき、そのメールがスパムである確率を言います。こちらは、事後確率となります。

次に、P(メールの特徴 F| スパム) ですが、これは、スパムにメールの特徴 F が現れる確率です。どの程度、スパムメールの特徴を持っているのか、もっともらしさを表します。統計の用語で「尤度（ゆうど）」と呼びます。

最後に、P(メールの特徴 F) は、スパムメールを含むすべてのメールの中で、メールの特徴 F を持ったメールが出現する確率です。

ナイーブベイズ分類について

ナイーブベイズ分類では、ある文章をカテゴリー分けする際に、テキスト中の単語の出現率を調べます。その際、その文章をどのカテゴリーに分類するのが相応しいかを調べるものです。テキストを単語に分割し、これを利用して調べる方法を「bag-of-words(単語が詰まった袋)」と呼びます。

こおでは、スパムフィルターではなく、X を入力テキスト、Y を決定されるカテゴリーとして話を進めましょう。

```
X = 入力テキスト
Y = 決定されるカテゴリー
```

カテゴリーを推定する場合、P(Y|X) の値は、一つの確率の値ではなく、複数のカテゴリーの中で、どのカテゴリーになる確率が一番高いかを表す情報となります。このとき、ベイズ定理の分母である P(X) は、入力テキストが与えられる確率となりますが、どのカテゴリーでも同じ値になると考えて、考慮しないことにします。すると、ナイーブベイズ分類の式は、次のように簡単になります。

```
P(Y|X) = P(Y)P(X|Y)
```

ここで、入力テキストである X を各単語 x の集合と考えると、次の式に直すことができます。

```
P(X|Y) = P(x1|Y) P(x2|Y) P(x3|Y) ... P(xN|Y)
```

すると、P(xN|Y) の確率は、分割した単語があるカテゴリーに属する確率を求めることとなります。ですから、あるカテゴリーにその単語が出現した確率を求めます。これは、次の計算で求められるでしょう。

```
単語の出現確率 = 単語の出現回数 / カテゴリーの全単語数
```

つまり、入力テキストを学習させて訓練させる時には、出現した単語が各カテゴリーに分類された回数を数えておけば良いことになります。実際にカテゴリー分けの推定を行う時には、P(Y)P(X|Y) をカテゴリーごとに計算する事になります。

ベイジアンフィルタのライブラリを使おう

さて、だいたいベイジアンフィルタの仕組みがわかったでしょうか。よくわからないという方もいることでしょうが、ベイジアンフィルタのプログラムは比較的単純ですし、ベイズの定理がわからなくても、そのライブラリを使うことは容易です。

なぜかというと、Node.js のモジュールの中に、ずばりそのもののベイジアンフィルタを実装した「bayes」モジュールがあり、とても簡単に利用することができるからです。はじめに、このモジュールを利用して、その使い方や仕組みに慣れていきたいと思います。まずは、モジュールをインストールして使ってみましょう。

```
$ npm install bayes
```

また、本書の5章で作ったMeCabを扱うモジュールを「mecab-lite」という名前でnpmに登録してみました。そこで、このライブラリもインストールしておきましょう。「mecab-lite」では、同期的に形態素解析を行うparseSync()メソッドや分かち書きの結果だけを取得するwakatigakiSync()メソッドなども作っておきました。

```
$ npm install mecab-lite
```

ベイジアンフィルタで「信長」と「漱石」を判定

さて、モジュールがインストールできたら、ベイジアンフィルタで遊んでみましょう。ここでは、Wikipediaから織田信長と夏目漱石に関する記述をベイジアンフィルタに学習させてみました。学習させた結果、「新しく読み込んだ文章が漱石」と「信長」どちらに関する記述かを類推させるというものになっています。

● file: src/ch07/02-bayes/simple-bayes.js

```
// 簡単な文章の分類分け for Node.js

// モジュールの取り込み
var bayes = require('bayes');
var Mecab = require('mecab-lite'),
    mecab = new Mecab();

// サンプルのテキスト
var t_soseki = '夏目漱石は、日本の小説家、評論家、英文学者。江戸の牛込馬場下横町出身。
中略
などを書く。';
var t_nobunaga = '織田信長は、戦国時代から安土桃山時代にかけての武将・戦国大名。三英傑
中略
がら領土を拡大した。';

// テキストの分割方法を設定 ---- (※1)
var classifier = bayes({
  tokenizer: function (text) {
    return mecab.wakatigakiSync(text);
  }
});

// テキストを学習させる --- (※2)
classifier.learn(t_soseki, '夏目漱石');
classifier.learn(t_nobunaga, '織田信長');

// いざ、判定させよう! --- (※3)
categorize('明治の人気小説家。猫に関する物語でも有名。');
categorize('尾張の戦国武将。');
categorize('戦国時代に領土を広げた。');

// カテゴリー分けした上で結果をわかりやすく表示 ---- (※4)
function categorize(text) {
  var r = classifier.categorize(text);
  console.log("カテゴリー=[" + r + "] - " + text);
}
```

このプログラムを実行するには、以下のコマンドを打ち込みます。

```
$ node simple-bayes.js
```

実行すると、ベイジアンフィルタに漱石と信長に関するテキストを学習させ、三つの新たなテキストが、どちらのタイプに属するかを判定させた結果が表示されます。

ベイジアンフィルタの利用例

見やすく結果を抜粋すると以下のようになります。

カテゴリー =[夏目漱石] - 明治の人気小説家。猫に関する物語でも有名。
カテゴリー =[織田信長] - 尾張の戦国武将。
カテゴリー =[織田信長] - 戦国時代に領土を広げた。

うまく分類できました。ほんの少ししか学習させてないので、いろいろ試すと間違えてしまうかもしれません。当然ですが、学習させたテキストに登場しないキーワードを持つ文章を指定すると判定ミスをしてしまいます。しかし、間違えたら学習させれば良いのが、ベイジアンフィルタの良いところです。

プログラムを見てみましょう。ここでは「bayes」モジュールを利用してベイジアンフィルタを実現しています。

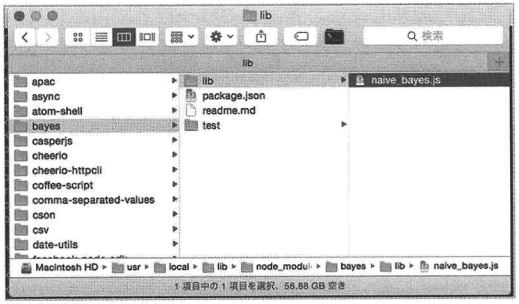
プログラムの (※ 1) では、「bayes」モジュールを初期化している部分ですが、ここで、テキストの分割方法を設定しています。英語ではスペースで単語を区切るだけで良いのですが、日本語では、形態素解析を行う必要があります。そこで、「mecab-lite」モジュールの wakatigakiSync() メソッドを利用して、文章を分割するように指定しています。

プログラムの (※ 2) では、夏目漱石と織田信長に関する記述を学習させます。learn() メソッドを使って学習させます。引数には、学習対象のテキスト、分類カテゴリーの名前の順で指定します。学習対象の説明文の語彙だけに注目してみると、ずいぶん異なる語彙が使われているので、良い感じに学習してくれそうだと期待できるでしょう。

プログラムの (※ 3) では、三つのテキストを指定してカテゴリー分けに挑戦しています。プログラムの (※ 4) では、classifier.categorize() メソッドで、カテゴリーの判定を行っています。

「bayes」モジュールのコードを読む

ここで、「bayes」モジュールのコードを読んでみましょう。npm でインストールした場合、このライブラリの本体は、「node_modules/bayes/lib/naive_bayes.js」になります。



ライブラリを読んでみよう

このファイルを見てみると、なんと 300 行以内で書かれています。非常にシンプルなコードでベイジアンフィルタを実装できることがわかります。

テキストを学習している部分が、learn() メソッドの前半部分です。英語の部分のコメントを日本語に直しています。カテゴリーの初期化をしたり単語を分割したりしています。

```
Naivebayes.prototype.learn = function (text, category) {
  // 初出のカテゴリーであれば初期化する
  self.initializeCategory(category)
  // そのカテゴリーでいくつのドキュメントを学習したか記録
  self.docCount[category]++
  // トータルで学習したドキュメントの数を記録
  self.totalDocuments++
  // 単語に分割する
  var tokens = self.tokenizer(text)
  ...
```

そして、lean() メソッドの後半では、テキスト中の各単語について出現回数を加算しています。ここでは、ひたすたカテゴリーの中の単語を加算しています。カウントしているのは、カテゴリー中の単語出現回数 (wordFrequencyCount) と、カテゴリーの総単語数 (wordCount) です。

```
  ...
  // 単語の出現回数を数えるメソッドを実行
  var frequencyTable = self.frequencyTable(tokens)

  /* カテゴリー中のボキャブラリーとカテゴリーの単語の出現回数を更新 */
  Object
  .keys(frequencyTable)
  .forEach(function (token) {
    // 各単語ごとに以下の処理を行う
    if (!self.vocabulary[token]) { // ボキャブラリーを更新
      self.vocabulary[token] = true
      self.vocabularySize++
    }
    var frequencyInText = frequencyTable[token]

    // カテゴリーの中でこの単語が出現した回数を加算
    if (!self.wordFrequencyCount[category][token])
      self.wordFrequencyCount[category][token] = frequencyInText
    else
      self.wordFrequencyCount[category][token] += frequencyInText

    // このカテゴリーの総単語数を加算
    self.wordCount[category] += frequencyInText
  })
  return self
}
```

続いて、判定を行っている部分 categorize() メソッドを見てみましょう。各カテゴリーごとに可能性を調べています。その方法としては、テキスト内に含まれる単語について、各単語の確率を加算するという方法です。非常にシンプルですね。

```
Naivebayes.prototype.categorize = function (text) {
  ...
  // テキストを分割して、単語の出現回数を数えるメソッドを実行
  var tokens = self.tokenizer(text)
  var frequencyTable = self.frequencyTable(tokens)

  // 各カテゴリーごとに確率を求める
  Object
  .keys(self.categories)
  .forEach(function (category) {
    // はじめに、このカテゴリーの確率を求める（いくつの文書が分類されているかを考慮）
    var categoryProbability = self.docCount[category] / self.totalDocuments
    // アンダーフローを起こさないよう log() を計算
    var logProbability = Math.log(categoryProbability)

    // 今回出現した各単語を調べて、P(W|C) を決定する
    Object
    .keys(frequencyTable)
    .forEach(function (token) {
      var frequencyInText = frequencyTable[token]
      // 単語ごとの確率を加算していく
      var tokenProbability = self.tokenProbability(token, category)
      logProbability += frequencyInText * Math.log(tokenProbability)
    })

    // これまでに登場した最も高い確率かどうかを調べる
    if (logProbability > maxProbability) {
      maxProbability = logProbability
      chosenCategory = category
    }
  })
  return chosenCategory
}
```

この節のまとめ

- この節では、ベイジアンフィルタを利用したプログラムと、その仕組みについて紹介しました。
- 数式が出てくると難しく感じますが、判定の仕組み自体はシンプルなものでした。
- 「bayes」モジュールを使うだけなら比較的簡単に文書の分類ができます。
- 迷惑メール判定以外にもいろいろな分類ができるので、幅広い用途が考えられます。

03

移動平均を利用した予測とグラフの描画

需要予測とは物の需要を予測することです。需要予測にはさまざまな手法があり、状況に適した方法を選択したり、複数の手法を試したりすることができます。ここでは簡単に計算が可能な、移動平均法と指数平滑法を解説します。

ここで学ぶポイント	ツールやライブラリの一覧
●移動平均法について ●Google Chartsを利用したグラフの描画	●Node.js ●Google Charts

需要予測について

ビジネスにおいては、需要を予測することで在庫削減や欠品削減に大きな効果が得られます。需要予測は実際の業務に役立つノウハウです。計算もそれほど難しくないので、一度理解してしまえば、さまざまな所で利用することができるでしょう。膨大なデータの中からデータマイニングを行う際にもこの手法が役立ちます。

しかし、需要予測をはじめ、為替や株の値動きの予測を行う際によく言われるのが「予測はあくまでも予測なので現実にはその通りにはならない」ということです。現実の社会は複雑で完全な予測はできないものです。それは、どんなに複雑なアルゴリズムを利用しても同じです。ですから、利用する際には、予測の誤差を考慮する必要があります。それでも、企業は需要予測を行うことで生産計画や販売キャンペーンを策定したり、設備投資や採用計画を行うことになります。需要予測によって、企業は、無駄な在庫を抱えたり販売機会を逃すことが少なくなることが期待されます。勘や気分で計画を立てるよりも、それまでの実績を元に予測を行い、それに基づいて計画を立てる方が理にかなっているからです。

さて、需要予測にはいろいろな手法がありますが、最も広く利用されているのは、移動平均法 (Moving Average) と指数平滑法 (Exponential Moving Average; EMA) です。

単純移動平均について

「単純移動平均 (Simple Moving Average; SMA)」とは、時系列データを平滑化する手法です。金融や気象などの計測分野で使われています。直近の n 個のデータに対して単純な平均を取る方法です。

例えば、定期的な英語のテストを行ったとして、点数が今日は 60 点、前回は 46 点、その前は 53 点だったとします。すると、その平均点は、(60+46+53)/3=53 という式で求めることができます。

直近 n 個の各データを、Pm, Pm-1, Pm-2, Pm-3... で表すと次のようになります。

```
        Pm + Pm-1 + Pm-2 + Pm-3 ...
SMAm = ----------------------------
                    n
```

しかし、直近 100 個のデータの総計を，データの個数 100 で割って 1 つの値を得るならば、データ全体が増加傾向にあるのか、減少傾向にあるのかを知ることができません。平均はあくまでも平均であり、貴重な情報が失われていることになります。「単純移動平均」はグラフに描画したときに上下するギザギザしたデータを、なめらかなグラフになるようデータを変換する手法です。

グラフを描画してみよう

ここで、移動平均を計算する前に、普通に折れ線グラフ（ラインチャート）を描画する方法を紹介します。JavaScript でグラフを描画する場合には、素直に HTML と Web ブラウザー、そして気の利いたビジュアライゼーションのためのライブラリを使うのが良いでしょう。ここでは、Google Charts を利用してみます。Google Charts は使い勝手もよく堅実なグラフを描画するのに最適なライブラリです。詳しくは、8 章で扱いますが、予測傾向を把握するために線グラフを描画してみましょう。

```
Google Charts
https://google-developers.appspot.com/chart/
```

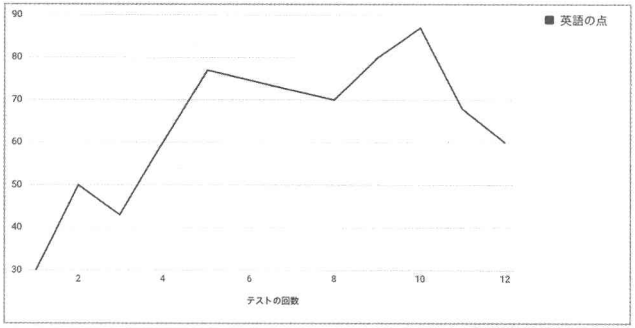
簡単に折れ線線グラフを描画してみましょう。以下は、Google Charts を利用して、英語の点数を折れ線グラフで表示したものです。

● file: src/ch07/03-yosoku/linechart.html

```
<html><head>
  <!-- Google Charts の取り込み -->
  <script type="text/javascript" src="https://www.google.com/jsapi"></script>
  <!-- 気温データの読込 -->
  <script type="text/javascript" src="2014kion.js"></script>
  <script type="text/javascript">
    // Charts API の初期化
    google.load('visualization', '1.1', {packages: ['line']});
    google.setOnLoadCallback(drawChart);
    // 実際の描画
    function drawChart() {
      // データオブジェクトを作成
      var data = new google.visualization.DataTable();
      // データのカラムを指定 ----- (※1)
      data.addColumn('number', 'テストの回数');
      data.addColumn('number', '英語の点');
      // データの値を指定 [テストの回数, 英語の点] ---- (※2)
      data.addRows([
        [1, 30],[2, 50],[3, 43],[4, 60],[5, 77],
```

```
        [8, 70],[9, 80],[10,87],[11,68],[12,60]
    ]);

後略
```

HTML5 に対応した Web ブラウザーに、HTML ファイルをドラッグ＆ドロップすると、以下のように描画が行われます。



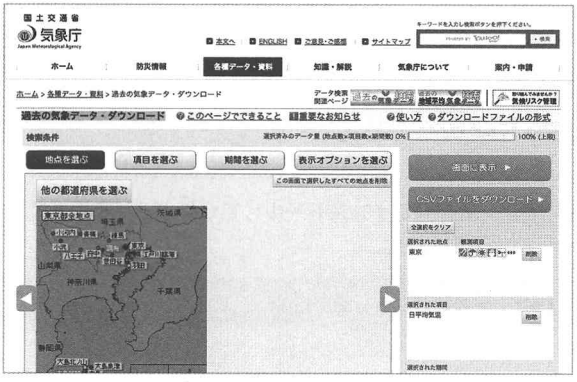
簡単な折れ線グラフを描画した

基本的には、Google Charts のラインチャートに関する描画処理の指定を行ってます。プログラムの (※ 1) と (※ 2) の部分を少し変更するだけで、任意の折れ線グラフを描画することができます。(※ 1) の部分では、カラム数を指定します。そして、(※ 2) で実際のデータを二次元配列で指定しています。

過去の気象データのダウンロード

2014 年の平均気温をグラフに描画してみましょう。気象庁の過去の気象データ・ダウンロードのページから平均気温をダウンロードしてください。CSV 形式でダウンロードできます。

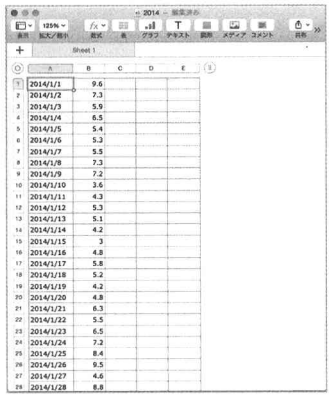
過去の気象データ・ダウンロード
`http://www.data.jma.go.jp/gmd/risk/obsdl/index.php`



過去の気象情報をダウンロード

CSV 形式でダウンロードできるとは言え、そのまま利用できる形式ではないので、Excel など表計算ソフトで読み込んで、「日付」「平均気温」だけの情報に整形しましょう。通常、CSV ファイルの文字コードは

SHIFT_JIS で保存しますが、ここでは日付と気温だけなので、JavaScript で扱いやすいよう UTF-8 で保存します。



	A	B	C	D	E
1	2014/1/1	9.6			
2	2014/1/2	7.3			
3	2014/1/3	5.9			
4	2014/1/4	6.5			
5	2014/1/5	5.4			
6	2014/1/6	5.3			
7	2014/1/7	5.5			
8	2014/1/8	7.3			
9	2014/1/9	7.2			
10	2014/1/10	3.6			
11	2014/1/11	4.3			
12	2014/1/12	5.3			
13	2014/1/13	5.1			
14	2014/1/14	4.2			
15	2014/1/15	3			
16	2014/1/16	4.8			
17	2014/1/17	5.8			
18	2014/1/18	5.2			
19	2014/1/19	4.2			
20	2014/1/20	4.8			
21	2014/1/21	6.3			
22	2014/1/22	5.5			
23	2014/1/23	6.5			
24	2014/1/24	7.2			
25	2014/1/25	8.4			
26	2014/1/26	9.5			
27	2014/1/27	4.6			
28	2014/1/28	8.8			

表計算ソフトで日付と気温だけに編集

その上で、CSV ファイルを Google Charts 用の二次元配列データに変換して、グラフで表示してみましょう。ここでは、CoffeeScript を利用して、CSV ファイルを JSON 形式に変換するプログラムを作ります。

● file: src/ch07/03-yosoku/csv2js-2014kion.coffee

```
# Node.jsでCSVをGoogle Charts用のJSに変換

FILE_CSV = './2014kion.csv';
FILE_JS  = './2014kion.js';

fs = require 'fs'

txt = fs.readFileSync FILE_CSV, "utf-8"
lines = txt.split "\r\n"

result = [];
for v in lines
  cells = v.split ','
  date_s = cells[0].split("/").splice(1,2).join("/");
  temp   = parseFloat cells[1]
  result.push([date_s, temp])

json = JSON.stringify result
js = "var kion_data = " + json;
fs.writeFileSync FILE_JS, js, "utf-8"
console.log "ok"
```

プログラムを実行するには、CoffeeScript が必要です。4 章を見てインストールしてください。その上で以下のコマンドを実行して、CSV を JavaScript のデータに変換します。

```
$ coffee csv2js-2014kion.coffee
```

すると、以下のようなデータが「2014kion.js」という名前で出力されます。(実際は改行がありません)

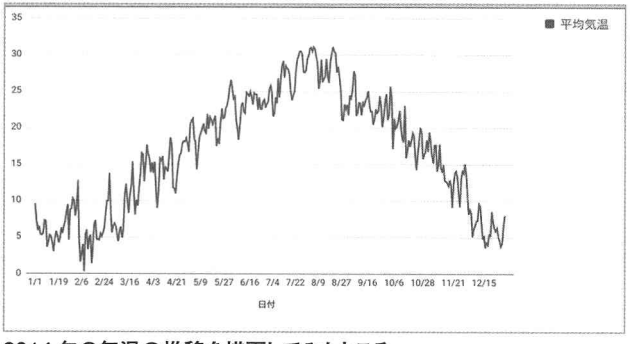
```
var kion_data = [
  ["1/1",9.6],
  ["1/2",7.3],
  ["1/3",5.9],
  ["1/4",6.5],
  ["1/5",5.4],
  ["1/6",5.3],
  ["1/7",5.5],
  ...
]
```

データができたら、これをチャート上に表示してみます。以下がチャートを読み込んで表示するプログラムです。

● file: src/ch07/03-yosoku/lc-2014kion.html

```
<html><head>
  <!-- Google Chartsの取り込み -->
  <script type="text/javascript" src="https://www.google.com/jsapi"></script>
  <!-- 気温データの読込 -->
  <script type="text/javascript" src="2014kion.js"></script>
  <script type="text/javascript">
    // Charts APIの初期化
    google.load('visualization', '1.1', {packages: ['line']});
    google.setOnLoadCallback(drawChart);
    // 実際の描画
    function drawChart() {
      // データオブジェクトを作成
      var data = new google.visualization.DataTable();
      // データのカラムを指定 ----- (※1)
      data.addColumn('string', '日付');
      data.addColumn('number', '平均気温');
      // 値を設定 ---- (※2)
      data.addRows(kion_data);

後略
```

HTMLファイルをWebブラウザーで表示してみましょう。



2014年の気温の推移を描画してみたところ

このようにして、一年の気温の推移をグラフにすることができました。冬は寒く夏は暑いという様子を見て取ることができます。また、グラフのギザギザが大きく、一日ごとの気温差が大きいということもわかります。

プログラム自体は、前回の英語のテストのグラフを作った時と、それほど違いありません。違うところは、グラフに描画するデータを、外部の JavaScript ファイル「2014kion.js」から読み込んでいる点です。プログラムの (※ 1) では、データのカラムを日付と平均気温と設定し、(※ 2) では外部で定義されている気温データを設定しています。

気温の移動平均を計算してグラフに描画

それでは、毎日の気温の移動平均を計算してグラフに描画してみましょう。
先ほど 2014 年の平均気温を記録した CSV ファイルを元にして、移動平均を加えた JavaScript のデータを計算し挿入してみます。まずは、JavaScript のデータを作成する CoffeeScript のプログラムです。

● file: src/ch07/03-yosoku/csv2js-2014kion-sma.coffee

```
# Node.js で CSV を Google Charts 用の JS に変換
# 移動平均も計算する

FILE_CSV = './2014kion.csv'
FILE_JS  = './2014kion-sma.js'
MA_RANGE = 7

fs = require 'fs'

# ファイルから CSV ファイルを読む
loadCSV = (filename) ->
  txt = fs.readFileSync filename, "utf-8"
  lines = txt.split "\r\n"
  # CSV テキストを二次元配列変数に変換
  list = []
  for v in lines
    cells = v.split ','
    date_s = cells[0].split("/").slice(1,3).join("/")
    temp   = parseFloat cells[1]
    list.push([date_s, temp])
  return list

# 移動平均を計算 ------ ( ※ 1)
calcMovingAverage = (i, list, range) ->
  # 期間を決定
  m_from = i - range
  m_to   = m_from + range - 1
  # 期間が不完全ならば 0 を返す
  if m_from < 0 then return NaN
  # 合計を計算
  sum = 0
  for j in [m_from .. m_to]
    sum += list[j][1]
  # 平均を計算
  return sum / range

# メイン処理
main = ->
  # CSV を読み込む
  list = loadCSV(FILE_CSV)
```

```
  # 各行について移動平均を求めリストに追加 --- (※2)
  for val, index in list
    av = calcMovingAverage(index, list, MA_RANGE)
    list[index].push(av)
  # JavaScriptを出力
  js = "var kion_data = " + JSON.stringify(list)
  fs.writeFileSync FILE_JS, js, "utf-8"

main()
console.log "ok"
```

プログラムを実行するには、以下のコマンドを入力します。

```
$ coffee csv2js-2014kion-sma.coffee
```

すると「2014kion-sma.js」というJavaScriptのデータが作成されます(実際は改行がありません)。

```
var kion_data = [
  ["1/1",9.6,0],
  ["1/2",7.3,0],
  ...
  ["1/8",7.3,6.499999999999999],
  ["1/9",7.2,6.171428571428572],
  ["1/10",3.6,6.1571428571428575],
  ["1/11",4.3,5.828571428571429],
  ...
]
```

プログラムで主に注目したいのは、移動平均を計算している(※1)の部分でしょう。ここでは過去7日間の平均値を求めて返すという処理になります。そして、(※2)の部分で、CSVの各行ごとに移動平均を求め、移動平均の列を追加するという処理になっています。

それでは、これをグラフに描画してみましょう。以下がグラフを表示するHTMLのコードです。こちらは、ほぼ前回と同じです。異なるのは表示するカラム「移動平均」が増えているという部分です。

● file: src/ch07/03-yosoku/lc-2014kion-sma.html

```
<html><head>
  <!-- Google Chartsの取り込み -->
  <script type="text/javascript" src="https://www.google.com/jsapi"></script>
  <!-- 気温データの読込 -->
  <script type="text/javascript" src="2014kion-sma.js"></script>
  <script type="text/javascript">
    // Charts APIの初期化
    google.load('visualization', '1.1', {packages: ['line']});
    google.setOnLoadCallback(drawChart);
    // 実際の描画
    function drawChart() {
      // データオブジェクトを作成
      var data = new google.visualization.DataTable();
      // データのカラムを指定 ----- (※1)
      data.addColumn('string', '日付');
      data.addColumn('number', '平均気温');
      data.addColumn('number', '移動平均');
      // 値を設定 ---- (※2)
      data.addRows(kion_data);
```

```
後略
```

Web ブラウザーに「lc-2014kion-sma.html」をドラッグ＆ドロップして表示してみましょう。

2014 年の気温と移動平均を表示したところ

無事に移動平均が表示されました。このように、実際の気温に推移に比べて、移動平均は緩やかな曲線となっています。

指数平滑法について

指数平滑法（指数移動平均法、Exponential Moving Average; EMA）は、移動平均法と並んで広く使われている分析手法です。得られた過去データのうち、より新しいデータに大きなウェイトを置き、過去になるほどウェイトが小さくなる（指数関数的に減少する）というものです。

移動平均法と比べて、今回の出来事が直前の出来事に強く影響される場合、出来事の変動にできるだけ追従させたい場合などに用いられ、短期的な予測に適しています。財務上の時系列予測や株価変動分析などでも使用されます。

下記のような式で表されます。なお、αの値は、$0 < \alpha < 1$とします。

```
予測値 = α × 前回実績値 + (1 - α) × 前回予測値
       = 前回予測値 + α × (前回実績値 - 前回予測値)
```

前回の実績値が予測値からどれほど外れたかを算出し、それに一定の係数αをかけて得た修正値を、前回予測値に加減することで、今回の予測値を算出します。

それでは、前回同様、CoffeeScript で CSV を JavaScript に変換しますが、今回は、指数平滑法を利用して予測値を計算しています。

● file: src/ch07/03-yosoku/csv2js-2014kion-ema.coffee

```
# Node.js で CSV を Google Charts 用の JS に変換
# 指数移動平均も計算する

FILE_CSV = './2014kion.csv'
FILE_JS  = './2014kion-ema.js'

EMA_RANGE = 10
EMA_ALPHA = 2 / (EMA_RANGE + 1)

fs = require 'fs'
```

```
# 指数移動平均を計算する関数 ---- (※1)
calcExpMovingAverage = (i, list, alpha) ->
  # 今回の値
  value = list[i][1]
  # 前回の値
  if i == 0
    last_value = value
  else
    last_value = list[i - 1][2]
  # 予測値を計算
  new_value = last_value + alpha * (value - last_value)
  # 予測値を記録
  list[i][2] = new_value
  return new_value

# ファイルから CSV ファイルを読む
loadCSV = (filename) ->
  txt = fs.readFileSync filename, "utf-8"
  lines = txt.split "\r\n"
  # CSV テキストを二次元配列変数に変換
  list = []
  for v in lines
    cells = v.split ','
    date_s = cells[0].split("/").slice(1,3).join("/");
    temp   = parseFloat cells[1]
    list.push([date_s, temp])
  return list

# メイン処理
main = ->
  # CSV を読み込む
  list = loadCSV(FILE_CSV)
  # 各行について EMA を求める --- (※2)
  for v, index in list
    av = calcExpMovingAverage(index, list, EMA_ALPHA)
    console.log list[index][0], list[index][1], av
  # JavaScript を出力
  js = "var kion_data = " + JSON.stringify(list)
  fs.writeFileSync FILE_JS, js, "utf-8"

main()
console.log "ok"
```

プログラムを実行するには、以下のコマンドを実行します。

```
$ coffee csv2js-2014kion-ema.coffee
```

すると、前回と同様、予測値が記録されたデータが書き出されます。出力ファイル名は「2014kion-ema.js」となります。

それでは、プログラムを確認してみましょう。基本的には、前回のプログラムと似ているのですが、プログラムの(※1)の部分で、指数平滑（指数移動平均）で予測値を計算している部分が異なります。先ほどの式とまったく同じ形で、予測値を求めています。

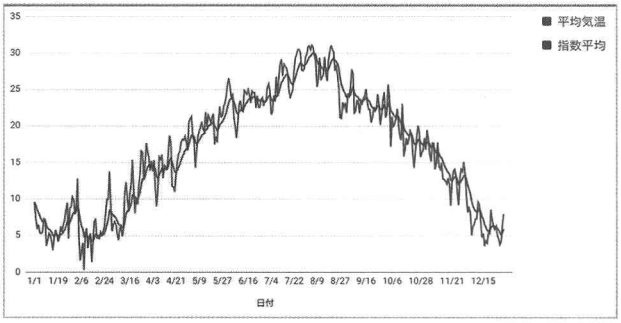
```
new_value = last_value + alpha * (value - last_value)
```

プログラムの(※2)で、前回の実績値を取り出すのに、配列変数 list に前回記録した値を参照しています。もし、前回の値がなければ、最初の値をそのまま利用しています。
さらに、ここでは平滑化定数「α」値の値に、0.1818(=2/(10+1)) を指定しました。αの値を決める方法は、ここで計算しているように「$\alpha = 2 \div (N + 1)$」とするほかに「$\alpha = 1 \div N$」と計算するものもあります。
それから、出力ファイルを読んで、グラフに値を描画するのが、以下の HTML ファイルです。

● file: src/ch07/03-yosoku/lc-2014kion-ema.html

```
<html><head>
  <!-- Google Charts の取り込み -->
  <script type="text/javascript" src="https://www.google.com/jsapi"></script>
  <!-- 気温データの読込 -->
  <script type="text/javascript" src="2014kion-ema.js"></script>
  <script type="text/javascript">
    // Charts API の初期化
    google.load('visualization', '1.1', {packages: ['line']});
    google.setOnLoadCallback(drawChart);
    // 実際の描画
    function drawChart() {
      // データオブジェクトを作成
      var data = new google.visualization.DataTable();
      // データのカラムを指定 ----- (※1)
      data.addColumn('string', '日付');
      data.addColumn('number', '平均気温');
      data.addColumn('number', '指数平均');
      // 値を設定 ---- (※2)
      data.addRows(kion_data);

  後略
```

グラフを表示するには、以下の HTML ファイルを Web ブラウザーにドラッグします。



実際の気温と EMA の値をグラフに描画

確かに、先ほど作った、単純移動平均の値とは異なる予測値が出力され、グラフも異なる形になりました。

明日の平均気温を予測しよう

それでは、紹介した予測方法を利用して、翌日の平均気温を予測してみましょう。例えば、ここでは、2004 年の平均気温のデータをダウンロードしていますので、このデータを利用して、3 月 3 日であれば、3 月 4 日の気温を予測するという形で一年を通して予測結果をコンソールに表示するというものを作ってみます。

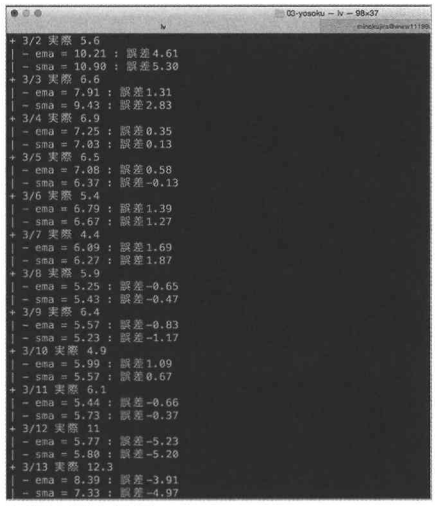
● file: src/ch07/03-yosoku/yosoku2014.coffee

```
# EMA と SMA を利用して翌日の気温を予測する

# 予測データ
FILE_CSV = './2014kion.csv'
# 定数
SMA_RANGE = 3
EMA_ALPHA = 2 / (SMA_RANGE + 1)

# モジュールの取り込み
fs = require 'fs'

後略
```

プログラムを実行するには、以下のようにコマンドをタイプします。

```
$ coffee yosoku2014.coffee
```

プログラムを実行すると以下のような結果になります。



気温予測の結果

どうでしょうか。一年を通して確認すると、ほとんど予測が当たったという日もあれば、あまり当てになっていないという日もあります。プログラムの最後に、実際の気温と予測気温の誤差の統計を表示しています。結果は以下のようなものです。

```
誤差 EMA=573.67 平均 =1.58
誤差 SMA=631.28 平均 =1.73
```

それほど正確ではないものの、思ったよりも外しているわけではないですよね。

プログラム的には、本節で紹介した二つのアルゴリズムを利用して予測を行ったものに対して、実際の気温との誤差を調べるプログラムを追加したものとなっています。

この節のまとめ

- この節では、プログラムによる予測について紹介しました。
- 予測の方法として、移動平均法と指数平滑法を使った気温推移のプログラムを作りました。
- この２つの方法は、とても計算が簡単でありながら広く普及しているものなので、さまざまな分野へ応用させることができるでしょう。

04

人工無能と会話しよう

人工無能は古くからたくさん作られてきました。ユーザーが語る言葉に反応して、ロボットがそれらしい応答するというプログラムです。ここでは、人工無能の仕組みを解説し、基本的な処理に注目して、プログラムを作ってみましょう。

ここで学ぶポイント	ツールやライブラリの一覧
●人工無能の仕組み	●Node.js ●「mecab-lite」モジュール ●「mongodb」モジュール

人工無能について

人工無脳 (英語 :Chatterbot;Chatbot) とは、人工知能に対応する用語です。ボトムアップ的な人工知能のアプローチでは「人らしさ」に到達するまでの道のりが遠いので、トップダウン的に「人らしさ」のモデルを作りこむことで「人らしさ」を作り出そうとする立場を言い表したものです。

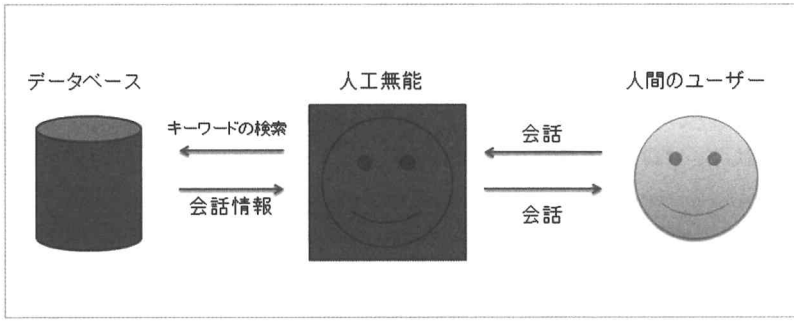
また、人工無能を実装したプログラムを「ボット (bot)」と呼びます。ボットというのはロボットの略称で、もともと人間がコンピューターを操作して行っていたような処理を、人間に代わって自動的に実行するプログラムのことを言います。ここでは、人工無能で作ったボットを「会話ボット」と呼びます。

会話ボットの目的は、ユーザーと知的な会話をすることです。一見すると、知能を持っていて、考えて会話しているように見えるのですが、多くの会話ボットは単にキーワードを拾って、内部のデータベースとのマッチングによって、最もそれらしい応答を返しているだけであることが多いものです。

Twitter 上にも数々の会話ボットが居て、独り言をつぶやいたり、あるいは誰かの発言に反応して機械的に返信を行ったりします。ずっと人間だと思って喜んで会話していた Twitter のアカウントが、実は会話ボットで、その事実を知って驚いたということもあるようです。そのくらい巧妙なボットを作ることができたら良いのですが、知的な会話ボットを作るのはなかなか難しいものです。

人工無能の仕組み

人工無能の仕組みを図にすると以下のようになります。ボットの中にはユーザーの発言に応じて、辞書を更新するものもありますし、インターネットにつながっていて、会話をインターネットから取り出すこともあります。



人工無能の仕組み

会話ボットで最初に有名になった「ELIZA」というプログラムは、DOCTORという来談者中心療法のセラピストのシミュレーションとして知られています。1960～1970年代に作られたもので歴史があります(ELIZAで調べると、iPhoneの「Siri」が友達であるという面白いエピソードも知ることができます)。

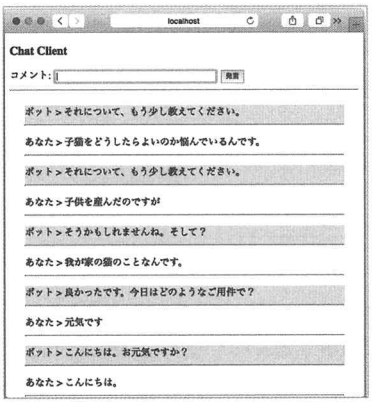
ELIZAの仕組みは、打ち込まれた言葉からキーワードを探し出して それに合わせた言葉を返します、というものです。単純な仕組みですが、実際にやってみた人の多くは それが本当にカウンセリングを行う機械であると思ったそうです。

例えば、ユーザーが「最近、眠れないのです。」と入力すると、ELIZAは「眠れないのですね。もう少し詳しく教えてください。」と会話が進んでいきます。システム自身は具体的に答えず、ユーザーに発言を続けさせるようなメッセージを返します。これにより、会話が進んでいきます。

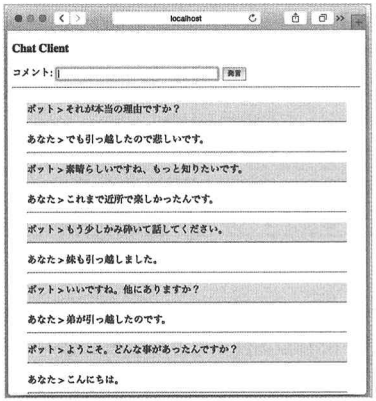
ここで作る人工無能

人工無能についてわかったところで、ここでどんな人工無能を作るのかを考えてみましょう。いきなり難しい会話ボットを作るのは大変なので、まずは、人工無能の仕組みを確かめる意味でも、キーワードにマッチした会話の語彙の中から、単純な応答を返す会話ボットを作ってみましょう。

下の例は、人工無能と会話をしているところです。新しい会話が一番上に追加されますので、実際の会話を辿るには、下から上へと読んでみてください。また、単語辞書は、ユーザーが「です・ます」調で話すことを前提として作成されているので、丁寧に語りかけてみてください。



人工無能との会話



何となく会話になっています

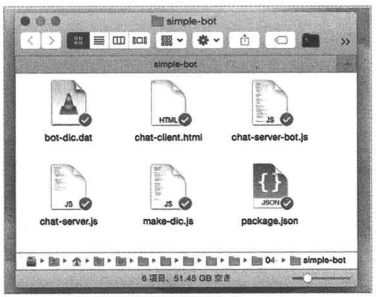
技術的な要件を決めよう

ここでは、Node.js のサーバー機能を使って、Web ブラウザー上で会話ができるプログラムを作ってみましょう。また、簡単なキーワードに反応する会話ボットを作ります。今回用意した会話辞書は、100 行程度ですが、将来的にもっと拡張できるように、MongoDB にキーワードを入れておいて、そこから会話データを引っ張ってくることにします。

会話の仕組み

このプログラムでは、以下のようにして会話をします。

(1) **ユーザーから入力を得る**
(2) **入力を形態素解析**
(3) **各単語についてデータベースの会話辞書を調べる**
(4) **もし一致するものがあればその返答を返す**



プロジェクトファイルの一覧

ここでは、複数のファイルを扱いますが、そのファイルは次のような役割を持っています。

ファイル名	説明
bot-dic.dat	会話辞書テキスト
make-dic.js	会話辞書テキストを MongoDB に挿入するプログラム
chat-client.html	チャットする画面 HTML
chat-server.js	会話ボットとチャットするサーバー
chat-server-bot.js	会話ボットの動作を定義したモジュール
package.json	必要なモジュールなどの情報を記録したファイル

人工無能のプロジェクト一覧

今回のプログラムを実行するには、前段階として、会話辞書をデータベースに登録する必要があります。そのための辞書データが「bot-dic.dat」であり、データベースに登録するプログラムが「make-dic.js」です。

そして、Web ブラウザーから利用するために、HTTP サーバーを作成しますが、それが「chat-server.js」です。基本的にこのプログラムを実行し、Web ブラウザーでこのサーバーにアクセスすると、会話を行うことができます。

プログラムの実行方法

プログラムを実行する前に、必要なモジュールを npm を使ってインストールしておきましょう。プロジェクトのファイル一式があるディレクトリに移動したら、以下のコマンドを実行します。package.json の「dependencies」の項目に依存するモジュール（ここでは「mecab-lite」と「mongodb」）の情報が書かれているので、一気に必要なモジュールをインストールすることができます。

```
$ npm install
```

そして、まずはデータベースに辞書を登録します。

データベースに辞書を登録するのに当たって、MongoDB を起動しておく必要があります。MongoDB は 4 章で紹介していますが、OS によって起動の仕方が異なりますので、MongoDB のインストールの項目で確認してください。以下のようにしても MongoDB を起動できます。

```
$ mongod --dbpath ~/mongodb
```

次に、辞書をデータベースに登録するスクリプトを実行します。

```
$ node make-dic.js
```

それから、会話ボットとのチャットサーバーを起動します。デフォルトの設定では、ポート 1337 番を使用して、サーバーを起動します。

このとき、仮想マシン上の CentOS を利用する場合には、ホストマシンの Web ブラウザーから仮想マシンの中のサーバーにアクセスすることができません。そこで、設定ファイルの書き換えが必要となります。

まず、Vagrant の仮想マシン上の設定ファイル「Vagrantfile」をテキストエディターで開きます。そして、以下の設定を加えます。（設定ファイル末尾の end より上に挿入します。）

```
# 設定ファイル「Vagrantfile」に以下の一行を書き加える
config.vm.network "forwarded_port", guest: 1337, host: 1337
```

コマンド「vagrant reload」を実行すれば、ホストマシンの1337番ポートから仮想マシンの1337番ポートにアクセスできるようになります。

では、以下のコマンドを実行して、サーバーを起動しましょう。

```
$ node chat-server.js
```

これで、準備が整いました。Webブラウザーを開き、以下のURLにアクセスしましょう。

```
http://localhost:1337/
```

無事チャット画面が表示されたら成功

会話辞書を作ろう

会話ボットを作るにあたって、その心臓部分と言えるのが、会話辞書です。今回の人工無能は、簡単なキーワードに反応して会話を返すという単純なものです。ですから、会話辞書を作る際にも、「検索用の語句」に加えて、どの順番でパターンを調べるかの「ランク」、否定・肯定を調べるための「パターン」、そして返答用の「メッセージ本文」と、4つのパラメーターを入力していくことにしました。

以下は、辞書ファイル「bot-dic.dat」からの抜粋です。左から、検索語句、検索ランク、パターン、応答メッセージの順にカンマ区切りで並んでいます。

```
; key, rank, pattern, message
こんにちは , 0, *, こんにちは。お元気ですか?
こんにちは , 0, *, ようこそ。どんな事があったんですか?
元気 , 2, 元気ですか , 元気です。あなたは?
元気 , 1, 元気です , 良かったです。今日はどのようなご用件で?
おはよう , 0, *, おはようございます。今日のご用件は?
こんばんは , 0, *, こんばんは。お元気ですか?
おやすみ , 0, *, おやすみなさい。ゆっくり休んでください。
; 語尾
です , 0, ですね , そうですね。
です , 0, ですね , 良いですね。それで?
です , 0, です , そうかもしれませんね。そして?
です , 0, ですか , そうかもしれません。
です , 0, ですか , わかりませんが、どうして?
です , 0, でした , お疲れ様でした。
です , 0, でした , そうでしたね。それから、どうしたんですか?
...
```

この辞書データを、データベースのMongoDBに挿入するのが、以下のプログラムです。辞書データ「bot-dic.dat」を読み込んで、MongoDBの「simple-bot」というコレクションにデータを挿入します。テキストファイルの辞書データを更新したら、毎回、スクリプトを実行してデータベースを更新する必要があります。

● file: src/ch07/04-munou/simple-bot/make-dic.js

```
// ボットのためのキーワード辞書を作成
//-----------------------------------
// 会話辞書テキストファイルの指定
var FILE_DIC = 'bot-dic.dat';
// MongoDBの接続情報 --- (※1)
var MONGO_DSN = "mongodb://localhost:27017/simple-bot";
var mongo_db; // 接続オブジェクト

// モジュール
var mongo_client = require('mongodb').MongoClient;
var fs = require('fs');

// MongoDBに接続 --- (※2)
mongo_client.connect(MONGO_DSN, function (err, db) {
  // エラーチェック
  if (err) { console.log("DB error", err); return; }
  // MongoDBの接続オブジェクトを記憶
  mongo_db = db;

  // コレクションを取得 --- (※3)
  var collection = db.collection('keywords');

  // 既存のデータがあれば一度初期化 ---- (※4)
  collection.drop(function(err, reply) {
    // 初期化後に挿入
    insertKeywords(collection);
  });
});

// MongoDBに辞書データを挿入 --- (※5)
function insertKeywords(collection) {
  var cnt = 0, dataCount = 0;
  // テキストデータを読み込む
  var txt = fs.readFileSync(FILE_DIC, "utf-8");
  // 各行を処理
  var lines = txt.split("\n");
  for (var i in lines) {
    var line = trim(lines[i]);
    if (line == "") continue; // 空行
    if (line.substr(0,1) == ";") continue; // コメント
    var cells = line.split(",");
    var key = trim(cells[0]);
    var rank = parseInt(trim(cells[1]));
    var pat = trim(cells[2]);
    var msg = trim(cells[3]);
    // 挿入 ---- (※6)
    collection.insert({
      "key": key, "rank": rank,
      "pattern": pat, "msg": msg
    }, function(err, result) {
      console.log(cnt+":inserted:", result.ops);
      if (++cnt == dataCount) {
        console.log("done");
```

```
        mongo_db.close();
      }
    });
    dataCount++;
  }
}

// 前後の空白トリムを行う
function trim(s) {
  s = "" + s;
  return s.replace(/(^\s+|\s+$)/g, "");
}
```

プログラムを確認してみましょう。プログラムの(※1)では、MongoDBへの接続情報を記述しています。そして、データベースに接続するのが、プログラムの(※2)のconnect()メソッドです。データベースには複数のコレクションを保存できますが、プログラムの(※3)で「keywords」コレクションを選択しています。

プログラムの(※4)では、辞書データを更新する際に、会話データが重複するのを防ぐため、挿入前に、一度、コレクションの中身を初期化します。それが、drop()メソッドです。初期化が完了したら、いよいよ辞書データの挿入です。

プログラム(※5)のinsertKeywords()関数でテキストデータを読み込み、辞書データを挿入する処理を行います。テキストデータを読み出して、一行ずつデータベースに挿入するという作業になります。実際に挿入するのが(※6)の部分です。ここでは、読み出したテキストに、キーを付けて次のようなJSONデータを挿入します。

```
{
  key     : "キーワード",
  rank    : 検索順位,
  pattern : "絞り込み用のパターン",
  msg     : "応答メッセージ"
}
```

会話ボットとのチャット画面

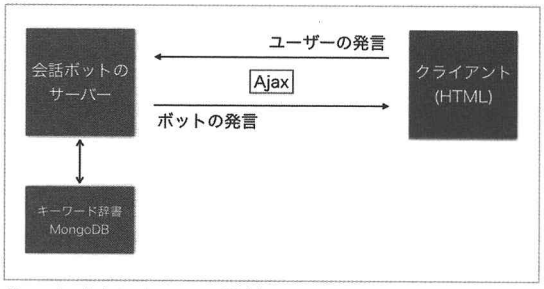
次に、会話ボットとチャットする画面を構成するHTMLファイル「chat-client.html」を見てみましょう。この画面をサーバーに対するチャットクライアントと呼ぶことにします。クライアントは、Ajaxを利用してサーバーと通信を行います。前半部分がHTMLのコードです。後半部分がJavaScriptでサーバーと通信する部分となっています。

● file: src/ch07/04-munou/simple-bot/chat-client.html

```
<!DOCTYPE html>
<html><meta charset="utf-8"><body>
<h3>Chat Client</h3>
<!-- ユーザーのコメント発言フォーム -->
<div>
  コメント:
  <input id="msg" type="text" value="こんにちは。" size="40">
  <button id="talk_btn">発言</button>
</div>
<hr>
<!-- ログの表示 -->
<div id="log" style="margin:24px;"></div>
```

```
<script type="text/javascript">
// サーバーAPIの指定 ----(※1)
var api = "http://localhost:1337/api?";

// ボタンをクリックした時のイベントを定義
$("#talk_btn").onclick = sendMsg;

// クリックした時の処理 --- (※2)
function sendMsg() {
  // コメントを取得し、パラメーターを組み立てる
  var msg = $("#msg").value;
  var url = api + "msg=" + encodeURIComponent(msg);
  // AjaxでAPIにメッセージを送信 --- (※3)
  $ajax(url, function(xhr, txt) {
    // サーバーからの返信をログに表示
    $("#msg").value = "";
    $("#msg").focus();
    var e = JSON.parse(txt);
    // ユーザーの発言
    var p_you = document.createElement("p");
    p_you.innerHTML = "あなた &gt; " + msg + "<hr>";
    // ボットの発言
    var p_bot = document.createElement("p");
    p_bot.innerHTML = "ボット &gt; " + e["msg"] + "<hr>";
    p_bot.style.backgroundColor = "#e0f0ff";
    // ログに追加
    var log = $("#log");
    log.insertBefore(p_you, log.firstChild);
    log.insertBefore(p_bot, p_you);
  });
}

// DOMを返す
function $(q) { return document.querySelector(q); }

// Ajax関数 --- (※4)
function $ajax(url, callback) {
  var xhr = new XMLHttpRequest();
  xhr.open('GET', url, true);
  xhr.onreadystatechange = function() {
    if (xhr.readyState == 4) { //通信完了
      if (xhr.status == 200) { //HTTPステータス200
        callback(xhr, xhr.responseText);
      }
    }
  };
  xhr.send(''); // 通信を開始
  return xhr;
}
</script>
</body></html>
```

では、プログラムを見てみましょう。ユーザーがメッセージを入力した後、発言ボタンをクリックすると、メッセージをサーバーに送信します。すると、サーバー側が会話ボットの返答を用意して、返答を送り返して来るという仕組みです。



サーバーとクライアントの関係

プログラムの(※1)では、サーバーAPIのURL指定を行っています。今回のサーバー側プログラムでは「/api」にアクセスがあるかどうかで処理を分岐しています。

プログラムの(※2)以下、sendMsg()関数では、発言ボタンをクリックした時の処理を記述しています。実際にAjaxで通信しているのは、(※3)の部分です。その一行前の部分で、サーバーに送信するURLパラメーターを作成しています。Ajaxでサーバーと通信し、サーバーからの返信があれば、それをHTMLのログに追加します。

プログラムの(※4)では、Ajaxを手軽に行う関数を定義しています。一般的には、jQueryなどのライブラリで定義されているAjaxの関数を使うことが多いのですが、ここでは、簡易関数を定義して使っています。具体的には、XMLHttpRequestを利用して、非同期通信が可能となります。

会話ボットサーバー

では、HTTPサーバーを作成する、Node.jsのコードを見てみましょう。機能としては、二つあります。まず、ルートパスにアクセスがあった時には「chat-client.html」を送出します。そして、クライアントのAjaxから「/api」にアクセスがあったときには、会話ボットの応答を返します。ちなみに、会話ボットの応答を作成する部分は、別ファイル「chat-server-bot.js」で定義しています。

● file: src/ch07/04-munou/simple-bot/chat-server.js

```
// チャットサーバーを作る
//-----------------------------
// 設定
var SERVER_PORT = 1337; // サーバーポート
var FILE_CLIENT = "chat-client.html";

// モジュールの取り込み
var
  http = require('http'),
  URL  = require('url'),
  path = require('path'),
  fs   = require('fs'),
  bot  = require('./chat-server-bot.js');

// サーバーを起動 --- (※1)
var svr = http.createServer(checkRequest);
svr.listen(SERVER_PORT, function(){
  console.log("サーバー起動しました");
  console.log("http://localhost:" + SERVER_PORT);
});
```

```
// サーバーにリクエストがあった時の処理 --- (※2)
function checkRequest(req, res) {
  var uri = URL.parse(req.url, true);
  var pathname = uri.pathname;
  // パス名で処理を分岐
  if (pathname == "/api") {
    apiRequest(req, res, uri);
  } else if (pathname == "/") {
    res.writeHead(200, {'Content-Type':'text/html'});
    res.end(fs.readFileSync(FILE_CLIENT, "utf-8"));
  } else {
    res.writeHead(404, {'Content-Type':'text/plain'});
    res.end("File not found");
  }
  console.log(pathname);
};

// APIへのリクエストを処理 --- (※3)
function apiRequest(req, res, uri) {
  msg = uri.query["msg"];
  bot.getResponse(msg, function(bot_msg) {
    body = JSON.stringify({"msg":bot_msg});
    res.writeHead(200, {'Content-Type':'application/json'});
    res.end(body);
  });
};
```

プログラムを見ていきましょう。HTTP サーバーを起動しているのがプログラムの (※1) です。Node.js で HTTP サーバーを起動するのは、このように、わずか 2 行で実現できるのが良い所です。createServer() メソッドの引数には、クライアントからアクセスがあったときに実行する処理を記述します。そして、listen() メソッドには、サーバーのポート番号と、サーバーが起動したときに実行する処理を指定します。

プログラムの (※2) では、サーバーにアクセスがあったときに実行する処理です。URL.parse() メソッドにより、アクセス情報を扱いやすくパースします。この pathname プロパティには、どの URI にアクセスがあったのか「/」や「/api」などのパス名が入ります。そして、pathname で処理を分岐します。ルートへのアクセス「/」であれば、クライアント用の HTML を送出し、「/api」へのアクセスであれば、API リクエストを処理します。

プログラムの (※3) は、会話ボットの動作を定義しているモジュール「chat-server-bot.js」から、getResponse() メソッドを実行し、会話ボットの返答を、JSON 形式で送り返すというものになっています。

会話ボットの会話生成モジュール

続いて、会話ボットの会話を生成するモジュール「caht-server-bot.js」を見ていきましょう。

● file: src/ch07/04-munou/simple-bot/chat-server-bot.js

```
// 会話ボットの応答を生成するモジュール
//---------------------------------

// MongoDBの設定情報
var MONGO_DSN = "mongodb://localhost:27017/simple-bot";

// モジュールの取り込み
var Mecab = require('mecab-lite'),
    mecab = new Mecab(),
```

```
    mongo_client = require('mongodb').MongoClient;

// MongoDBの接続情報を保持する変数
var mongo_db = null, keywords_co;

// 外部に getResponse() メソッドを公開 --- (※1)
module.exports = {
  "getResponse": getResponse
};

// 会話ボットの応答を返す関数 --- (※2)
function getResponse(msg, callback) {
  checkDB(function(){
    var bot = new Bot(msg, callback);
    bot.talk();
  });
}

// MongoDBへ接続 --- (※3)
function checkDB(next_func) {
  // 既に接続していれば何もしない
  if (mongo_db) {
    next_func(); return;
  }

  // MongoDBに接続
  mongo_client.connect(MONGO_DSN, function (err, db) {
    // エラーチェック
    if (err) { console.log("DB error", err); return; }
    // 接続情報を記録
    mongo_db = db;
    // コレクションを取得
    keywords_co = db.collection('keywords');
    // 次の処理を実行
    next_func();
  });
}

// ボットクラスの定義 ---- (※4)
function Bot(msg, callback) {
  this.callback = callback;
  this.msg = msg;
  this.results = [];
  this.words = [];
  this.index = 0;
}

// ボットからの応答を得るメソッド ---- (※5)
Bot.prototype.talk = function () {
  var self = this;
  // 形態素解析 --- (※6)
  mecab.parse(this.msg, function (err, words) {
    if (err) {
      self.callback("Error");
      return;
    }
    // 単語を一つずつ確認する ---- (※7)
    self.index = 0;
    self.words = words;
    self.nextWord();
  });
```

```
};

// 各単語を一語ずつ調べるメソッド ---- (※8)
Bot.prototype.nextWord = function() {
  // 単語を最後まで調べたか確認
  if (this.index >= this.words.length) {
    this.response();
    return;
  }
  // データベースを検索
  var w = this.words[this.index++];
  // 活用のない単語 -「頑張ら」なら「頑張る」を利用
  var org = (w.length >= 7) ? w[7] : w[0];
  var self = this;
  keywords_co
  .find({key:org})
  .toArray(function(err, rows) {
    // データベースに合致する語があったか?
    if (rows.length == 0) {
      self.nextWord(); return;
    }
    // パターンにマッチするか確認 --- (※9)
    var keys = rows.filter(function(el, index, ary) {
      if (el.pattern == "*") return true;
      if (self.msg.indexOf(el.pattern) >= 0) return true;
      return false;
    });

後略
```

Node.js では、基本的に外部のファイルで定義されている変数にはアクセスできません。しかし、module.exports プロパティに設定したメソッドやオブジェクトを require() を通じて取得できる仕組みとなっています。プログラムの (※ 1) の部分では、getResponse() メソッドを外部に公開するように定義します。

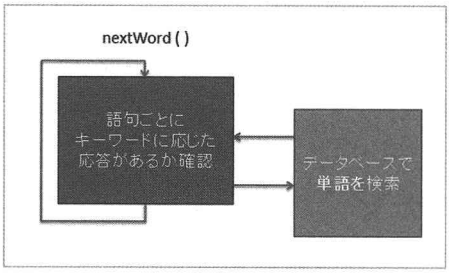
プログラム (※ 2) の getResponse() メソッドが、会話ボットの応答を返すメイン関数となっています。ここでは、MongoDB のデータベースに接続することと、Bot オブジェクトを作成し応答を作成することの二つの処理を実行しています。

プログラムの (※ 3) では、MongoDB への接続を行います。しかし、DB への接続は一度だけ行って、接続をキープし続けます。そのため、二度目以降に接続処理が実行された場合は、すぐに次の処理を行うようにしています。

プログラムの (※ 4) 以降で、Bot クラスの定義を行います。JavaScript でオブジェクト指向プログラミングを行う場合、関数の定義が即ちクラス定義およびコンストラクタの定義となります。このクラスでは、返答メッセージを作成します。

プログラムの (※ 5) にある、talk() メソッドで、ボットとの会話を行います。具体的には、(※ 6) の部分で形態素解析を行い、(※ 7) の部分で分割済みの単語を一つずつ確認していきます。ちなみに、形態素解析のためには、MeCab(mecab-lite モジュール) を利用しています。

ちょっとわかりにくいのが、形態素解析で分割した単語を一つずつ確認する方法です。プログラムの (※ 8) の部分ですが、ここでは、for 構文を使わず、変数 index と、単語の一覧配列の変数 words の二つを利用して、毎回、nextWord() メソッドを呼び出す方法で実現しています。その理由は、単語を確認する際に、MongoDB のデータベースを検索する必要がありますが、データベースの結果を得るのに、非同期関数を利用するため、for 構文の中に収めることが難しいのです。単語を一つ処理するごとに、nextWord() メソッドを呼んで、全部の単語を確認しているのです。



nextWord メソッドの動作

また、プログラムの (※ 9) では、データベースを検索して語句が見つかったときに、さらに任意のパターンにマッチするかどうかを調べています。例えば、「嬉しい」という単語でデータベースを検索し、さらに、「嬉しくない」か「嬉しいか」のどちらにマッチするかで処理を変えるという訳です。

この節のまとめ

- この節では、基本的な人工無能（会話ボット）の作り方を紹介しました。
- 今回作成したのは、かつて一世を風靡した「ELIZA」を模したものです。
- しかし、ここで作った人工無能は、会話の応答を全部自分で作らねばならかったので大変です。
- 返答をする際に、マルコフ連鎖などを使った自動作文機能を追加したり、インターネットからダウンロードした文章を語るようにするなら、より面白いでしょう。ぜひオリジナルの人工無能を作って育ててみてください。

05

サポートベクターマシンで文字認識しよう(前編)

「サポートベクターマシン」とは、「教師あり学習を用いる」パターン認識モデルの一つです。識別や回帰分析へ適用できるものです。ここでは、機械学習の利用例として、文字認識に挑戦してみたいと思います。

ここで学ぶポイント	ツールやライブラリの一覧
●サポートベクターマシン(SVM) ●文字認識のやり方	●Node.js ●「node-svm」モジュール

サポートベクターマシンとは？

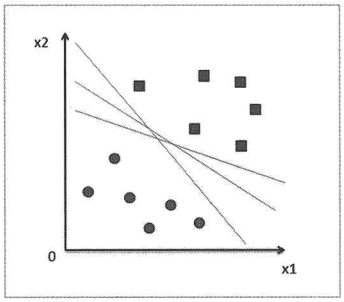
私たち人間の学習能力は無限です。人間が持っている五感を活かして、さまざまな事柄を認識し、学習していくことができます。それに比べてコンピューターの能力は有限です。人間がプログラムを作って、データや式を与えられてはじめて、指示通りに処理を行うことができます。

そこで、コンピューターに人間のような学習能力を持たせるという研究が行われてきました。その成果の一つが「多層パーセプトロン」です。これは、1980年代に開発され、さまざまな分野で応用されてきました。しかし、いくつか問題点もありました。

これを解決したのが、「サポートベクターマシン (Support Vector Machine; SVM)」です。これは、パターン認識手法の一つです。1995年に統計的学習理論の枠組みで提案された学習機械です。特に、パターン認識の能力に優れています。サポートベクターマシン（以下、SVMと略す）には「マージン最大化」という方針を採用して識別平面を決定します。

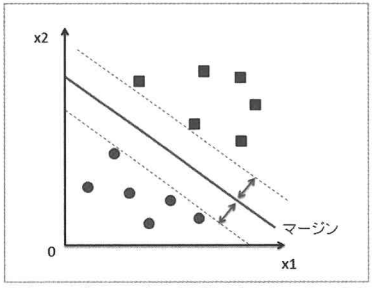
そもそも、一般的に「パターン認識」の問題を扱う場合を考えてみます。例えば、AとBの二種類のパターンがあったとしたら、AとBのパターンをどのように分けるのかを決めるのがパターン認識の目標となります。そのために、AとBの出現場所を調べ、分類の基準となる境界を決めることになります。このパターンの境界のことを「識別平面」と呼びます。そして、その正しい境界を決めることができれば、新しい未知の要素が現れたときにも、AとBのどちらのパターンに分類したら良いのかを決定することができるというわけです。

具体的なグラフを見るとわかりやすいので、次のグラフを見てください。次のグラフには、「●」と「■」の二種類のデータがあります。そして、次の図のように、この二種類のデータをどのように分けるのかを考えたとき、適当な境界線を引こうと思えば、いくつもの線を引くことができます。



データを●と■のどちらかに分けるとき、いろいろな分類が考えられる

いくつもの境界線を引くことができるものの、普通に考えてみると、●と■のちょうど真ん中を通るような線を引くことができれば、一番良さそうだというのがわかるのではないでしょうか。この、真ん中を通る線を決めるとき、下図のように、識別平面からもっとも近い既知のパターンとの距離（マージン）を最大になるように決定するのが最良の結果となります。



既知のパターンとの距離を最大になるようにしたもの

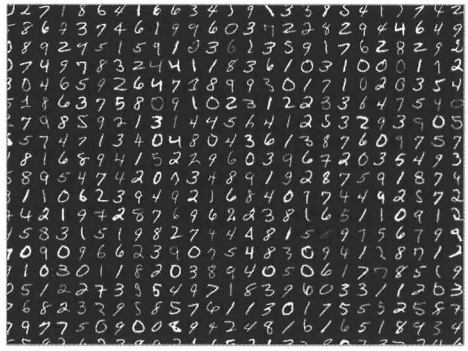
これが、SVM の特徴の「マージン最大化」という方針です。SVM を使うと、未知のサンプルに対しても、正しく分類する確率が高いと言われています。このような能力を、学習理論では汎化能力と呼びます。

文字認識に挑戦してみよう

ここでは、SVM を手書き文字の認識に利用してみたいと思います。手書き文字の認識で最も簡単なのが、数字の認識です。大量の画像データを学習して、０から９までのいずれかに分類するというものです。

都合の良いことに、大量の手書き数字データを公開している Web サイトがあります。MNIST というサイトです。ここでは、０から９までの手書き数字データ 70,000 点が公開されており、機械学習やパターン認識に利用することができます。各手書き数字のデータは、縦横 28x28 ピクセルデータに揃っています。

```
THE MNIST DATABASE of handwritten digits
[URL] http://yann.lecun.com/exdb/mnist/
```



手書き数字データ七万点が公開されています

SVM で学習モデルを作るまでのシナリオ

膨大な七万点の手書き数字データを学習させて、精度の高い文字認識を目指しましょう。ここでは、作業を始める前に SVM で文字認識させるまでの手順の概要を述べておきましょう。

SVM を利用する基本的な手順は、

(1) **パターンを学習させる**
(2) **学習データに基づいて手書き文字を判定させる**

というものです。

そこで、手書き数字のデータベースから、パターンを学習させるまでの手順は以下のようになります。

(1) **MNIST の手書き数字データをダウンロード**
(2) **データを、扱いやすいように、CSV ファイルに変換**
(3) **CSV ファイルを元に、学習データ (.svm) を作成する**
(4) **学習データを SVM に学習させ、モデル (.model) を作成する**

手書き数字データをダウンロードしよう

MNIST のサイトで公開されているデータベースをダウンロードしましょう。ダウンロードが必要なのは、以下の四つのファイルです。

ファイル名	説明
train-images-idx3-ubyte.gz	学習用イメージ
train-labels-idx1-ubyte.gz	学習用イメージのラベル
t10k-images-idx3-ubyte.gz	テスト用イメージ
t10k-labels-idx1-ubyte.gz	テスト用イメージのラベル


THE MNIST DATABASE

of handwritten digits

Yann LeCun, Courant Institute, NYU
Corinna Cortes, Google Labs, New York
Christopher J.C. Burges, Microsoft Research, Redmond

The MNIST database of handwritten digits, available from this page, has a trainin
60,000 examples, and a test set of 10,000 examples. It is a subset of a larger se
available from NIST. The digits have been size-normalized and centered in a fixe
image.

It is a good database for people who want to try learning techniques and pattern
recognition methods on real-world data while spending minimal efforts on
preprocessing and formatting.

Four files are available on this site:

train-images-idx3-ubyte.gz: training set images (9912422 bytes)
train-labels-idx1-ubyte.gz: training set labels (28881 bytes)
t10k-images-idx3-ubyte.gz: test set images (1648877 bytes)
t10k-labels-idx1-ubyte.gz: test set labels (4542 bytes)

please note that your browser may uncompress these files without telling

MNIST のページ：この画面の下方にあるリンクからデータをダウンロードする

ファイルは、４つなのですが、実際は、学習用イメージ (60,000 点) と、テスト用イメージ (10,000 点) の２つのファイルです。　それぞれについて、手書きの数字を表すピクセルデータ (*-images-*) の一覧と、そのピクセルデータが 0 から 9 のどの数字なのかを示すラベルデータ (*-labels-*) の一覧に分けられています。しかも、それがバイナリデータとして提供されているのです。

これらは「.gz」形式で圧縮されているので、まずは解凍しておきましょう。CentOS(Mac OS X) では、以下のような「gzip」コマンドで解凍することができます。

```
# gzip でファイルの解凍例
$ gzip -dc infile.gz > outfile
```

ファイルが４つだけなので、このようにして、一つずつ解凍しても良いのですが、ファイル処理の練習もかねて、複数の GZ ファイルを一気に解凍するプログラムも作ってみましょう。以下のプログラムは、カレントディレクトリにある GZ ファイルの一覧をすべて解凍するというものです。

● file: src/ch07/05-svm/kaitou.js

```
// カレントディレクトリの GZ ファイルを一気に解凍 for Node.js

var fs = require('fs');
var exec = require('child_process').exec;

// ファイル一覧を得る
fs.readdir('.', function(err,files) {
  files.forEach(function(file) {
    // .gz 以外は無視
    if (!/\.gz$/.test(file)) return;
    // 解凍
後略
```

プログラムを実行するには、以下のコマンドを入力します。

```
$ node kaitou.js
```

以上で、カレントディレクトリにある GZ ファイルが解凍されます。プログラムを見てみるとわかりますが、プログラムの中で、gzip コマンドを呼び出しているだけですね。今後、Node.js を使ってバッチ処理を書く際には、このようにして、外部のプログラムを実行するコマンドを生成して実行することもできます。

バイナリデータを CSV に変換しよう

ところで、MNIST から取得した手書き画像のデータですが、バイナリデータというちょっと特殊な形式で配布されています。私たちが扱いやすいと感じるのは、画像データそのまま、あるいは、CSV ファイルなどで提供されているものでしょう。そこで、まずは、扱いやすい形に変換してみましょう。

バイナリデータとは言っても、それほど複雑な形式ではなく、次のようなデータ形式となっています。先頭の 16 バイトがファイルヘッダーで、その後、28x28 バイトのピクセルデータが、60,000 個続くというものになっています。

```
イメージファイルの形式：
[offset] [type]          [value]          [description]
0000     32 bit integer  0x00000803(2051) magic number
0004     32 bit integer  60000            number of images
0008     32 bit integer  28               number of rows
0012     32 bit integer  28               number of columns
0016     unsigned byte   ??               pixel
0017     unsigned byte   ??               pixel
........
xxxx     unsigned byte   ??               pixel
```

以下がラベルファイルの形式です。これは、上記のイメージファイルが、0から9までのいずれに該当するかの情報が記されています。イメージファイルの何個目の画像が、何番の数字かを表すデータとなっています。

```
ラベルファイル：
[offset] [type]          [value]          [description]
0000     32 bit integer  0x00000801(2049) magic number (MSB first)
0004     32 bit integer  60000            number of items
0008     unsigned byte   ??               label
0009     unsigned byte   ??               label
........
xxxx     unsigned byte   ??               label
The labels values are 0 to 9.
```

概要がわかりましたので、さっそくプログラムを作ってみましょう。データファイルサイズが大きいので、ファイルハンドルを利用して、ファイルからデータを少しずつ読み出すことでデータを読み込んでいます。ファイルハンドルを利用したファイル操作を行うには、fs.openSync() メソッドを使います。また、データは、バイナリファイルなので、Buffer オブジェクトを利用して任意のバイトを読み出しています。

● file: src/ch07/05-svm/mnistdb2csv.js

```
// MNIST 手書きデータを CSV に変換する for Node.js

// ファイル名を指定
var DIR_IMAGE   = __dirname + "/image";

// モジュールの取り込み
var fs = require('fs');

// 変換処理
convertToCsv("train");
convertToCsv("t10k");

function convertToCsv(basename) {
  console.log("convert: " + basename);
```

```
// 各種ファイル名を決定
var file_images = basename + "-images-idx3-ubyte";
var file_labels = basename + "-labels-idx1-ubyte";
var file_csv = basename + ".csv";

// ファイルを開く
var f_img = fs.openSync(file_images, "r");
var f_lbl = fs.openSync(file_labels, "r");
var f_out = fs.openSync(file_csv, "w+");

if (!fs.existsSync(DIR_IMAGE)) {
  fs.mkdir(DIR_IMAGE);
}

// ヘッダーを読む
var buf_i = new Buffer(16);
fs.readSync(f_img, buf_i, 0, buf_i.length);
var buf_l = new Buffer(8);
fs.readSync(f_lbl, buf_l, 0, buf_l.length);

// ヘッダーをチェック
var magic      = buf_i.readUInt32BE(0);
var num_images = buf_i.readUInt32BE(4);
var num_rows   = buf_i.readUInt32BE(8);
var num_cols   = buf_i.readUInt32BE(12);
var num_size   = num_rows * num_cols;
if (magic != 0x803) {
  console.error("[ERROR] Broken file=", magic.toString(16));
  process.exit();
}
console.log("num_of_images=" + num_images);
console.log("num_of_rows=" + num_rows);
console.log("num_of_cols=" + num_cols);
console.log("num_of_pixel_size=" + num_size);

// 画像を取り出す
var buf_img = new Buffer(num_size);
var buf_lbl = new Buffer(1);
var mini_csv = "";
for (var i = 0; i < num_images; i++) {
  // 経過を表示
  if (i % 1000 == 0) console.log( i + "/" + num_images );
  // 画像を読む
  var pos_i = i * num_size + 16;
  fs.readSync(f_img, buf_img, 0, num_size, pos_i);

  // ラベルを読む
  var pos_l = i * 1 + 8;
  fs.readSync(f_lbl, buf_lbl, 0, 1, pos_l);
  var no = buf_lbl[0];

  // PGM形式として保存（テスト用）
  if (i < 30) {
    var s = "P2 28 28 255\n";
    for (var j = 0; j < 28 * 28; j++) {
      s += buf_img[j] + " ";
      s += (j % 28 == 27) ? "\n" : "";
    }
    var img_file =
      DIR_IMAGE + "/" + basename +
      "-" + i + "-" + no + ".pgm";
```

```
      fs.writeFileSync(img_file, s, "utf-8");
    }

    // CSVとして保存
    var cells = [];
    for (var j = 0; j < 28 * 28; j++) {
      cells.push(buf_img[j]);
    }
    s = no + "," + cells.join(",") + "\n";
    fs.writeSync(f_out, s, null, "utf-8");

    // テスト用のミニサイズCSVを作成
    if (i < 1000) {
      mini_csv += s;
      if (i == 999) {
        fs.writeFileSync(
            basename + "-mini.csv",
            mini_csv, "utf-8");
      }
    }
  }
  console.log("ok:" + basename);
}
```

プログラムを実行するには、以下のコマンドを実行します。

```
$ node mnistdb2csv.js
```

プログラムを実行すると、以下の四つのCSVファイルが出力されます。

- **train.csv　　手書きデータ六万件をCSVに変換したもの**
- **train-mini.csv　　上記から先頭の1000件だけを抽出したもの**
- **t10k.csv　　検証用の一万件の手書きデータをCSVに変換したもの**
- **t10k-mini.csv　　上記から先頭の1000件だけを抽出したもの**

基本的には「train.csv」と「t10k.csv」の二つがメインで利用するデータです。では、なぜ1000件だけをCSVに変換したものを用意したのかというと、SVMに学習させるのにものすごく時間がかかるので、とりあえず、1000件分だけを学習させて、すぐに結果を確認したいときのために用意してみました。

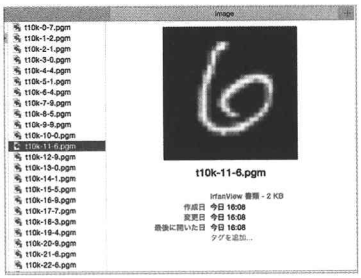
ここで作成したCSVファイルは、次のようなデータとなっています。

```
(どの数字か),(画像データの各ピクセル)
(どの数字か),(画像データの各ピクセル)
(どの数字か),(画像データの各ピクセル)
(どの数字か),(画像データの各ピクセル)
...
```

画像データの各ピクセルの部分は、左上(0,0)から右下(27,27)へと28x28個順にカンマで区切って並んだものとなっています。

COLUMN　廃止された「E4X」について

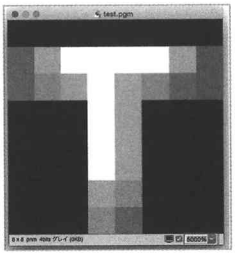
正しく画像が抽出できているかどうかを確かめるために、はじめの 30 件だけを PGM 形式のファイルに出力しています。PGM 形式というのは、画像データの各ピクセルの値をテキストで指定できるという都合の良い形式です。



PGM 形式の画像

PGM 形式のファイルを実際にテキストファイルで開いて見るとわかるのですが、ただのテキストファイルであり、画像サイズや色数などの情報をいくつか書いた後は、ピクセルの値が並ぶというものです。例えば、8x8 ピクセルのグレイスケール画像であれば、次のようなデータになります。先頭の「P2」が PGM 画像を識別するコード、そして「8 8」が画像サイズ、「9」がグレイスケールの最大値となっています。

```
P2 8 8 9
0 0 0 0 0 0 0 0
6 8 9 9 9 9 8 5
6 7 8 9 8 7 6 5
0 0 0 9 8 0 0 0
0 0 0 9 8 0 0 0
0 0 0 9 8 0 0 0
0 0 0 8 7 0 0 0
0 0 0 7 6 0 0 0
```



PGM 形式の画像

Windows の IrfanView や Mac の ToyViewer などの画像ビューワーであれば、PGM をサポートしているので、画像データの様子を確認できます。このように、PGM 画像を利用することで、容易に画像データを確認することができます。

node-svm をインストールしよう

一行が一画像となっているCSVファイルができたので、いよいよSVMに画像を学習させましょう。それにあたって、Node.jsの「node-svm」モジュールをインストールしましょう。この「node-svm」モジュールは、C++で書かれたSVMの実装「libsvm」をNode.jsのモジュールにしたものです。

Node.jsから使えるモジュールに加えて、SVMをコマンドラインから使うためのツールもインストールしてくれるので、「-g」オプションをつけて、コマンドが利用できるようにしておきましょう。

```
$ sudo npm install -g svm
```

学習用データファイルを作る

SVMに画像を学習させるために、SVM用の学習ファイルを作成する必要があります。このSVMファイルは、CSVファイルを元に作成するのですが、SVMファイルがどのようなフォーマットであるのか紹介します。一行が一データとなっています。

```
[書式] SVMファイル
<label> <index1>:<value1> <index2>:<value2> <index3>:<value3>...
<label> <index1>:<value1> <index2>:<value2> <index3>:<value3>...
<label> <index1>:<value1> <index2>:<value2> <index3>:<value3>...
...
```

このように、学習データであるSVMファイルの仕組みは、一行に一つの画像データを与える点など、CSVファイルとそう対して変わりません。ちなみに「index:value」の内、valueが0になるものについては、省略して記述しても良いことになっています。そのため、画像データのピクセルが0のものについては、省略して記述することができます。

それでは、CSVファイルから、こうした学習用SVMファイルを作成するプログラムを作ってみましょう。プログラムを実行すると、上記で生成した四つのCSVファイルを元にした、それぞれのSVMファイルが生成されます。

● file: src/ch07/05-svm/csv2trainfile.js

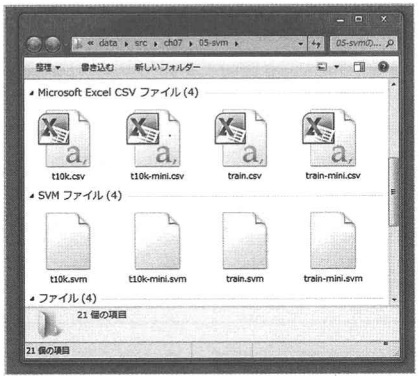
```
// CSVを元に、SVM学習データ(SVM)を生成

var fs = require('fs');

// 二種類のデータを処理
csv2svm('train-mini.csv');
csv2svm('train.csv');
csv2svm('t10k-mini.csv');
csv2svm('t10k.csv');
console.log("ok");

// CSVファイルからSVMファイルを作成
function csv2svm(file_csv) {
  // ファイル名を決定
  var file_svm = file_csv.replace(/\.csv$/, "") + ".svm";
  console.log("[I N] " + file_csv);
  console.log("[OUT] " + file_svm);
  console.log(file_svm);

  // 保存用ファイルを開く
```

```
  var f_svm = fs.openSync(file_svm, "w");

  // 読込
  var csv = fs.readFileSync(file_csv, "utf-8");
  var lines = csv.split("\n");

  // データを作成
  for (var i in lines) {
    // 経過報告
    if (i % 1000 == 0) console.log(i + "/" + lines.length);

    // 一行を処理
    var line = lines[i];
    var cells = line.split(",");
    var no = cells.shift();
    var vals = [];
    for (var j = 0; j < cells.length; j++) {
      var index = j + 1;
      var v = cells[j];
      if (v == 0) continue; // 0のデータは省略できる
      var value = v / 255;      // データをスケーリング
      vals.push(index + ":" + value);
    }
    if (vals.length == 0) continue;
    var v_str = no + " " + vals.join(" ");
    var dat = v_str + "\n";
    // 書込
    fs.writeSync(f_svm, dat, null, "utf-8");
  }
  console.log("saved = " + file_svm);
}
```

プログラムを実行するには、以下のコマンドを入力します。

```
$ node csv2trainfile.js
```

プログラムを実行すると次のような画面になります。



プログラムを実行してできたファイル

SVM ファイルを学習させてモデルを作成する

学習データの SVM ファイルができたらモデルを作成しましょう。「node-svm」を利用して、プログラムを書いて学習させることも、もちろんできます。しかし、「node-svm」に「-g」オプションを付けてインストールしてあれば、コマンドラインから「node-svm」を利用することができます。

```
# node-svm で学習モデルを作る
$ node-svm train（入力 svm ファイル）（出力 model ファイル）
```

では、実際に使ってみましょう。以下は、「train-mini.svm」という学習データから、学習モデル「train-mini.model」を生成するコマンドです。

```
$ node-svm train train-mini.svm train-mini.model
```

すると、どんなパラメーターでデータを学習するのか、いくつか質問があります。ここでは、全てデフォルトで学習させてみましょう。質問を読まずに全部 [Enter] を押すとデフォルト設定となります。しばらく待っていると、「train-mini.model」というモデルデータが生成されます。「train-mini.svm」には、1000 個しかデータがありませんが、マシンの性能によっては学習には時間がかかることがあります。

無事に学習が終わると以下のような画面が表示されます。

SVM ファイルを学習したところ

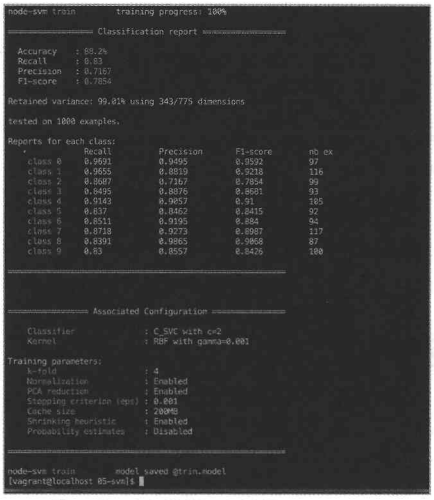
分類の正解率を確認しよう

テスト用の一万個の手書きデータが「t10k.svm」という名前で作成されています。せっかくモデルを作ったので、このファイルを利用して、上で学習したデータの認識率を確かめてみましょう。もちろん、「t10k.svm」にも、何番の数字かという情報が記されていますが、この値は数字の判定時には利用されませんが、分類の正解率を計算するために利用されます。

では、以下のコマンドを実行してみましょう。

```
$ node-svm evaluate train-mini.model t10k.svm
```

実行すると、一万件のデータを分類した結果が、以下のように表示されます。



一万件のデータを分類した結果

実行結果をよく見てみましょう。まず、分類してみた正解率 (Accuracy) ですが「88.2%」と表示されています。まずまずの成果ですね。そして、その下を見ると、class 0、class 1... とあり、各クラス（ここで言えばどの番号に分類するか）の確率が記されています。興味深いことに、「0」と「1」の正解率が非常に高くなっており、これを判別するのが用意であることがわかります。

ここでは、1000 件の学習データから作成した小さなモデルで試しましたから、次にフルで学習データを作成してみましょう。先ほどと同じ手順で、フルの六万件のデータでモデルを作成するには、以下のコマンドを実行します。

```
$ node-svm train train.svm train.model
```

ただし、筆者の Mac OS X(CPU:1.3GHz Intel Core i5/ メモリ :8GB) では、あえなくメモリ不足でエラーとなり、残念ながら、学習モデルを作成することが出来ませんでした。学習データの作り方を工夫したり、画像サイズを小さくしたりと、もう少し工夫できそうです。とは言え、ここでは、SVM を作成するとき、データを間引くようにしてみました。

この節のまとめ

- **この節では、サポートベクターマシン (SVM) についての説明をし、手書き数字のデータを学習させてみました。**
- **学習モデルを作成するところまでを作ったので、次の節では、これをもとにユーザーが書いた文字を認識するというプログラムを作ります。**

06

サポートベクターマシンで文字認識しよう(後編)

前節では、SVM を使って手書き文字の学習モデルを作るところまでを紹介しました。ここでは、HTML5 を利用してユーザーが入力した文字を認識するプログラムを作ってみます。(233B,83 字)

ここで学ぶポイント

- サポートベクターマシン(SVM)
- 文字認識のやり方
- HTML5のCanvas

ツールやライブラリの一覧

- Node.js
- 「node-svm」モジュール
- 「libsvm」ツール

node-svm の使い方

前節では、node-svm を利用して文字認識を行う学習モデルの作成方法を紹介しました。そこで、これを利用して文字認識を行うプログラムを作ってみます。はじめに、「node-svm」の簡単な使い方を紹介します。

以下のプログラムは、node-svm のサンプルコードを少し改良したもので、XOR 演算を SVM を利用して解いてみるというものになっています。XOR 演算とは論理演算 (あるいはビット演算) の一つです。もちろん、JavaScript のプログラムで解くのであれば「1 ^ 0」のようにして記述することで解くことができます。つまり、これは、SVM の使い方を試すだけのテストコードです。

● file: src/ch07/06-svm2/svm-test.js

```
// node-svm を使って認識

// 学習データ ---------- (※1)
var train_data = [
  // [[データ配列], クラス]
  [[0, 0], 0],
  [[0, 1], 1],
  [[1, 0], 1],
  [[1, 1], 0]
];
```

```
// モジュールの取り込み ---- (※2)
var svm = require('node-svm');

// オブジェクトの作成 ------ (※3)
var clf = new svm.CSVC();
clf.train(train_data)    // データの学習
   .done(doTest);        // テストを実行

function doTest() {
  // test1 --- (※4)
  var v = clf.predictSync([1, 0]);
  console.log("[1, 0] => " + v);

  // test2 --- (※5)
  clf.predict([1, 1]).then(function (predicted) {
        console.log("[1, 1] => " + predicted);
  });
}
```

このプログラムを実行するには、以下のコマンドをタイプします。

```
$ node svm-test.js
[1, 0] => 1
[1, 1] => 0
```

少し、プログラムを見てみましょう。プログラムの(※1)では、学習データを記述しています。ここでは、JavaScriptの配列を記述しています。各要素には、配列データと分類先のクラスを指定します。

プログラムの(※2)では、require()を利用して、node-svmモジュールを取り込みます。そして、プログラム(※3)では、SVMのオブジェクトを作成します。train()メソッドでデータを学習します。学習が完了した後、done()メソッドの引数に指定した処理が実行されます。

プログラムの(※4)以降では、テストデータを与えて、データから答えを予測させています。プログラムの(※4)では、同期的に予測を行うpredictSync()メソッドを利用して、データから値の予測を行います。プログラムの(※5)では、predict()メソッドであれば非同期に、predictSync()メソッドであれば同期的に処理を行います。

手書き文字認識ツールの制作

node-svmの基本的な使い方がわかったところで、手書き文字認識ツールを作成してみましょう。当然、node-svmを利用して、文字認識を行いますが、ただ画像を読み込んで判定するだけでは面白くありません。せっかくなので、ユーザーがその場で書き込んだ文字を認識させるというプログラムを作ってみます。

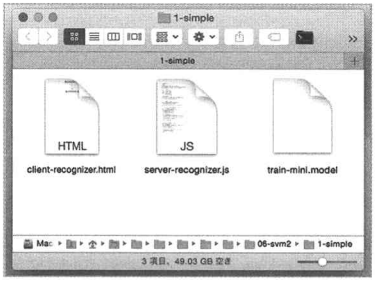
この手書きツールでは、Node.jsでHTTPサーバーを作成し、Webブラウザーからマウスで入力した数字を認識するというプログラムにしてみます。

手書き文字認識ツールを作る

2 を書いてみました

サーバーと通信しすぐに数字を予測する

プログラムの構成ですが、次のような内容にします。



ファイルの一覧

- server-recognize.js　　HTTP サーバーと画像認識 API
- client-recognize.html　　Web ブラウザーで操作するクライアント HTML
- train-mini.model 前節で作成した学習モデル

プログラムを実行するには、まず、HTTP サーバーを起動させて、Web ブラウザー上で手書き入力を行います。そして手書き入力したデータを、Ajax で HTTP サーバーに送信し推測を行います。

HTTP サーバーを起動するには、server-recognize.js を実行します。

```
$ node server-recognize.js
```

これで、サーバーが起動します。次いで、Web ブラウザーで、サーバーにアクセスしましょう。デフォルトでは、以下の URL にアクセスすることで手書き文字認識ツールを使うことができます。

```
[URL] http://localhost:1337
```

仮想サーバーで CentOS を利用している場合、1337 番のポートをホストマシンで使えるようポートフォワーディングの設定を行う必要があります。「Vagrantfile」に以下の設定を一行追加し、「vagrant reload」でマシンを再起動します。詳しくは、本章第 4 節の「人工無能」の項をご覧ください。

```
# 設定ファイル「Vagrantfile」に以下の一行を書き加える
config.vm.network "forwarded_port", guest: 1337, host: 1337
```

文字認識プログラムを見てみよう - HTTP サーバー側

それでは、まず、Node.js で作った HTTP サーバーである「server-recognizer.js」を見てみましょう。このプログラムでは、HTTP サーバーを起動して、Web ブラウザーで動く手書き入力クライアントをユーザーに公開、それから、Ajax 経由でアクセスされる手書きデータを元に数字を予測して返信します。

- file: src/ch07/06-svm2/1-simple/server-recognizer.js

```
// 文字認識サーバー
// 設定
var SERVER_PORT = 1337; // サーバーポート
var FILE_CLIENT = __dirname + "/client-recognizer.html";
var FILE_MODEL = __dirname + "/train-mini.model";

// モジュールの取り込み
var
  http = require('http'),
  URL  = require('url'),
```

```
  path = require('path'),
  fs   = require('fs'),
  svm  = require('node-svm');

// 学習モデルの読込 --- (※1)
var model_json = fs.readFileSync(FILE_MODEL,"utf-8");
var model_obj = JSON.parse(model_json, model_obj);
var clf = new svm.SVM({}, model_obj);

// サーバーを起動 --- (※2)
var svr = http.createServer(checkRequest);
svr.listen(SERVER_PORT, function(){
  console.log("サーバー起動しました");
  console.log("http://localhost:" + SERVER_PORT);
});

// サーバーにリクエストがあった時の処理 --- (※3)
function checkRequest(req, res) {
  var uri = URL.parse(req.url, true);
  var pathname = uri.pathname;
  // パス名で処理を分岐
  if (pathname == "/predict") {
    api_predict(req, res, uri);
  }
  else if (pathname == "/") {
    res.writeHead(200, {'Content-Type':'text/html'});
    res.end(fs.readFileSync(FILE_CLIENT, "utf-8"));
  } else {
    res.writeHead(404, {'Content-Type':'text/plain'});
    res.end("File not found");
  }
  console.log(pathname);
};

// APIへのリクエストを処理 --- (※4)
function api_predict(req, res, uri) {
  var p = uri.query.p;
  res.writeHead(200, {'Content-Type': 'text/plain'});
  var value = JSON.parse("[" + p + "]");
  for (var i in value) {
    value[i] = value[i] / 255;
  }
  console.log("value.length=" + value.length + "/" + 28*28);
  clf.predict(value).then(function (predicted) {
    console.log(predicted);
    res.end("" + predicted); // ----- (※5)
  });
}
```

プログラムの(※1)では、文字認識を行うために、まず、前節で作成した学習モデル「train-mini.model」を読み込み、node-svmのモジュールに設定しています。

プログラムの(※2)では、HTTPサーバーを起動します。サーバーが起動した後、ユーザーからアクセスがあると、プログラムの(※3)が実行されます。ここでは、ユーザーからどのようなアクセスがあったかを確認して、必要に応じてHTMLファイルを送出したり、Ajax経由でアクセスされた文字認識の結果を返します。そして、実際に文字認識を行っているのが(※4)です。ここは、肝心かなめの文字認識処理なのですが、コードを見ると驚くほど簡単です。サーバーから送られてきた手書きデータは、各ピクセルが0から255となっているので、範囲を0から1の間に変換した後、それを引数として、predict()メソッドを実行するだけです。そして、予測した数字をサーバーに戻します。ちなみに、予想した数字を保持している変

数「predicted」は数値ですが、サーバーの応答を出力する、end() メソッドには文字列を与える必要があります。そこで、プログラムの (※ 5) では、変数 predicted を文字列に変換した上で end() メソッドを呼んでいます。

文字認識プログラムを見てみよう - クライアント HTML 側

次に、手書きクライアントである HTML 側のコードを見ていきましょう。このコードでは、HTML5 の Canvas API を利用して、ユーザーからの手書き入力を受け付け、Ajax でサーバー側 API にアクセスします。

● file: src/ch07/06-svm2/1-simple/client-recognizer.html

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>文字認識 </title>
</head>
<body onload="init()" style="text-align:center;">
<h1> 数字手書き文字認識 </h1>
<div>

 中略

// 初期化 --- (※1)
function init() {
  // キャンバスのオブジェクトやコンテキストを取得
  main_cv = $("#main_cv");
  ctx = main_cv.getContext("2d");
  // キャンバスの位置とサイズを取得
  main_r = main_cv.getBoundingClientRect();
  // リサイズ処理用のイメージ
  back_cv = $("#back_cv");
  back_ctx = back_cv.getContext("2d");

  // マウスイベントの設定 --- (※2)
  main_cv.onmousedown = function (e) {
    isDown = true;
    s_pt = getXY(e);
    ctx.beginPath();
    ctx.lineWidth = border;
    ctx.lineCap = "round";
    ctx.strokeStyle = "white";
    ctx.moveTo(s_pt.x, s_pt.y);
  };
  main_cv.onmousemove = function (e) {
    if (!isDown) return;
    var pt = getXY(e);
    ctx.lineTo(pt.x, pt.y);
    ctx.stroke();
    s_pt = pt;
    ctx.beginPath();
    ctx.moveTo(pt.x, pt.y);
  };
  main_cv.onmouseup = function (e) {
    if (!isDown) return;
    isDown = false;
    ctx.closePath();
    recognize();
```

```
  };
  main_cv.onmouseout = main_cv.onmouseup;

  // 画面を白くする
  resetCanvas();
}

// マウスの座標を取得 --- (※4)
function getXY(e) {
  var x = e.clientX;
  var y = e.clientY;
  x -= main_r.left;
  y -= main_r.top;
  return {"x":x, "y":y};
}

// キャンバスの初期化
function resetCanvas() {
  ctx.clearRect(0, 0, main_cv.width, main_cv.height);
  ctx.fillStyle = 'black';
  ctx.fillRect(0, 0, main_cv.width, main_cv.height);
  ctx.beginPath();
  back_ctx.clearRect(0,0,back_cv.width,back_cv.height);
  back_ctx.beginPath();
  $("#result").innerHTML = "";
  x_min = main_cv.width;
  x_max = 0;
  y_min = main_cv.height;
  y_max = 0;
}

// コピー --- (※5)
function copyToBack() {
  back_ctx.fillStyle = "black";
  back_ctx.fillRect(0,0,28,28);
  back_ctx.drawImage(main_cv,
    0, 0, main_cv.width, main_cv.height,
    0, 0, 28, 28);
}

// パターンを作成する --- (※6)
function getPixelData() {
  //  画像を28x28にリサイズ
  copyToBack();
  // 画像イメージを取得 --- (※7)
  var img = back_ctx.getImageData(0, 0, 28, 28).data;
  var buf = [];
  console.log(img);
  for (var i = 0; i < 28 * 28; i++) {
    var k = i * 4;
    var r = img[k + 0]; // red
    var g = img[k + 1]; // green
    var b = img[k + 2]; // blue
    var a = img[k + 3]; // alpha
    var v = Math.floor((r + g + b) / 3.0); // ---- (※7a)
    buf.push(v);
  }
  return buf.join(",");
}
```

```
// 文字認識処理を実行 --- (※8)
function recognize() {
  // 手書き文字のピクセルを取得
  var txt = getPixelData();
  // サーバーへ送信 --- (※9)
  var uri = API_PREDICT + "p=" + txt;
  $ajax(uri, function (xhr, txt) {
    $("#result").innerHTML = "" + txt;
    console.log("predict=" + txt);
  });
}

// DOMを返す
function $(q) { return document.querySelector(q); }

// Ajax関数
function $ajax(url, callback) {
  var xhr = new XMLHttpRequest();
  xhr.open('GET', url, true);
  xhr.onreadystatechange = function() {
    if (xhr.readyState == 4) { //通信完了
      if (xhr.status == 200) { //HTTPステータス200
        callback(xhr, xhr.responseText);
      }
    }
  };
  xhr.send(''); // 通信を開始
  return xhr;
}
</script>
</body>
</html>
```

では、HTMLファイルの内容を確認していきましょう。HTMLの前半は、描画先の<canvas>やリセットボタンなどを配置しており、JavaScriptのプログラムは後半部分となっています。

プログラムの(※1)では、初期化処理を記述しています。<canvas>タグのDOM要素や、Canvasに描画するために必要なコンテキストを取得してます。HTML5のCanvas APIは強力で、さまざまな描画APIが用意されています。描画APIを利用するには、コンテキストを取得する必要があります。Canvasを画用紙に例えるなら、コンテキストはクレヨンと言うことができるでしょう。

プログラムの(※2)では、マウスイベントを設定しています。ユーザーがCanvasへ描画できるようにマウスイベントを利用して、描画処理を実装します。ここで利用するのは、次のイベントです。

マウスイベント	イベントの意味
onmousedown	マウスボタンを押した時の処理
onmousemove	マウスを移動した時の処理
onmouseup	マウスボタンを離した時の処理

Webブラウザーのマウスイベント一覧

<canvas>タグ上でマウスを動かすと、onmousemoveイベントがたくさん発生します。そのため、基本的な描画はこのonmousemoveイベントで行うのですが、マウスをクリックしている間のみ描画するようにします。そこで、onmousedownイベントで、フラグ変数isDownをtrueに設定し、onmouseupイベントでisDownをfalseにします。そして、onmousemoveイベントではフラグisDownを確認し、これがtrueの間のみ描画を行うという処理にしています。

描画を行うAPIは、描画コンテキストのメソッドである、moveTo()とlineTo()です。ここで利用した描画APIの意味は以下のようになります。

描画API	意味
ctx.moveTo(x1, y1)	起点(x1, y1)を設定
ctx.lineTo(x2, y2)	起点から(x2, y2)へ直線パスを指定し起点を移す
ctx.stroke()	指定したパスへ線を描画
ctx.fill()	指定したパスを塗りつぶす
ctx.beginPath()	既存の描画パスを破棄し新しくパスの指定を始める
ctx.closePath()	現在の描画パスを閉じる

HTML5の描画API

ちょっと注意が必要なのは、lineTo()メソッドでは、実際に線を描画するのではなく、描画パスを指定するだけという点です。実際にパスを画面に反映するのが、stroke()メソッドです。つまり、moveTo()メソッドで起点を指定、lineTo()メソッドで描画パスを指定、stroke()メソッドでパスの輪郭を描画するという手順になります。今回は利用していませんが、fill()メソッドを利用すると描画パスの内側を塗りつぶすことができます。

プログラムの(※5)から(※8)の部分で、ユーザーが描画した画面をサーバーに送信するという処理を記述しています。プログラムの(※5)では、ユーザーが描画を行うmain_cvを、小さなキャンバス(28x28ピクセル)のback_cvに縮小コピーを行います。ここにあるdrawImage()メソッドを利用すると、画像イメージの部分コピーや拡大縮小ができます。引数が多いので利用の際には、ちょっと混乱しますが、コピー元の情報、次いで、コピー先の情報と覚えると良いでしょう。

```
[書式] 画像のコピー
ctx.drawImage(
  src, sx, sy, sw, sh,
       dx, dy, dw, dh);
- src ... コピー元のCanvasやImage
- (sx, sy) ... コピー元の左上座標
- (sw, sh) ... コピー元のサイズ
- (dx, dy) ... コピー先の左上座標
- (dw, dh) ... コピー先のサイズ
```

次に、プログラムの(※6)では、画像を28x28ピクセルに縮小コピーした後で、画像イメージをJavaScriptの配列で取得します。それを行うのが、getImageData()メソッドです。このメソッドの実行結果のdataプロパティに画像のピクセルデータが入っています。ピクセルデータは、左上から右下へと順に入っており、1ピクセルに対して、赤・緑・青・透明度の4要素が設定されています。
次の図は、getImageData(4, 4)を実行した時の配列の様子を表しています。1ピクセルに対して4要素、それぞれ0から255までの値が得られます。

```
[例] getImageData(4, 4)の結果
+---------+---------+---------+---------+
| R G B A | R G B A | R G B A | R G B A |
+---------+---------+---------+---------+
| R G B A | R G B A | R G B A | R G B A |
+---------+---------+---------+---------+
| R G B A | R G B A | R G B A | R G B A |
+---------+---------+---------+---------+
| R G B A | R G B A | R G B A | R G B A |
+---------+---------+---------+---------+
```

しかし、今回は必要となる情報はカラーではなく色の濃淡のみです。そこで、プログラムの(※7a)の部分で、赤緑青の三色の平均を出し、色の濃淡を0から255までの数値として計算しています。

さて、プログラムの(※8)は、文字認識処理を実行する関数です。この関数では、手書きしたピクセルデータを取得して、これをAjaxを利用してサーバーに送信します。そして、サーバーからの戻り値が、数字の予測情報となります。プログラムの(※9)の部分で、Ajaxを利用してサーバーに対してデータを送信しています。$ajax()と$()関数をその後で用意しています。

誤認識の問題を解決しよう

実際にプログラムを実行した感想はいかがでしょうか。今回、1000件だけの手書きデータを学習したモデルでしたので、認識率は、それほど高くないと思ったのではないでしょうか。数字の種類や書き癖などの要因によって、誤認識してしまうことも多く感じます。

あまり認識率が高くない?・・・なぜ?!

いくつか解決策が考えられます。こうした機械学習では、ちょっとしたコツやチューニングにより認識率を飛躍的に高めることができます。

ここでは、正攻法で認識率を高めてみます。もともと六万件ある手書きデータを全部学習させるという方法です。とは言え、説明したようにnode-svmのツールを利用した学習では、六万件のデータを学習する前に、メモリ違反で途中で処理が落ちてしまいます。そこで、node-svmが利用している本家のツール「libsvm」とそれに付属しているツールを使います。

仮想マシンのCentOSでは、次の手順でlibsvmのインストールが可能です。まず、gitを利用して、GitHubでホスティングされているlibsvmのソースコードを手元にダウンロードしましょう。以下のコマンドを実行します。

```
$ cd ~/
$ git clone https://github.com/cjlin1/libsvm
```

次いで、libsvmをコンパイルしましょう。「gmake」コマンド一発でコンパイルが成功します。

```
$ gmake
```

コンパイル済みのツールが利用できるように、設定ファイル「~/.bashrc」でコマンドのエイリアスを定義しましょう。テキストエディターで「~/.bashrc」を開いて以下の3行を追加しましょう。

```
# libsvm のツール --- 以下のコマンドエイリアスを記述
alias svm-predict=/home/vagrant/libsvm/svm-predict
alias svm-scale=/home/vagrant/libsvm/svm-scale
alias svm-train=/home/vagrant/libsvm/svm-train
```

そして、前節で作成した学習データ「train.svm」を元にして学習モデル「train.model」を作成しましょう。以下のコマンドを実行します。

```
$ svm-train train.svm train.model
```

すると、やはり時間はかかりますが、しばらくすると、六万件のデータを学習し、「train.model」というファイルが作成されます。ちなみに、「node-svm train」コマンドで作成した「.model」ファイルはJSON形式であり、「svm-train」コマンドで作成した「.model」ファイルを、そのまま node-svm で利用することはできません。

次に、認識率を確かめるために、テストデータである「t10k.svm」で認識率を確認してみましょう。libsvm の「svm-predict」を利用します。

以下のコマンドを実行すると、文字認識の正解率を表示します。また、実際の判定結果を result.txt に保存します。

```
# 認識率の確認 （テストファイル）（学習モデル）（結果ファイル）
$ svm-predict t10k.svm train.model result.txt
Accuracy = 94.46% (9446/10000) (classification)
```

だいぶ改良されて、認識率が高くなっているのがわかると思います。

手書き文字認識ツールに組み込む

コマンド「svm-predict」を使うことで、データの予測を行うことができます。先ほど試したように「t10k.svm」と同じ書式、つまり、学習データを作成したときと同じ書式の SVM ファイルを作り、「svm-predict」を実行することで、判定結果を得ることができます。

```
［書式］SVM ファイル
<label> <index1>:<value1> <index2>:<value2> <index3>:<value3>...
```

ただし、上記の <label> の部分、どのクラスに分類されるかの情報は、もちろんわからないので、適当な値を書いておけば良いことになります。

それでは、この libsvm をプログラムに組み込んでみましょう。ここでは、node-svm を利用せず、libsvm の svm-predict をコマンドラインから実行するという方法で組み込んでみました。

改良版の手書き文字認識ツールでは、以下の三つのファイルを利用します。

- **server-recognizer-cmd.js**　サーバー側
- **client-recognizer.js**　クライアント側
- **train.model**　学習モデル

クライアント側のHTMLファイルは、変更なしです。先ほどのプロジェクトとまったく同じHTMLファイルを利用します。今回修正したのは、サーバー側のプログラムです。学習モデルは、libsvmの「svm-train」コマンドで作成したものです。

そこで、ここでは、修正を行ったサーバー側のプログラム「server-recognizer-cmd.js」だけを紹介します。

● file: src/ch07/06-svm2/2-full/server-recognizer-cmd.js

```
// 文字認識サーバー
// 設定
var SERVER_PORT = 1337; // サーバーポート
var FILE_CLIENT = __dirname + "/client-recognizer.html";
var FILE_MODEL = __dirname + "/train.model";
var SVM_PREDICT = "~/libsvm/svm-predict"; // svm-predict のパスを指定
var DIR_TEMP = __dirname;

// モジュールの取り込み
var
  http = require('http'),
  URL  = require('url'),
  path = require('path'),
  fs   = require('fs'),
  exec = require('child_process').exec;

// サーバーを起動
var svr = http.createServer(checkRequest);
svr.listen(SERVER_PORT, function(){
  console.log("サーバー起動しました");
  console.log("http://localhost:" + SERVER_PORT);
});

// サーバーにリクエストがあった時の処理
function checkRequest(req, res) {
  var uri = URL.parse(req.url, true);
  var pathname = uri.pathname;
  // パス名で処理を分岐
  if (pathname == "/predict") {
    api_predict(req, res, uri);
  }
  else if (pathname == "/") {
    res.writeHead(200, {'Content-Type':'text/html'});
    res.end(fs.readFileSync(FILE_CLIENT, "utf-8"));
  } else {
    res.writeHead(404, {'Content-Type':'text/plain'});
    res.end("File not found");
  }
  console.log(pathname);
};

// APIへのリクエストを処理
function api_predict(req, res, uri) {
  var p = uri.query.p;
  res.writeHead(200, {'Content-Type': 'text/plain'});
  var value = JSON.parse("[" + p + "]");
  var list = [];
  for (var i in value) {
    var v = value[i] / 255;
    if (v == 0) continue;
    list.push( (parseInt(i) + 1) + ":" + v );
  }
```

```
  // テスト用のデータを作成 ---- (※1)
  var testdata = "0 " + list.join(" ") + "\n";
  console.log(testdata);

  // 一時ファイルに保存する
  var r = Math.random();
  var t = (new Date()).getTime();
  var tmp_test = DIR_TEMP + "/test-" + t + "-" + r;
  var tmp_res  = DIR_TEMP + "/res-" + t + "-" + r;
  fs.writeFileSync(tmp_test, testdata, "utf-8");

  // コマンドを生成 ---- (※2)
  var cmd_a = [
    SVM_PREDICT,
    '"' + tmp_test + '"',
    '"' + FILE_MODEL  + '"',
    '"' + tmp_res + '"'
  ];
  var cmd = cmd_a.join(" ");
  console.log("*** cmd ***",cmd, "***");

  // コマンドを実行 --- (※3)
  exec(cmd, function (err, stdin, stdout) {
    if (err) {
      res.end("ERROR: exec commnad");
      return;
    }
    // 結果ファイルを開く
    var a = fs.readFileSync(tmp_res, "utf-8");
    console.log("predict>" + a);
    console.log(stdout);
    res.end("" + a);
    // 一時ファイルを削除
    fs.unlink(tmp_test);
    fs.unlink(tmp_res);
  });
}
```

プログラムを実行するには、先ほどと同じで、まずサーバーを起動します。

```
$ node server-recognizer-cmd.js
```

そして、Web ブラウザーで「http://localhost:1337/」にアクセスします。

先ほど誤認識していたデータも認識できるようになった

プログラムの基本部分は同じですが、node-svm を利用していた部分を削って、コマンドラインで libsvm のツール「svm-predict」を利用するように修正を行っています。

プログラムの (※ 1) の部分は、Web ブラウザー上で数字を描画したときに、ピクセルデータが送られて来るので、それを SVM ファイルの形式に加工して保存しています。プログラム (※ 2) では、svm-predict コマンドのパラメーターを生成します。それから、プログラム (※ 3) でコマンドを実行します。

この節のまとめ

- この節では、SVM を利用して手書き文字を認識するプログラムを作ってみました。
- 六万件の学習データを利用することで、認識率はかなり高くなりました。
- それでも間違えることがあります。
- 誤認識をする条件を考え、簡単な補正処理を行うなどの工夫を行って認識率を高めて下さい。

第8章

データの視覚化と応用

本章では、データの視覚化（ビジュアライゼーション）の方法について紹介します。JavaScriptは、Webブラウザと密接に結びついています。そのため、JavaScriptを使うのであれば、Webブラウザのレンダリング機能を利用した視覚化が可能です。

01

Google Chartsで簡単チャート作成

JavaScript で手軽にグラフを描画しようと思った時、最初に候補に挙がるのが Google Charts でしょう。手軽で使いやすく見た目が堅実です。ここではさまざまなチャートの表示の仕方を紹介します。

ここで学ぶポイント

- Google Chartsを利用した描画

ツールやライブラリの一覧

- Google Charts
- Node.js
- 「cheerio-httpcli」モジュール

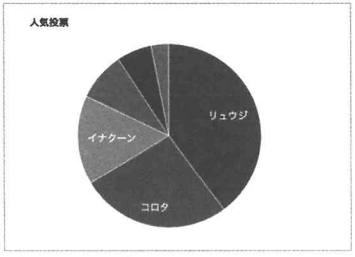
Google Charts とは何か？

Google Charts は、SVG を使ったグラフを表示するツールです。線グラフ、棒グラフ、円グラフ、ツリーマップなど、さまざまな種類のグラフを作成することができます。本家サイトは以下の URL になります。

```
Google Charts
https://developers.google.com/chart/
```

円グラフ（パイチャート）を描こう

簡単な例として、以下のような円グラフ（パイチャート）を描いてみましょう。ここでは、架空の人気投票の結果を表示するものにしてみました。



パイチャートを描画したところ

実際のプログラムは、以下の通りです。Web ブラウザーに HTML ファイルをドラッグ＆ドロップすると、前ページのようなグラフを描画することができます。

● file: src/ch08/01-gcharts/pichart.html

```
<html><head><meta chartset="utf-8">
<title> 円グラフ </title>
<!-- ライブラリの読み込み ======( ※ 1) -->
<script type="text/javascript" src="https://www.google.com/jsapi"></script>
<script type="text/javascript">

  // どのグラフを使うかを指定する ------ ( ※ 2)
  google.load('visualization', '1.0', {'packages':['corechart']});

  // グラフ API の読み込みが完了したら実行するようイベントを指定 ---- ( ※ 3)
  google.setOnLoadCallback(drawChart);

  // チャートの描画を行う --- ( ※ 4)
  function drawChart() {
    // データオブジェクト作成する --- ( ※ 5)
    var data = new google.visualization.DataTable();

    // データのカラムを指定 -------- ( ※ 6)
    data.addColumn('string', '人物 ');
    data.addColumn('number', '獲得票 ');

    // 実データを指定 ------------- ( ※ 7)
    data.addRows([
      ['リュウジ ',   51],
      ['コロタ ',     34],
      ['イナクーン ', 20],
      ['サラダー ',   11],
      ['イガタ ',      8],
      ['キョウダ ',    4],
    ]);

    // グラフのオプションを指定 ------- ( ※ 8)
    var opt = {
      'title': '人気投票 ',
      'width':  600,
      'height': 400,
      pieSliceText: 'label',
      legend: 'none'
    };
    // グラフを表示 ------------- ( ※ 9)
    var chart = new google.visualization.PieChart(
      document.getElementById('chart_div'));
    chart.draw(data, opt);
  }
</script>
</head><body>
  <div id="chart_div"></div>
</body></html>
```

それでは、HTML のコードを詳しく見ていきましょう。HTML の (※ 1) では、JavaScript のライブラリを読み込みを行っています。ここで、まず読み込んでいるのは、Google の「JSAPI API」です。この API を読み込むことで、プログラムの (※ 2) の部分にある、load() メソッドが利用できるようになります。このメソッドでは「Google Visualization ライブラリ」を読み込んでいます。

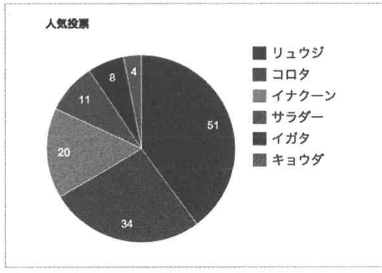
プログラム (※ 3) の部分では、Visualization ライブラリの読み込みが完了したときに実行するイベントを指定します。これにより、ライブラリが読み込まれたタイミングで、任意の処理を実行することができます。ここでは、ライブラリ読み込みと同時に、チャートの描画を行います。それを行っているのが、プログラムの (※ 4) の部分です。

プログラム (※ 5) の部分では、データオブジェクトを作成します。(※ 6) では、データのカラムを指定しています。円グラフでは、二つのカラムが必要です。そのために、addColumn() メソッドを使います。実際のデータは、プログラム (※ 7) の部分、addRows() メソッドで追加しますが、二次元配列変数で指定します。

プログラムの (※ 8) では、グラフのオプションを指定します。そして、プログラム (※ 9) の部分で、実際にグラフの描画を指定しています。このグラフのオプション指定ですが、width と height を指定することで、チャートのサイズを指定できます。また、pieSliceText に「label」を指定することで、円グラフの各項目に指定した文字列を表示することができます。ここを「percentage」や「value」にすることで、値のパーセンテージや実際の値を表示させることができます。

パイチャートのオプションを変えてみる

オプションを変えると、雰囲気の違う円グラフを描画することができます。以下は、オプションを少し変えただけですが、雰囲気が変わりました。



パイチャートのオプションを指定したところ

実際には、以下のような HTML コードで試しました。Web ブラウザーに以下の HTML をドラッグして実行してみてください。

● file: src/ch08/01-gcharts/pichart2.html

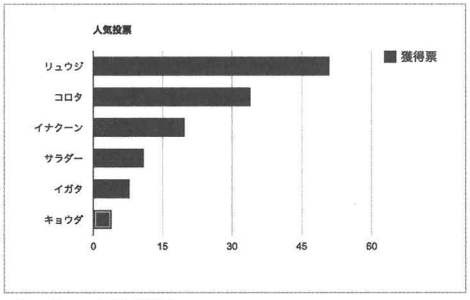
```
<html><head><meta chartset="utf-8">
<script type="text/javascript" src="https://www.google.com/jsapi"></script>
<script type="text/javascript">
  // ライブラリの読み込み
  google.load('visualization', '1.0', {'packages':['corechart']});
  google.setOnLoadCallback(drawChart);

  // チャートの描画を行う
  function drawChart() {

    // データを設定
    var data = new google.visualization.DataTable();
    data.addColumn('string', '人物');
    data.addColumn('number', '獲得票');
    data.addRows([
      ['リュウジ',   51],
      ['コロタ',     34],
```

```
      ['イナクーン', 20],
      ['サラダー',   11],
      ['イガタ',      8],
      ['キョウダ',    4],
    ]);

    // グラフのオプションを指定 ----- (※1)
    var opt = {
      'title': '人気投票',
      'width':  600,
      'height': 400,
      pieSliceText: 'value',  // グラフの中に示す値
      legend: {               // グラフの凡例の設定
        position: 'right',
        textStyle: {color: 'blue', fontSize: 16}
      }
    };
    // グラフを表示
    var chart = new google.visualization.PieChart(
      document.getElementById('chart_div'));
    chart.draw(data, opt);
  }
</script>
</head><body><div id="chart_div"></div></body></html>
```

明らかにわかる部分のコメントを少し外しましたが、異なる部分は、プログラムの(※1)の部分とグラフのオプションを変更しただけです。pieSliceTextで、グラフ中の中に示す値を、ラベルではなく、データの値を表示しています。また、legendを指定することで、グラフの凡例を表示するようにしました。

棒グラフ（バーチャート）を描こう

次に、基本的なグラフの一つ、棒グラフを描いていきます。円グラフと同じデータを棒グラフで描画すると以下のようになります。



バーチャートの描画

そして、プログラムは、以下の通りです。以下のHTMLファイルをWebブラウザーにドラッグすることでグラフを描画することができます。

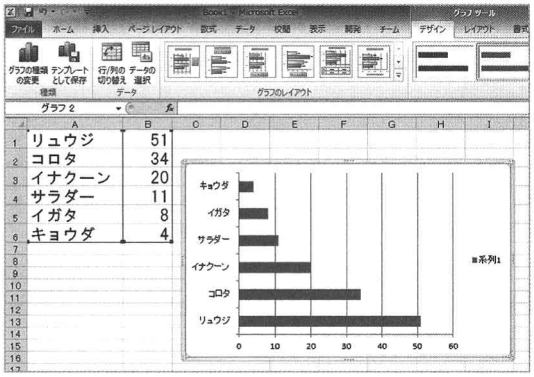
● file: src/ch08/01-gcharts/barchart.html

```
<html><head><meta chartset="utf-8">
<script type="text/javascript" src="https://www.google.com/jsapi"></script>
<script type="text/javascript">
  // ライブラリの読み込み ---- (※1)
  google.load('visualization', '1.0', {'packages':['corechart']});
  google.setOnLoadCallback(drawChart);

  // チャートの描画を行う
  function drawChart() {

    // データを設定
    var data = new google.visualization.DataTable();
    data.addColumn('string', '人物');
    data.addColumn('number', '獲得票');
    data.addRows([
      ['リュウジ',   51], ['コロタ',     34],
      ['イナクーン', 20], ['サラダー',   11],
      ['イガタ',      8], ['キョウダ',    4],
    ]);

    // グラフのオプションを指定
    var opt = {
      'title': '人気投票',
      'width':  600, 'height': 400,
    };
    // グラフを表示 ---- (※2)
    var chart = new google.visualization.BarChart(
      document.getElementById('chart_div'));
    chart.draw(data, opt);
  }
</script>
</head><body><div id="chart_div"></div></body></html>
```

プログラム的には、ライブラリの読み込み(※1)からデータの設定の部分まで、先ほどのパイチャートとまったく同じです。異なる部分と言えば、プログラムの(※2)の部分で、BarChartのオブジェクトを作成しています。

基本的にはExcelでグラフを作るのと同じ？！

パイチャートとバーチャートのプログラムを見て気づくのですが、プログラムの大半は、ライブラリを読み出すセットアップの部分と、実際のデータという二つの要素で成り立っています。そういえば、Excelでグラフを挿入する際も、二次元のシートを選択してグラフ挿入を選ぶだけですよね。つまり、項目数と実際の二次元のデータさえあれば、そこからいろいろなチャートが作れるということになります。



Excel でグラフを作る方法

ラインチャート

しかしながら、プログラムを書けば Excel 以上に柔軟で動的なグラフが作れます。ここでは、Yahoo! 天気から一週間分の予想気温を取得し、それを元にして、ラインチャートを描画してみます。

まずは、一週間分の予想気温をダウンロードするプログラムを作りましょう。以下は、Yahoo! 天気の RSS を読んで、[日付 , 最高気温 , 最低気温] の配列を保存するプログラムです。

● file: src/ch08/01-gcharts/get-temperature.js

```
// Yahoo! 天気予報の RSS から JSON ファイルを出力

// RSS のアドレス ----- (※1)
var RSS = "http://rss.weather.yahoo.co.jp/rss/days/4410.xml";
var SAVE_PATH = "temperature-data.js";

// モジュールを読む
var client = require('cheerio-httpcli');
var fs = require('fs');

// RSS をダウンロード ----- (※2)
client.fetch(RSS, {}, function(err, $, res) {
  if (err) { console.log("error"); return; }

  // 必要な項目を抽出して表示
  var res = [];
  $("item > title").each(function(idx) {
    var title = $(this).text();
    // 正規表現で日付、最高気温、最低気温を抽出 ---- (※3)
    var tm = title.match(/(\d+日).*?(\d+)℃ \/(\d+)℃/);
    if (!tm) return;
    var line  = [tm[1], parseInt(tm[2]), parseInt(tm[3])];
    res.push(line);
    console.log(line);
  });
```

```
  // 保存 ---- (※4)
  res.unshift(['日付','最高気温','最低気温']);
  var src = "var tempData = " + JSON.stringify(res);
  fs.writeFileSync(SAVE_PATH, src, "utf-8");
  console.log("ok!");
});
```

プログラムを実行するには、以下のコマンドを実行します。プログラムの中では「cheerio-httpcli」モジュールを利用していますので、「npm install cheerio-httpcli」でインストールしておきましょう。

```
$ node get-temperature.js
```

そして、プログラムを実行すると、天気予報を取得し「temperature-data.js」という名前でファイルをローカルに保存します。ファイルの中身は以下のようになります。

● file: src/ch08/01-gcharts/temperature-data.js

```
var tempData = [["日付","最高気温","最低気温"],["7日",24,19],["8日",24,20],["9日",24,19],["10日",29,21],["11日",30,22],["12日",29,22],["13日",29,22],["14日",30,22]]
```

少し、プログラムを見ていきましょう。プログラムの(※1)では、RSSのアドレスを指定しています。ここでは、Yahoo! 天気(東京)のRSSのURLを指定しています。プログラムの(※2)では、fetch()メソッドを利用して、ダウンロードを行います。「cheerio-httpcli」については、2章で詳しく紹介しています。プログラムの(※3)の部分では、RSSファイルから、任意の項目だけを抽出し、それに対して、正規表現で日付と最高気温・最低気温の三項目を取り出しています。

最後に、プログラムの(※4)では、抽出したデータを、JavaScriptの変数定義を行っているソースコードとして保存します。これによって、別のHTMLファイルからデータを手軽に読み込むことができます。

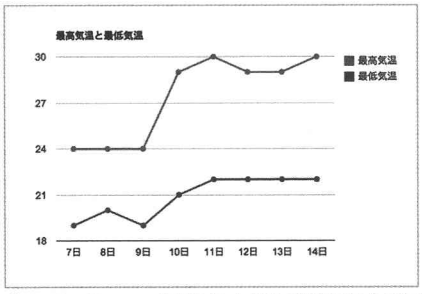
このデータを読み込んでグラフを作るHTMLファイルが以下になります。

● file: src/ch08/01-gcharts/temperature.html

```
<html><head><meta chartset="utf-8">
<!-- ライブラリの取り込み -->
<script type="text/javascript" src="https://www.google.com/jsapi"></script>
<!-- 気温データの取り込み -->
<script type="text/javascript" src="temperature-data.js"></script>

<script type="text/javascript">
  // ライブラリの読み込み ---- (※1)
  google.load('visualization', '1.0', {'packages':['corechart']});
  google.setOnLoadCallback(drawChart);

  // チャートの描画を行う
  function drawChart() {

    // データを設定 ----- (※2)
    var data = google.visualization.arrayToDataTable(tempData);

    // グラフのオプションを指定 ---- (※3)
    var options = {
      'title': '最高気温と最低気温',
      'width':  600, 'height': 400,
```

```
        'colors': ['red', 'blue'],
        'pointShape': 'circle',
        'pointSize': 5
      };
      // グラフを表示 ---- (※4)
      var chart = new google.visualization.LineChart(
        document.getElementById('chart_div'));
      chart.draw(data, options);
    }
</script>
</head><body><div id="chart_div"></div></body></html>
```

HTMLファイルをWebブラウザーにドラッグすると、次のようなグラフが描画されます。



Google Chartsを利用した折れ線グラフを描画した

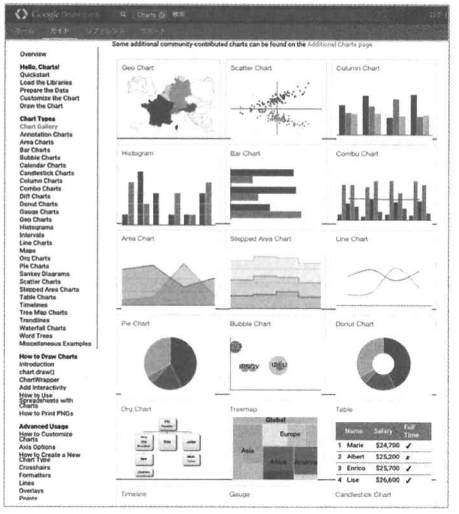
プログラムを見ていきましょう。プログラムの(※1)でライブラリを読み込んでいる部分は他のチャートを描画するところと同じです。しかし、そのちょっと上の部分、タグのところで、Googleのライブラリ以外に、「temperature-data.js」というJavaScriptファイルも読み込んでいるところに注目してください。これは、先ほど、Node.jsの「get-temperature.js」で生成したJavaScriptファイルです。続いて、プログラムの(※2)の部分で、データを設定します。先ほどのように、データオブジェクトを作成する方法ではなく、ここでは、arrayToDataTable()メソッドを利用してデータを設定しています。ここで出てくる「tempData」というのは、「temperature-data.js」の中で定義されている変数です。

プログラムの(※3)では、グラフのオプションを指定します。colorsプロパティを指定して、ラインチャートのラインの色を明示的に指定しています。一つ目のラインは最高気温ですがこれがred(赤色)になるように、また、2つ目のラインの最低気温がblue(青色)になるように指定しました。それから、pointShapeと、pointSizeプロパティを指定して、ラインチャートでデータの節の部分が見やすくなるように指定しました。そして、プログラムの(※4)でLineChartのオブジェクトを作成するよう指定して描画しています。

チャートの種類とマニュアル

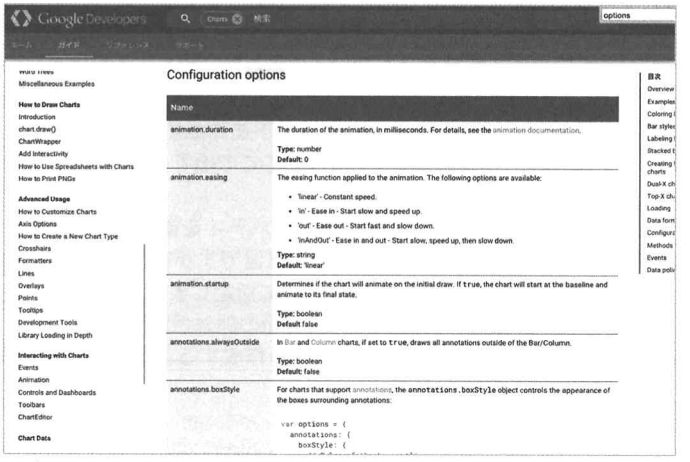
Google Chartsにはどんな種類のグラフが用意されているのでしょうか。その問いは、Google Chartsのトップページから「Chart Gallery」のページを見ると解決します。このページには、作成できるグラフの一覧が載っています。

```
Google Charts > ガイド > Chart Gallery
[URL] https://google-developers.appspot.com/chart/interactive/docs/gallery
```



チャートギャラリーのページ

グラフの種類を選んで、グラフの上のタイトルをクリックすると、実際のプログラム例と、詳しいオプションの説明ページが表示されます。オプションの説明自体は、英語ですが、それほど難しいことは書かれていません。また、実際にオプションを追加すると動作がすぐに反映されるものが多いので、試してみることができます。



オプションの説明

この節のまとめ

- ここでは、Google Charts を利用してグラフを描画する方法を紹介しました。
- 紹介したプログラムを見るとわかりますが、、このライブラリを利用すれば、手軽に見栄えが良いグラフを描画できます。
- このように収集したデータを視覚化することで、たくさんの発見があります。

02

D3.jsで自由度の高いチャート作成

D3.jsはデータに基づいてドキュメントを操作するためのJavaScriptライブラリです。非常に自由度が高く、表現力豊かなグラフを描画することができます。ここでは、D3.jsの基本的な使い方を紹介します。

ここで学ぶポイント	ツールやライブラリの一覧
●D3.jsについて	●D3.js

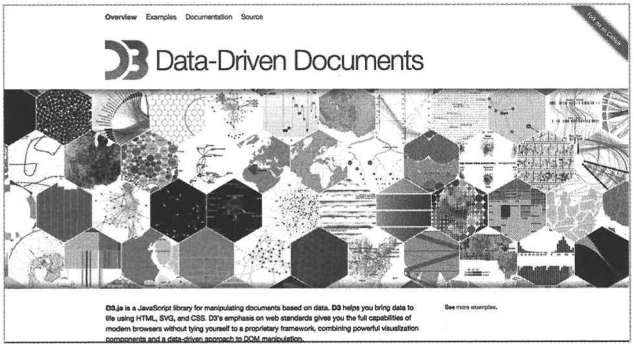
D3.js — データ駆動ドキュメント生成ライブラリ

D3.jsはきわめて表現力の高いグラフ・ライブラリです。D3.jsという名前は「Data-Driven Documents」(頭文字のDが三つあるのでD3)に由来しています。データ駆動のドキュメント生成ライブラリという意味です。D3.jsはデータを視覚化する強力な機能を備えています。また、「ライブラリの出来が良い」と評価も高いので一度使ってみる価値は十分にあります。

前節でみたGoogle Chartsなど一般的なグラフ描画ライブラリは、データを与えると、それを元にグラフをそのまま描画するところまで、自動でやってくれます。しかし、D3.jsは描画座標の計算は行いますが、実際に図を描画することはしません。図を描画する処理はユーザー自身が記述する必要があります。そう聞くと、ちょっと面倒くさいと思うかもしれません。しかし、逆に言えば、与えられた座標をどのように描画するのかを自由に決めることができるので、凝った描画処理を行うことができるのです。多くのグラフ描画ライブラリで、デザインをカスタマイズするのは非常に面倒です。その点、D3.jsは自由度が高く、凝ったデザインのグラフを作ることができます。

D3.jsのWebサイトは以下の場所にあります。

```
D3.js
[URL] http://d3js.org/
```



D3.js の Web サイト

D3.js をセットアップしよう

D3.js を利用するには、上記の D3.js のページから、ライブラリをダウンロードして、それをリンクするのが一つの方法です。また、CDN（コンテンツ配信ネットワーク）で配布されているものを使うという方法もあります。

CDN を使う方法が一番簡単なので初めに紹介します。CDN で配布されている D3.js のライブラリを取り込むように、その URL を HTML ファイルのタグに書き込みます。

```
<!DOCTYPE html>
<html><head>
<!-- D3.js を取り込む（CDN を利用する場合） -->
<script
  type="text/javascript"
  src="https://cdnjs.cloudflare.com/ajax/libs/d3/3.5.5/d3.min.js"
  charset="utf-8"></script>
</head><body>
</body></html>
```

D3.js のページからライブラリをダウンロードした場合には、HTML ファイルと同じディレクトリに、アーカイブに含まれるライブラリファイルの「d3.min.js」をコピーし、上記の D3 の URL を以下のように修正します。

```
<!DOCTYPE html>
<html><head>
<!-- D3.js を取り込む（同じフォルダにコピーした場合） -->
<script
  type="text/javascript"
  src="./d3.min.js"
  charset="utf-8"></script>
</head><body>
</body></html>
```

棒グラフ（バーチャート）の作成

それでは、D3.js を利用して、棒グラフ（バーチャート）を描画してみましょう。まずは、データを単純に描画するプログラムから作ってみます。

● file: src/ch08/02-d3js/bar1.html

```
<!DOCTYPE html>
<html><head><meta charset="utf-8"></head><body>
<!-- D3.jsを取り込み -->
<script type="text/javascript" src="./d3.min.js"></script>
<script>
  // データの一覧 ---- (※1)
  var dataset = [30, 25, 44, 21, 30, 50];

  // 描画領域を作る --- (※2)
  var svg_width = 600, svg_height = 400;
  var bar_h = svg_height / dataset.length;
  var bar_w = svg_width / 50;
  var svg = d3.select("body")
    .append("svg")
    .attr("width", svg_width)
    .attr("height", svg_height);

  // グラフを描画する --- (※3)
  svg.selectAll("rect")
    .data(dataset)      // 描画データを与える    ----- (※4)
    .enter()            // これ以降で描画を指定 ----- (※5)
    .append("rect")     // 何を追加するかを指定 ----- (※6)
    .attr({             // 追加する際の属性を指定 --- (※7)
      x:      0,
      y:      function(d, i) { return i * bar_h; },
      width:  function (d) { return d * bar_w; },
      height: (bar_h - 5),
      fill:   "blue"
    });
</script>
</body></html>
```

このプログラムは、HTML ファイルなので、Web ブラウザーに「bar1.html」をドラッグすると実行することができます。以下のように表示されるでしょう。

棒グラフを描画したところ

実際のプログラムを見ていきましょう。ポイントとなる部分を確認しておくと、まず、グラフを描画するために、D3.js の機能を利用していることです。ここでは、SVG 要素を操作してグラフを描画しています

が、それほど複雑なコードを書いていません。また、d3 ライブラリを利用した、DOM 操作が何カ所かで行われています。そして、グラフを描画するに当たって、繰り返し構文の for などを記述する必要がないこともポイントでしょう。

では一つずつ見ていきましょう。まずプログラムの (※ 1) でグラフを描画するためのデータを与えています。この値を元にしてグラフを描画します。

次に、プログラムの (※ 2) の部分で、描画領域を作成します。ここでは、SVG を利用して描画しますが、タグを作成するために、d3 のライブラリを利用します。select() メソッドでタグを選択します。そして、続く、append() メソッドでタグを作成します。そして、attr() メソッドで、作成したタグの属性を指定します。ここでは、width と height の属性を指定しています。ちなみに、ここでは、attr() メソッドを二回メソッドチェーンでつなげて指定していますが、attr() メソッドに、JavaScript のオブジェクトを指定して一気に設定する方法もあります。この部分を次のように書き換えることができます。

```
var svg = d3.select("body")
  .append("svg")
  .attr({
    "width" : svg_width,
    "height": svg_height
  });
```

プログラムの (※ 3) の部分で、グラフを描画します。ここが D3.js の本領が発揮されている部分です。グラフを描画するために、タグを作成しているのですが、selectAll() メソッドを利用して、既存のタグを選択します。続く、(※ 4) の data() メソッドで、ここで描画するデータを設定します。そして、(※ 5) の enter() メソッド以降の部分で、どのような描画を行うのかを指定します。先の data() メソッドで与えたデータ数だけ、enter() メソッド以降の部分で指定した処理が繰り返し実行される仕組みとなっています。

ここで、データの数だけ個々に実行される描画ですが、プログラムの (※ 6) の append() メソッドで指定している通り複数のタグが追加されます。その要素には、その後 (※ 7) の attr() メソッドで指定した値が適用されます。

この、プログラム (※ 7) の attr() メソッドがポイントです。実際に数値を与えている部分もあれば、関数を指定している部分もあります。関数を与えている部分は、(※ 4) の data() メソッドで指定したデータが引数として与えられ、その値を関数により計算することができるようになっているのです。

この関数には２つの引数を指定できるようになっていますが、第一引数の d がデータの値、第二引数の i が何番目の値かを示すインデックスとなっています。この時、インデックスは、JavaScript の配列と同じく 0 起点となっています。例えば、ここでタグの位置 (y プロパティ) や幅 (width プロパティ) というのは、棒グラフの各バーの値を表示する部分ということです。ここの部分を見てみると以下のように指定しています。

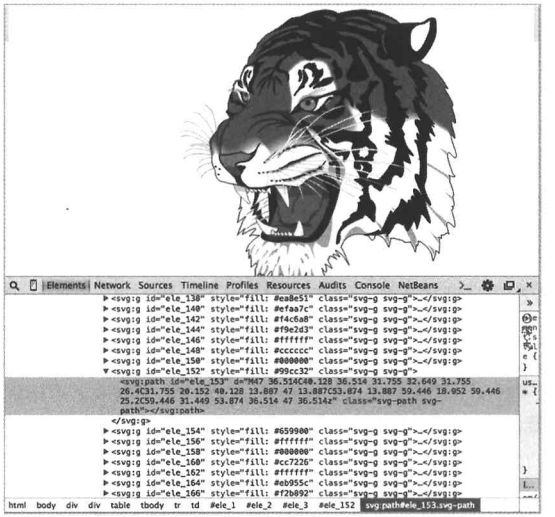
```
var svg_width = 600, svg_height = 400;
var bar_w = svg_width / 50;
var bar_h = svg_height / dataset.length;

中略

  y : function (d, i) { // バーの位置を指定
    return return i * bar_h;
  },
  width : function (d) { // バーの幅を指定
    return d * bar_w;
  },
```

つまり、画面上でバーがどこに来るのか、またその幅が何ピクセルになるのかを計算しているのです。

SVGについて

ところで、ここではタグを利用して描画を行っていましたが、肝心のSVGについてあまり詳しくない方もいるかもしれません。SVG(Scalable Vector Graphics)とは、XMLをベースとした二次元ベクターイメージ描画用の画像形式です。ベクターイメージというのは、ピクセル単位の描画を行うPNGやJPEGなどの画像形式と違い、画像上の対象がすべて座標で配置されます。そのため、拡大縮小しても、画像がぼやけたり劣化することがないのが特徴です。

ただし、当然ですがSVGをサポートしていないブラウザーでは見ることができません。最近のHTML5をサポートしているブラウザーでは、大抵SVGがサポートされていますが、古いブラウザー（IE8以前、また、Androidのブラウザー2.x）では表示することができません。



SVGで描画されたトラ

棒グラフの描画でスケールを自動計算させよう

さて、先ほどのサンプルでは、棒グラフの幅(ピクセル数)を、わざわざ計算していましたが、D3.jsには、自動的にピクセル数を計算する機能が備わっています。その機能を利用して、棒グラフの描画を書き換えたのが以下のプログラムです。

● file: src/ch08/02-d3js/bar2.html

```
<!DOCTYPE html>
<html><head><meta charset="utf-8"></head><body>
<!-- D3.jsを取り込み -->
<script type="text/javascript" src="./d3.min.js"></script>
<script>
  // データの一覧
  var dataset = [30, 25, 44, 21, 30, 50];

  // 描画領域を作る
  var svg_width = 600, svg_height = 400;
  var svg = d3.select("body")
    .append("svg")
```

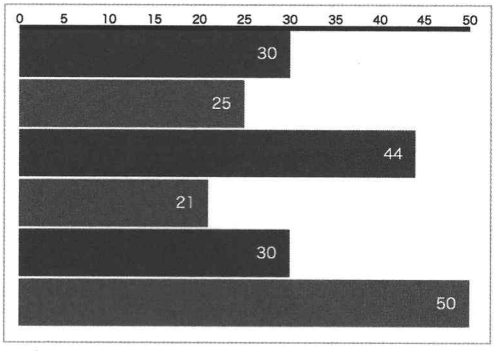
```
    .attr( {"width":svg_width, "height":svg_height});

  // スケール自動計算用 ---- (※1)
  var x_scale = d3.scale.linear()
    .domain([0, d3.max(dataset)]) // データの幅        ---- (※2)
    .range([0, svg_width]);       // 実際の描画サイズ ---- (※3)

  // バーの高さを計算
  var bar_h = svg_height / dataset.length;

  // グラフを描画する
  svg.selectAll("rect")
    .data(dataset)
    .enter()
    .append("rect")
    .attr({
      x      : 0,
      width  : function(d, i) { return x_scale(d); }, // ----(※4)
      y      : function(d, i) { return i * bar_h; },
      height : bar_h - 5,
      fill   : "blue"
    });
</script>
</body></html>
```

Web ブラウザーに「bar2.html」をドラッグすると実行することができます。プログラムは異なりますが、見た目には、先ほどとまったく同じグラフになります。

プログラムのポイントとなる部分だけ押さえてみましょう。スケールを自動計算する機能を準備しているのがプログラムの (※ 1) の部分です。d3.scale.liner() と指定することにより、スケールの自動計算オブジェクトを取得しています。ここでは、実際の入力データを (※ 2) の domain() メソッドにて指定し、(※ 3) の range() メソッドにて、それをどの描画範囲に描画するかを指定しています。このように指定したら、プログラムの (※ 4) の部分で、x_scale(データの値) と指定することで、描画用の値を返すようになります。

目盛り付き棒グラフを描画しよう

次に、棒グラフのサンプルを改良して、棒グラフに目盛り軸 (Axis) を付けてみましょう。D3.js では、手軽に軸を付けることができます。そこで、以下のような目盛り付きの棒グラフを描画してみたいと思います。



目盛り付き棒グラフを描画

以下が、今回のグラフを描画するためのプログラムです。

● file: src/ch08/02-d3js/bar3.html

```
<!DOCTYPE html>
<html><head><meta charset="utf-8"></head><body>
<!-- D3.jsを取り込み -->
<script type="text/javascript" src="./d3.min.js"></script>
<script>
  // データの一覧
  var dataset = [30, 25, 44, 21, 30, 50];

  // 画面サイズの指定
  var gr_w = 600; // グラフの幅
  var gr_h = 400; // グラフの高さ
  var margin = {top:40, right:20, bottom:30, left:20};
  var bar_h = gr_h / dataset.length;

  // スケールの指定 --- (※1)
  var x_scale = d3.scale.linear()
    .domain([0, d3.max(dataset)])
    .range([0, gr_w]);

  // Axisの指定 --- (※2)
  var x_axis = d3.svg.axis()
    .scale(x_scale)  // スケール情報
    .orient("top");  // 目盛りを配置する場所

  // 描画領域を作る
  var svg = d3.select("body")
    .append("svg")
    .attr("width",  gr_w + margin.left + margin.right)
    .attr("height", gr_h + margin.top + margin.bottom);

  // グラフを描画 ---- (※3)
  var bar = svg.selectAll(".bar")
    .data(dataset)
    .enter();

  // バーの部分を生成 ---- (※4)
  bar.append("rect")
    .attr({
      x:      margin.left,
      width:  function(d, i) { return x_scale(d); },
      y:      function(d, i) { return margin.top + i * bar_h;  },
      height: function(d, i) { return bar_h - 4; },
      fill:   function(d, i) { return (i % 2) ? 'red' : 'blue'; }
    });

  // ラベルテキスト(バー数値)の部分を生成 ------ (※5)
  bar.append("text")
    .text(function(d) { return d; })
    .style({
      "font-size":"20px",
      "text-anchor":"middle"
    })
    .attr({
      x: function(d, i){ return x_scale(d) - 10; },
      y: function(d, i){ return margin.top + i * bar_h + bar_h / 2 + 5; },
      fill: "white"
```

```
    });

  // Axisを追加 ------ (※6)
  svg.append("g")
    .attr('class', 'axis')
    .attr('transform', 'translate(' + margin.left + "," + margin.top + ')')
    .style({'font-size':'15px'})
    .call(x_axis);

</script>
</body></html>
```

プログラムを実行するには、Webブラウザーに上記のHTMLファイルをドラッグしてください。

さて、プログラムを見ていきましょう。目盛り軸(Axis)を表示させるには、プログラムの(※2)の部分d3.svg.axis()を利用してオブジェクトを作成します。このとき、(※1)のスケール指定で作成したd3.scale.linear()オブジェクトを利用できます。このオブジェクトは、既に実データの範囲(domain)と描画範囲(range)の情報を持っているからです。そして、(※6)の部分で、目盛り軸を実際に描画します。

実際に棒グラフを描画している部分についても見てみましょう。プログラムの(※3)以降の部分で描画しています。今回は、棒グラフのバーと、バーの上に重ねるラベルの2つを描画しています。そこで、まず、(※4)の部分で、バーの部分の描画方法を指定し、(※5)でラベル部分の描画方法を指定しています。

折れ線グラフ（ラインチャート）を描画しよう

棒グラフの作成で、D3.jsについて理解が深まったので、一気に折れ線グラフの描画に挑戦してみましょう。D3.jsには、CSVファイルなどのデータを読み込む機能もあるので、Node.jsでWebサーバーを起動しつつ、7章で作成した2014年の気温グラフを読み込んで描画してみたいと思います。ただし、D3.jsでは、デフォルトでCSVファイルのヘッダー行を認識しますので、CSVにヘッダーを追加し、次のようなCSVファイルを用意しました。

```
date,value
2014/1/1,9.6
2014/1/2,7.3
2014/1/3,5.9
2014/1/4,6.5
2014/1/5,5.4
2014/1/6,5.3
2014/1/7,5.5
...
```

まずは、Node.jsでWebサーバーを起動するプログラムから見ていきましょう。

● file: src/ch08/02-d3js/chart-server.js

```
// HTTPサーバーを作る
//----------------------------
// 設定
var SERVER_PORT = 1337; // サーバーポート
var FILE_DEFAULT = "/line.html";
```

```
// モジュールの取り込み
var
  http = require('http'),
  URL  = require('url'),
  path = require('path'),
  fs   = require('fs');

// サーバーを起動
var svr = http.createServer(checkRequest);
svr.listen(SERVER_PORT, function(){
  console.log("サーバー起動しました");
  console.log("http://localhost:" + SERVER_PORT);
});

// サーバーにリクエストがあった時の処理
function checkRequest(req, res) {
  var uri = URL.parse(req.url, true);
  var pathname = uri.pathname;
  if (pathname == "/") pathname = FILE_DEFAULT;
  console.log(pathname);

  // ファイルの存在確認
  var filename = path.join(__dirname, pathname);
  if (!fs.existsSync(filename)) {
    res.writeHead(404, {'Content-Type':'text/html'});
    res.end("404 file not found");
    return;
  }

  // ディレクトリであればエラーにする
  var stat = fs.statSync(filename);
  if (stat && stat.isDirectory()) {
    res.writeHead(403, {'Content-Type':'text/html'});
    res.end("403");
    return;
  }

  // ファイルを送出
  res.writeHead(200, {'Content-Type':'text/html'});
  res.end(fs.readFileSync(filename, "utf-8"));
}
```

プログラムを実行するには、コマンドラインから以下のコマンドを入力します。

```
$ node chart-server.js
```

続いて、折れ線グラフを描画するHTMLは以下の通りです。

● file: src/ch08/02-d3js/line.html

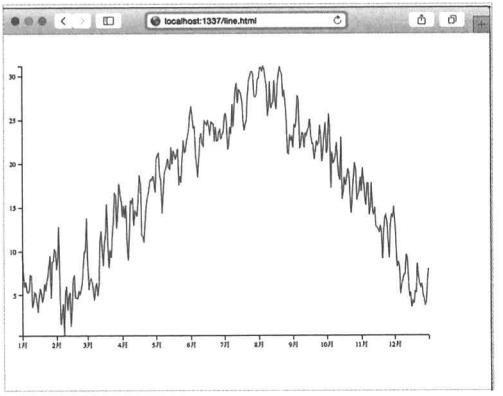
```
<!DOCTYPE html>
<html><head><meta charset="utf-8"></head><body>
<!-- D3.jsを取り込み -->
<script type="text/javascript" src="./d3.min.js"></script>
<!-- 描画スタイルの指定 -->
<style>
body { font-size: 10px; }
.axis path,
.axis line {
  fill: none;
```

```
  stroke: black;
  shape-rendering: crispEdges;
}
.line {
  fill:none;
  stroke:steelblue;
  stroke-width: 1.5px;
}
</style>
<script>
  // データファイル名の指定
  var CSVFILE = "./2014kion.csv";
  // 日付形式の指定 ---- (※1)
  var parseDate = d3.time.format("%Y/%m/%d").parse;

  // 画面サイズの指定
  var gr_w = 600; // グラフの幅
  var gr_h = 400; // グラフの高さ
  var margin = {top:40, right:20, bottom:50, left:20};

  // スケールの指定 ---- (※2)
  var x = d3.time.scale().range([0, gr_w]);
  var y = d3.scale.linear().range([gr_h, 0]);

  // Axis の指定 ---- (※3)
  var x_axis = d3.svg.axis()
    .scale(x)
    .orient("bottom")
    .tickFormat(function(d) {
      return (d.getMonth()+1) + "月";
    });
  var y_axis = d3.svg.axis().scale(y).orient("left").ticks(5);

  // SVG タグを追加 ---- (※4)
  var svg = d3.select("body")
    .append("svg")
      .attr("width", gr_w + margin.left + margin.right)
      .attr("height", gr_h + margin.top + margin.bottom)
    .append("g")
      .attr("transform",
        "translate(" + margin.left + "," + margin.top + ")");

  // 折れ線を作成 ---- (※5)
  var line = d3.svg.line()
    .x(function(d){ return x(d.date); })
    .y(function(d){ return y(d.value); });

  // データファイルを読み込む ---- (※6)
  d3.csv(CSVFILE, function(err, data) {
    if (err) {
      alert("データの読込エラー"); return;
    }
    // 日付形式などを変換 ---- (※7)
    data.forEach(function(d) {
      d.date = parseDate(d.date);
      d.value = parseFloat(d.value);
    });
    console.log(data);

    // スケールの範囲を指定
    x.domain(d3.extent(data, function(d) { return d.date; }));
```

```
    y.domain([
      d3.min(data, function(d) { return d.value; }),
      d3.max(data, function(d) { return d.value; })
    ]);

    // 折れ線を描画 ---- (※8)
    svg.append("path")
      .datum(data)
      .attr("class", "line")
      .attr("d", line);

    // 目盛り軸を追加
    svg.append("g")
      .attr("class", "y axis")
      .call(y_axis);
    svg.append("g")
      .attr("class", "x axis")
      .attr("transform", "translate(0," + gr_h + ")")
      .call(x_axis);
  });
</script>
</body></html>
```

そして、Web ブラウザーで次の URL にアクセスしましょう。すると、ブラウザー上に折れ線グラフが描画されます。

```
http://localhost:1337/line.html
```



線グラフを描画したところ

では、プログラムを確認してみましょう。今回の HTML では、JavaScript より前の部分でタグを用いて、CSS で描画スタイルの指定を行っています。D3.js でグラフを描くと、動的に SVG が生成されます。それで、SVG の描画スタイルも CSS で指定することができます。

プログラム (※ 1) では、日付形式の指定を行っています。これで「年 / 月 / 日」形式で書かれている日付を認識することができます。今回の気温データの CSV には、日付が書かれているので、このフォーマットで日付を読み込みます。

プログラム (※ 2) では、スケールの指定を行います。注目したいのは、今回の X 軸は日付なので、d3.time.scale を利用している点です。また、Y 軸は気温で下側が 0 で上に行くにしたがって値が上がります。しかし、SVG の座標体系では、左上が (0,0) であり、右下に行くにしたがって値が上がっていきます。

そのため、グラフを描画するには不便なのです。そこで、d3.scale.linear の range() メソッドで、[描画高さ ,0] のように指定すると、Y 軸の座標系を入れ替えることができます。

プログラム (※ 3) は、目盛り軸のオブジェクトを取得し、設定をしています。tickFormat() メソッドに関数を指定することで、目盛りのラベルに、任意の文字列を生成することができます。この関数を指定しない時には、英語で月名が表示されます。

プログラム (※ 4) では SVG タグを生成します。ちなみに、ここでは、タグを追加していますが、その中で transform 属性を指定しています。これは、グラフ周囲の余白（マージン）分をずらすことで、座標系の計算を単純にする狙いがあります。

プログラム (※ 5) では、折れ線を描画するオブジェクトを取得しています。そして、X 軸、Y 軸に、データのどの値を利用するかを指定します。

プログラム (※ 6) では、CSV ファイルの読み込みを行います。CSV の読み込みは、d3.csv() を呼ぶだけと、表示簡単なものになっています。また、ヘッダー付きの CSV を読み込んだ場合、CSV に記述されたヘッダーを利用して各行が読み込まれます。

今回読み込んだ CSV データには「date,value」というヘッダー行がありましたので、読み込まれると、次のような JavaScript のオブジェクトに変換されます。

```
[
  { date: "2014/01/01", value: "9.6" },
  { date: "2014/01/02", value: "7.3" },
  { date: "2014/01/02", value: "5.9" },
  ...
]
```

プログラム (※ 7) では、日付形式などを変換しています。特に、冒頭プログラム (※ 1) で定義した日付パーサーを利用して、日付を Date オブジェクトに変換します。また、value も実数に変換しています。

プログラム (※ 8) では、折れ線を指定します。折れ線は SVG のタグを利用して描画します。それで、データを設定するのに、datum() メソッドを利用します。

この節のまとめ

- **本節では、D3.js ライブラリの使い方を紹介しました。**
- **D3.js で検索すると魅力的なグラフをたくさん見つけることができます。**
- **ただしデータを与えるだけで描画してくれる訳ではないので、すぐにグラフを描画したい場合には、他のグラフ描画ライブラリを使うなどして使い分けると良いでしょう。**

03

D3.jsで地図を描画しよう

D3.js と TopoJSON を使うと、手軽に地図を描画したり、その上に視覚的にデータを描画できます。ここでは、日本地図を表示するプログラム、また、データに応じて地図を色分けする方法を紹介します。

ここで学ぶポイント

- 地図をブラウザー上に表示する方法
- 地図をデータに応じて色分けする方法

ツールやライブラリの一覧

- D3.js
- TopoJSON
- Node.js

地図情報を描画する

さて、地図上に、さまざまな情報を重ね合わせたいことがあると思います。そんなとき、使いたいのが、D3.js と TopoJSON です。地図データを元にして簡単に地図を描画してくれます。今回は、D3.js と TopoJSON を利用する方法を紹介します。

なお、地図上に情報を表示するには、Google Charts の機能である「Visualization:Geomap」も利用できます。データを表示したり、特定の地域にマウスを持って行くと、ポップアップするなどの機能が備わっています。Google Charts を利用したグラフも、手軽で良いものです。

TopoJSON で地図データを表示するまで

それでは、D3.js と TopoJSON で地図データを表示するプログラムを作っていきましょう。まずは、地図データを入手するところから始めなくてはなりません。また、地図データのうち、使いたい部分を切り取って、TopoJSON 形式に変換するという作業も必要になります。

改めて必要な作業をまとめてみます。

- **地図データ (Shape 形式) をダウンロード**
- **地図データから任意の部分を取り出す (GeoJSON 形式)**
- **TopoJSON のツールを使ってデータを変換 (TopoJSON 形式)**
- **D3.js と TopoJSON プラグインを使ってブラウザー上に表示**

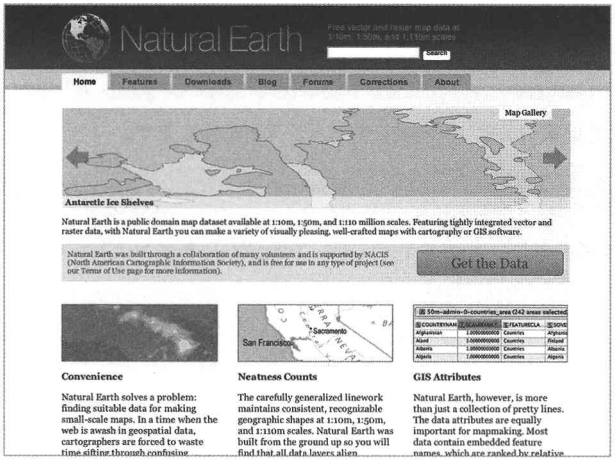
では、一つずつ順を追って作業していきましょう。

地図データをダウンロードしよう

地図データは、Natural Earth で公開されています。Natural Earth は、パブリックドメインの世界地図データを提供するサイトです。

人文ベクターデータ (Cultural Vector Data) や、地理ベクターデータ (Physical Vector Data)、ラスタデータ (衛星画像に基づくデータ) など、いろいろなデータが配布されています。

```
Natural Earth
[URL] http://www.naturalearthdata.com/
```



Natual Earth の Web サイト

さて、今回、TopoJSON で地図を描画するためにダウンロードするのは、都道府県が利用できるデータです。次の手順で開いていくとダウンロードできます。「Dwonloads タブ> Large scale data, 1:10m > Cultural > Download states and provinces」のデータです。

以下に挙げるのはデータファイルへの直リンクの URL です。原稿執筆時点で 13.9MB ありました。

（注意）URL の中に http と出てくるので誤植で間違っているのではないかと思うかもしれませんが、これが正確な URL です。

```
Download states and provinces:
[URL] http://www.naturalearthdata.com/http//www.naturalearthdata.com/
download/10m/cultural/ne_10m_admin_1_states_provinces.zip
```

ZIP ファイルなので、解凍して利用します。

データ形式の変換

それでは、ダウンロードしたShape形式のデータを、TopoJSONで使えるデータ形式に変換しましょう。ちなみに、なぜ変換が必要かというと、まず、Shapeファイルは世界中のデータが入っているので、今回必要な日本のデータだけ取り出す必要があるのです。また、GeoJSONとTopoJSONと二種類のJSONファイルがありますが、GeoJSONは中間フォーマットであり、TopoJSONでは座標データをコンパクトに変換したものなので、容量を軽くするためにも形式を変換する必要があるのです。。

それでは、必要となるツールをインストールしましょう。

まずは、「gdal」というパッケージをインストールします。「gdal」とは、ラスターデータのメタデータの検索や、データフォーマットの変換を行うためのツールです。ここでも、変換を行うのに利用します。

CentOSであれば、以下のコマンドを実行してインストールします。

```
$ sudo yum install gdal
```

Mac OS Xであれば、Homebrewを利用して以下のコマンドを実行してインストールします。

```
$ brew install gdal
```

続いて、TopoJSON形式のデータを作成するために、npmを利用して、TopoJSONをインストールしましょう。

```
$ npm install -g topojson
```

これで、準備完了です。では、変換作業を始めましょう。

まず、Shapeファイルから、日本地図のデータだけを抽出しましょう。ダウンロードした地図データを解凍したディレクトリにて、以下のコマンドを実行しましょう。これで、日本の地図データだけを取り出して、GeoJSON形式に変換できます。

以下のコマンドですが、実際には、改行を入れずに実行してください。

```
$ ogr2ogr -f GeoJSON
  -where 'geonunit = "Japan"'
  japan-geo.json
  ne_10m_admin_1_states_provinces.shp
```

すると、「japan-geo.json」というファイルが生成されます。正しいデータが作成されているか確認してみましょう。JSONデータを見るのに便利なjqコマンドを使って調べてみます。(jqコマンドについては、本節の後のコラムをご覧ください。)

```
$ jq '.features[].properties.name_local' japan-geo.json | column
null     "愛媛県"   null     "千葉県"   "岩手県"
"広島県"   "香川県"   "富山県"   "茨城県"   "宮城県"
"岡山県"   "高知県"   "北海道"   "神奈川県"   "新潟県"
"島根県"   "大分県"   "福井県"   "埼玉県"   "山形県"
"鳥取県"   "徳島県"   "兵庫県"   "栃木県"   "長崎県"
"山口県"   "愛知県"   "京都府"   "東京都"   "鹿児島県"
"佐賀県"   "岐阜県"   "奈良県"   "山梨県"   "沖縄県"
"福岡県"   "石川県"   "大阪府"   "秋田県"   "群馬県"
"熊本県"   "三重県"   "滋賀県"   "青森県"
"宮崎県"   "長野県"   "和歌山県"   "福島県"
```

ぱっと見た感じでは問題なさそうなのですが、先頭のいくつかに問題があるようです。調べてみると「静岡県」のラベルが欠けているようなので、手動で修正します。テキストエディターで、「"name_local": null」という部分を調べて「"name_local": " 静岡県 "」と修正します（ただし、二カ所あるので、周辺のデータを見つつ、適切に修正しましょう）。

そして、このGeoJSON形式のファイルを、TopoJSON形式のデータに変換します。そのために、以下のコマンドを実行します。ちなみに「-p」オプションは、地名などの名前のプロパティを埋め込む場合に指定します。

```
$ topojson japan-geo.json -p > japan-topo.json
```

先ほど、GeoJSONをコンパクトに圧縮したのが、TopoJSONだと紹介しました。どのくらいコンパクトになったのか見てみましょう。

```
$ ll japan*
  715093  7 13 22:57 japan-geo.json
   92871  7 13 22:57 japan-topo.json
```

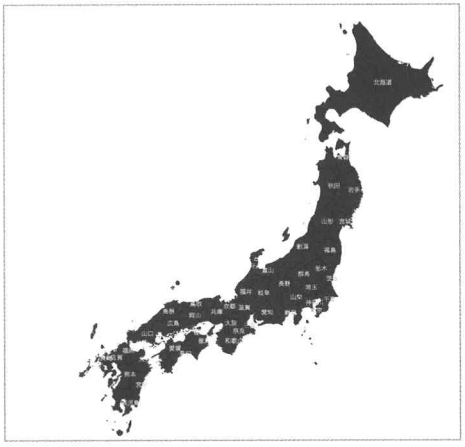
GeoJSONが698KBだったのに対して、90KBです。驚くほどコンパクトになっているのがわかります。

ついでに、D3.jsとTopoJSONのJavaScriptライブラリの圧縮版もここでダウンロードしておきましょう。

```
$ wget http://d3js.org/d3.v3.min.js
$ wget http://d3js.org/topojson.v0.min.js
```

D3.jsで日本地図を描画しよう

これで、準備が整いました。ここでは、以下のような日本地図を描画してみます。



日本地図を描画したところ

実際のプログラムは、以下のようになります。

● file: src/ch08/03-map/japan-map.html

前略

```
<script>
// SVG 領域を作成 ------ (※1)
var width = 1024, height = 1024;
var svg = d3.select("body")
  .append("svg")
  .attr({"width":width, "height":height});

// ファイルの読み込み ---- (※2)
d3.json("japan-topo.json", function(err, map) {
  // 描画オブジェクトを得る ------ (※3)
  var geo = map.objects["japan-geo"];
  var map_o = topojson.object(map, geo);

  // 縮尺を指定 ----- (※4)
  var projection = d3.geo.mercator()
        .center([137, 35])
        .scale(2000)
        .translate([width / 2, height / 2]);

  // パスを作成 ---- (※5)
  var path = d3.geo.path()
    .projection(projection);

  // SVG に追加 ---- (※6)
  svg.append("path")
     .datum(map_o)
     .attr("d", path);

  // 色を塗る ---- (※7)
  svg.selectAll("path").attr("fill", "green");

  // 境界線 ---- (※8)
  var mesh = topojson.mesh(
    map, geo,
    function(a, b) {
      return a !== b;
    });
  svg.append("path")
     .datum(mesh)
     .attr("d", path)
     .attr("class", "boundary");

  // 都道府県名を描画 ---- (※9)
  svg.selectAll(".place-label")
     .data(map_o.geometries)
     .enter()
       .append("text")
       .attr("class", function(d) {
          return "place-label";
       })
       .attr("transform", function(d) {
          return "translate(" + path.centroid(d) + ")";
       })
```

```
        .text(function(d) {
            var s = d.properties.name_local;
            if (!s) return;
            s = s.replace(/[都府県]$/,""); // ---- (※10)
            return s;
        });
});
</script>
</body>
</html>
```

プログラムを実行するには、ローカルサーバーを起動しておく必要があります。前節で作成した、chart-server.js をそのまま流用しましょう。以下のコマンドを実行してローカルサーバーを起動します。

```
$ node chart-server.js
```

続けて、Web ブラウザーで次の URL にアクセスします。すると地図データを表す JSON データが読み込まれ、日本地図が描画されます。

```
http://localhost:1337/japan-map.html
```

では、プログラムを確認してみましょう。プログラム (※1) の部分で、SVG の領域を設定します。ここでは、1024 × 1024 ピクセルのサイズで SVG を作成しています。

プログラム (※2) の部分では、地図データの JSON ファイルを読み込みます。プログラム (※3) の topojson.object() メソッドを使うことで、TopoJSON 形式のデータを、GeoJSON 形式に変換します。

プログラム (※4) の部分では、地図の縮尺および、中心の緯度経度を指定します。まず、d3.geo.mercator() メソッドを利用して、メルカトル投影法によるプロジェクションを返します。ちなみに、プロジェクトション (projection) とは射影、投影法を意味します。そして、プログラム (※5) の部分で、実際に描画するパスとなる座標データを作成します。あとは、SVG にパスを追加することで、実際に画面にパスを反映させることができます。このとき、描画した地図をどんな色で塗るのかを、(※7) の部分で指定しています。

プログラム (※8) の部分で、都道府県の境界を描画します。そのために、topojson.mesh() メソッドで、メッシュ情報を得た上で、SVG にパスを追加します。

最後に、都道府県の名前を示すラベルを追加しています (※9)。地図情報の geometries をデータに指定して、そこからプロパティ情報の「properties.name_local」を参照することで都道府県名をテキストとして追加しています。ただし、ラベル同士が重なってしまう部分も多いため、(※10) の部分で、都道府県名の末尾 (例えば、静岡県の「県」の部分) を削っています。

全体を眺めてみると、100 行以内のプログラムです。地図データがあるとは言え、100 行でカスタマイズ性の高い日本地図が描画できるのは便利です。

都道府県の詳細データ

なお、国土交通省国土政策局国土情報課も、「国土数値情報」をダウンロードできるように情報を公開しています。ここでも、Shape 形式のデータを入手できます。そのため、Natural Earth からダウンロードした Shape データと同様の方法で地図を表示させることができます。

```
国土交通省国土政策局国土情報課 > 国土数値情報
[URL] http://nlftp.mlit.go.jp/ksj/index.html
```

試しに、神奈川県の地図を表示させてみましょう。下記の行政区域データから、神奈川県を選んでダウンロードします。（ダウンロード時にデータを何に利用するのかアンケートに答える必要があります。）

```
国土数値情報　行政区域データ
[URL] http://nlftp.mlit.go.jp/ksj/gml/datalist/KsjTmplt-N03.html
```

ダウンロードしたら、ファイルを変換しましょう。神奈川県のデータは、「N03-150101_14_GML.zip」です。ZIP ファイルを解凍すると、Shape 形式のファイル「N03-15_14_150101.shp」がありますので、これを D3.js で利用できるように変換します。

まずは、Shape 形式を GeoJSON 形式に変換します。

```
$ ogr2ogr -f GeoJSON kanagawa-geo.json N03-15_14_150101.shp
```

次に、TopoJSON 形式に変換します。

```
$ topojson kanagawa-geo.json -p > kanagawa-topo.json
```

これを利用して、地図上に描画します。

● file: src/ch08/03-map/kanagawa-map.html

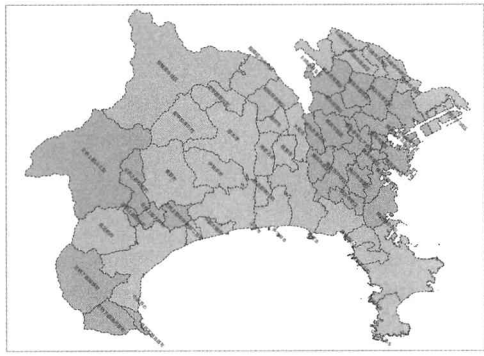
```
 前略

// カラーマップの指定 -------- (※1)
var colors = {
  "横浜市"   : "#dcd",
  "川崎市"   : "#ddc",
  "厚木市"   : "#ddd",
  "相模原市": "#cdd",
  "藤沢市"   : "#cdd",
  "足柄上郡": "#dcd",
  "足柄下郡": "#dcd",
  "others"   : "#ddc"
};

// ファイルの読み込み ---- (※2)
d3.json("kanagawa-topo.json", function(err, map) {
  // 描画オブジェクトを得る
  var geo = map.objects["kanagawa-geo"];
  var map_o = topojson.object(map, geo);

  // 縮尺を指定
  var projection = d3.geo.mercator()
        .center([cx, cy])
```

```
        .scale(scale)
        .translate([width / 2, height / 2]);

  // パスを作成
  var path = d3.geo.path()
    .projection(projection);

  // SVG に追加
  svg.append("path")
     .datum(map_o)
     .attr("d", path);

  // 色を塗る ------- (※3)
  svg.selectAll(".area")
     .data(map_o.geometries)
     .enter()
      .append("path")
      .attr("class", "area")
      .attr("fill", function(d) { // ---- (※4)
        var s = d.properties.N03_003;
        if (!s) { s = d.properties.N03_004; }
        if (colors[s] == undefined) {
          s = "others";
        }
        return colors[s];
      })
      .attr("d", path);

  // ラベルを作成 ---- (※5)
  var labels = {};
  svg.selectAll(".place-label")
     .data(map_o.geometries)
     .enter()
       .append("g")
       .attr("transform", function(d) {
          return "translate(" + path.centroid(d) + ")";
       })
       .append("text")
       .attr("text-anchor", "middle")
       .attr("class", "place-label")
       .attr("transform", "rotate(45,0,0)")
       .text(function(d) {
          var p = d.properties;
          var s = "";
          if (p.N03_003 !== null) s += p.N03_003;
          if (p.N03_004 !== null) s += p.N03_004;
          if (s == "") return;
          if (labels[s] === true) return;
          labels[s] = true;
          return s;
       });
});
</script>
</body>
</html>
```

それでは、実際に地図を表示させてみましょう。まずは、先ほどと同じようにWebサーバーを起動します。

```
$ node chart-server.js
```

続いて、Web ブラウザーで「http://localhost/kanagawa-map.html」にアクセスしましょう。市区名ラベルが重なってしまうところがあったので、45 度ラベルを傾けて重なるのを避けています。



神奈川県の地図

プログラムを見てみましょう。この地図では、市ごとに色分けを変えるようにしてみましたので、プログラムの (※ 1) で、市名と色の対応を指定しています。

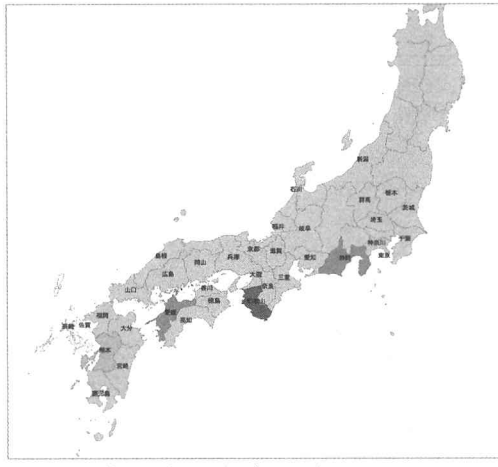
プログラムの (※ 2) では、JSON ファイルを読み込みます。これ以降の主な部分は、先ほどの描画プログラムと同じです。

プログラムの (※ 3) では、市区ごとに色を変えて塗ります。主に色を塗り分けているのは、(※ 4) の部分です。各データの properties.N03_003 に市名が、N03_004 に区名が入っています。そのため、これを利用して、塗り色 (fill) の値を切り替えています。色の指定は (※ 1) で行っています。

プログラムの (※ 5) では、市区名のラベルを作成しています。先ほど、各データの properties.N03_003 と N03_004 を見て区域を塗り分けましたが、ここでも、この値を利用してテキストを設定しています。

地図に出荷量データを重ねてみよう

次に、地図データに対して、ミカンの出荷量を示す以下のような地図を描画してみましょう。これは、日本地図の上に出荷量の多い地域に赤く色を付けるというものです。



ミカンの出荷量に応じて色づけしたところ

農林水産省のWebサイトを訪れると、ミカンなど果物の出荷量に関する情報を取得することができます。

農林水産省 > 作物統計 > 作況調査（果樹）
[URL] http://www.maff.go.jp/j/tokei/kouhyou/sakumotu/sakkyou_kazyu/index.html

上記のサイトより、ミカンの出荷量に関する情報をダウンロードします。そして、次のようなCSVファイルに変換しました。筆者は、Excelデータをダウンロードしたので、これをExcelで整形して、CSV形式で保存しました。

```
area,value
和歌山県,"157700"
愛媛県,"117300"
静岡県,"110200"
熊本県,"87100"
長崎県,"57400"
佐賀県,"47600"
愛知県,"25700"
...
```

「mikan.csv」から抜粋

続いて、これを日本地図上に色づけするのが次のプログラムです。

● file: /src/ch08/03-map/mikan-map.html

```
<!DOCTYPE html>

前略

// CSVファイルの読み込み --- (※1)
d3.csv("mikan.csv", function(err, data) {
  mikan = {};
  for (var i in data) {
    var area = data[i].area;
    var value = parseInt(data[i].value);
    if (value > max_value) max_value = value;
    mikan[area] = value;
  }
  drawData();
});
// 地図データの読込 ---- (※2)
d3.json("japan-topo.json", function(err, map) {
  // 描画オブジェクトを得る
  geo = map.objects["japan-geo"];
  map_o = topojson.object(map, geo);
  drawData();
});

// 画面描画 ---- (※3)
function drawData() {
  // 読み込み待ちする
  if (mikan == null || map_o == null) return;

  // 色の自動計算 --- (※4)
  var color_scale = d3.scale.quantize()
    .domain([0, max_value])
    .range(color_list);
```

```
// 縮尺を指定
var projection = d3.geo.mercator()
      .center([cx, cy])
      .scale(scale)
      .translate([width / 2, height / 2]);

// パスを作成
var path = d3.geo.path()
  .projection(projection);

// SVG に追加
svg.append("path")
   .datum(map_o)
   .attr("d", path);

// 色を塗る ------- (※5)
svg.selectAll(".area")
   .data(map_o.geometries)
   .enter()
    .append("path")
    .attr("class", "area")
    .attr("fill", function(d) {
      var area = d.properties.name_local;
      if (area == null) return;
      var value = mikan[area];
      if (value == undefined) value = 0;
      var c = color_scale(value);
      console.log(area, value, c);
      return c;
    })
    .attr("d", path);

後略
```

HTML をブラウザーで正しく見るために、ローカル Web サーバーを起動します。

```
$ node chart-server.js
```

起動後に Web ブラウザーで「http://localhost:1337/mikan-map.html」にアクセスします。

では、プログラムを確認してみましょう。プログラムの (※1) では、ミカンの CSV ファイルを読み込みます。読み込んだデータを変数 mikan にオブジェクト形式でセットしていきます。

続いて、プログラムの (※2) では地図データの読込を行います。そして、読み込みが完了したら、(※3) の部分、画面描画を行う drawData() 関数を呼び出します。

ここで、注目したいのが、drawData() が (※1) と (※2) の二カ所で呼ばれている点です。d3.json() も、d3.csv() も、共に Ajax でデータの読込を行います。それで、両方のデータが共に読み込まれてはじめて、画面の描画を行うようにしています。

プログラムの (※4) では色分けが自動的に行われるようデータの最大値とパレットの数を設定しています。

そして、今回最大の見せ場となるのが、プログラム (※5) の部分です。ここでは、都道府県ごとにミカンの出荷量に合わせて、色分けを行っています。塗り色の指定 (fill) をどの色にするのか、ミカンの出荷量に応じて決定します。

ところで、こうしてミカンの出荷量グラフを地図上に塗り分けてみると、綺麗に一直線に並んでいることがわかります。これだけでも面白いですね。

この節のまとめ

- 世界中の地図データが無料で入手できるので、手軽に世界地図を描画することができます。
- D3.js を利用するなら最小限の手間で、データを地図上に描画できます。

04

D3.jsから派生したライブラリ

ここまでの部分で、D3.js を利用してデータの視覚化を行いました。D3.js の柔軟性がよく理解できたことと思います。とは言え、もう少し手軽に使いたいと思った方もいるかもしれません。D3.js を利用して手軽にグラフを描画するライブラリを紹介します。

ここで学ぶポイント

- D3.jsを利用したライブラリ

ツールやライブラリの一覧

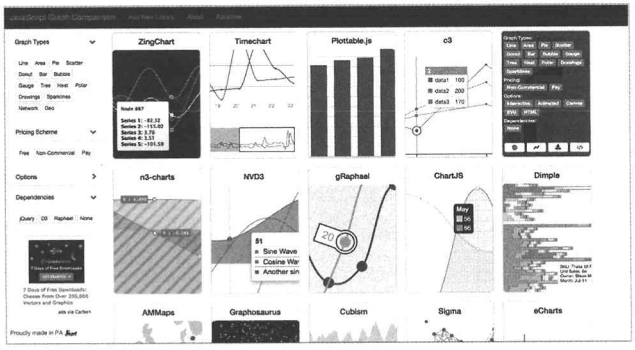
- NVD3.js
- C3.js

D3.js をベースに開発されたライブラリ

前節までの部分で、D3.js を使った基本的なグラフを紹介しました。実際に使ってみて、D3.js の柔軟性や拡張性について、刺激を受けたのではないでしょうか。とは言え、やはり、もう少し手軽にグラフを描画したいという要望は多くあるようです。というのも、D3.js を基礎ライブラリとして、多くの視覚化ライブラリが存在するからです。

JavaScript を対象にして、いろいろな視覚化ライブラリをまとめて紹介する「JavaScript Graph Comparison」という Web サイトがあります。そこでは、50 種類を超えるライブラリを紹介しています。

```
JavaScript Graph Comparison
[URL] http://www.jsgraphs.com/
```



JavaScript の視覚化ライブラリを集めた Web サイト

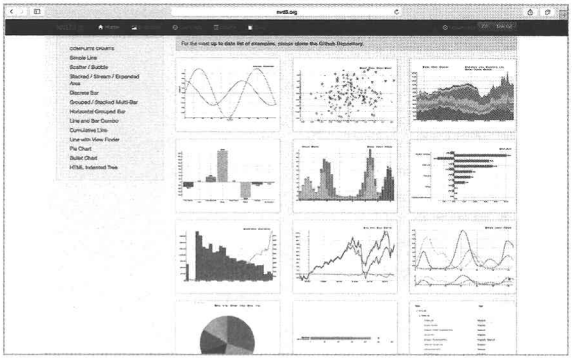
このサイトでは、D3.js を利用したライブラリだけを列挙することができるのですが、原稿執筆時点で 19 個ものライブラリが、D3.js に依存していることがわかります。

その中から代表的な二つのライブラリ「NVD3.js」と「C3.js」を紹介します。

NVD3.js について

まずは「NVD3.js」を紹介します。NVD3.js を使うと、さまざまな種類のグラフを簡単なコードで作成することができます。作成できるサンプルの一覧を見るとわかりますが、とても華やかで凝ったグラフが作成できます。

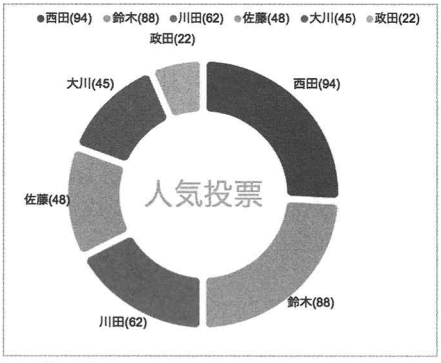
```
NVD3.js
[URL] http://nvd3.org/
```



NVD3.js のサンプル一覧

NVD3.js でドーナツチャート

どのくらい簡単かという例として、円グラフ（パイチャート）を少し変形させたドーナツチャートを描画してみようと思います。便利なことに、画面上部のラベルにある丸の部分をクリックすると、そのデータを除いたデータのグラフを描画します。



ドーナツチャート

このグラフを描画するHTMLが以下になります。

● file: src/ch08/04-ext/nvd3-pie.html

```
前略

<script>
// サンプルデータ
var sample_data = [
  { "label":"西田", "value":94 },
  { "label":"鈴木", "value":88 },
  { "label":"川田", "value":62 },
  { "label":"佐藤", "value":48 },
  { "label":"大川", "value":45 },
  { "label":"政田", "value":22 }
];

// グラフの描画 --- (※1)
nv.addGraph( function() {

  // パイチャートを作る ---- (※2)
  var chart = nv.models.pieChart()
    .x(function(d) {
      return d.label + "(" + d.value + ")";
    })
    .y(function(d) { return d.value })
    .title("人気投票")          // タイトルを指定
    .showLabels(true)          // ラベルを表示するか
    .donutLabelsOutside(true) // ラベルを外側に表示するか
    .padAngle(0.04)            // 各項目のパディング角度
    .cornerRadius(8)           // 角丸の指定
    .donut(true);              // ドーナツチャートにするか

  // パイチャートを実際に表示する ---- (※3)
  d3.select("#chart svg")
    .style({ "width": "800px", "height": "600px" })
    .datum(sample_data)
    .transition().duration(1000)
    .call(chart);

  // タイトルサイズの補正
  d3.select("#chart .nv-pie-title")
    .style({ "font-size": "50px" });

  return chart;
} );
</script>
</body></html>
```

HTMLファイルをWebブラウザーにドラッグ＆ドロップすることでグラフを表示することができます。

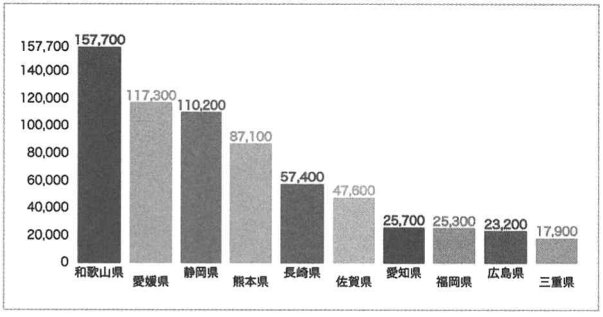
プログラムを少しずつ見てみましょう。まず、NVD3.jsを利用するには、次のように、CSSとJavaScriptを取り込む必要があります。これらのファイルは、NVD3.jsのサイトからダウンロードすることができます。ちなみに、NVD3.jsがD3.jsに依存しているのが「nv.d3.js」というファイル名にも表れています。

```
<!-- NVD3.js の CSS -->
<link href="lib/nvd3/nv.d3.css" rel="stylesheet" type="text/css">
<!-- D3.js を取り込む -->
<script src="lib/d3.min.js"></script>
<!-- NVD3.js を取り込む -->
<script src="lib/nvd3/nv.d3.js"></script>
```

NVD3.js では、プログラム (※ 1) の nv.addGraph() メソッドの引数にグラフ描画の処理を指定します。プログラムの (※ 2) では、パイチャートを作成します。ドーナツチャートはパイチャートの一種となっており、donut() メソッドに true を指定することでドーナツチャートにすることができます。その他にも、ラベルを表示するかどうかの showLabels() や、ラベルの表示位置を指定する donutLabelsOutside() メソッドなどを指定することで、グラフの見た目を大きく変えることができます。プログラムの (※ 3) の部分で、パイチャートにデータを指定した上で、画面に描画します。

NVD3.js でバーチャート

次に、NVD3.js で棒グラフ（バーチャート）を作成してみます。ただチャートを作成するのではなく、CSV ファイルを読み込んでグラフを表示しています。ここでは、前節で、農林水産省のサイトからダウンロードして加工したミカンの出荷量データのトップ 10 を棒グラフで、以下のように表示してみます。



CSV ファイルから棒グラフを描画した

ここで読み込む CSV ファイルですが、以下の通りです。

● file: src/ch08/04-ext/mikan-sub.csv

```
area,value
和歌山県 ,"157700"
愛媛県 ,"117300"
静岡県 ,"110200"
熊本県 ,"87100"
長崎県 ,"57400"
佐賀県 ,"47600"
愛知県 ,"25700"
福岡県 ,"25300"
広島県 ,"23200"
三重県 ,"17900"
```

この CSV ファイルを読み込んでグラフを描画するプログラムは次のようになります。

● file: src/ch08/04-ext/nvd3-bar.html

```
<!DOCTYPE html>
<html><head><meta charset="utf-8">
  <link href="lib/nvd3/nv.d3.css" rel="stylesheet" type="text/css">
  <script src="lib/d3.min.js"></script>
  <script src="lib/nvd3/nv.d3.js"></script>
</head><body>
<div id="chart"><svg></svg></div>
<script>
// CSV を読んでグラフを描画する
var mikan = null;

// CSV の読み込み --- (※1)
d3.csv("mikan-sub.csv", function (err, data) {
  for (var i in data) {
    data[i].value = parseInt(data[i].value);
  }
  mikan = [{"key":" 出荷量 ", "values":data}];
  // 続いて、グラフの描画
  nv.addGraph( drawGraph );
});

function drawGraph() {
  // バーチャートを作る ---- (※2)
  var chart = nv.models.discreteBarChart()
    .x(function(d) { return d.area })
    .y(function(d) { return d.value })
    .staggerLabels(true)
    .margin({"left":100,"right":20,"top":50,"bottom":50})
    .valueFormat(d3.format(',d'))
    .showValues(true);

  chart.yAxis.tickFormat(d3.format(',d'));

  // 実際に表示する ---- (※3)
  d3.select("#chart svg")
    .style({ "width": "800px", "height": "400px" })
    .datum(mikan)
    .call(chart);

  nv.utils.windowResize(chart.update);
  return chart;
};

</script>
</body></html>
```

ここでは、D3.js の機能を利用して CSV ファイルを読み込んでいます。つまり、Ajax を利用しますので、ローカルに配置した HTML ではセキュリティのエラーが出て読み込むことができません。そこで、前節と同じように、chart-server.js でローカル Web サーバーを起動します (chart-server.js は、本章の初めに作成したものをそのまま使います)。

```
$ node chart-server.js
```

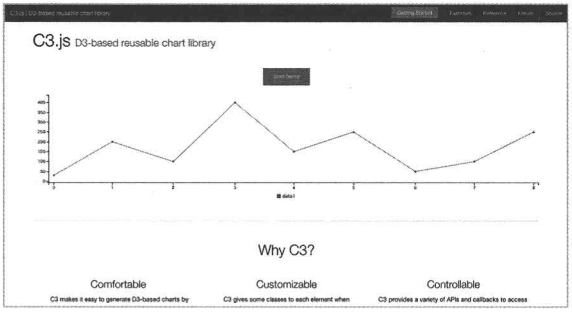
あとは、Web ブラウザーから「http://localhost:1337/nvd3-bar.html」へアクセスすると棒グラフが描画されます。

プログラムを見てみましょう。プログラムの (※ 1) の部分ですが、先ほど紹介したように、D3.js の機能を利用して、CSV ファイルを読み込んでいます。このように、手軽にデータの読込ができるのも、D3.js を基礎ライブラリとして使っていることのメリットであると言えるでしょう。

ついでグラフを作成します。プログラム (※ 2) では、discreteBarChart を作成します。その後の部分で、ラベルを指定したり、グラフのマージンを指定したり、値の表示の仕方を指定したりします。デフォルトでは、小数点以下「.00」まで表示されますが、ここでは、ミカンの出荷量は整数なので、d3.format(',d') と指定します。そして、プログラムの (※ 3) では、実際にグラフを画面に表示します。

C3.js を使う

次に、C3.js を紹介します。C3.js も D3.js を基礎データを利用したグラフライブラリです。C3.js の特徴は、動的な更新に強いこと、また、簡単にグラフの種類を差し替えることができること、さまざまなインタラクションが可能であることです。この辺りは、D3.js の柔軟性を上手に活かした作りとなっています。



C3.js の Web サイト

```
C3.js
[URL] http://c3js.org/
```

C3.js は、以下の URL からダウンロードできます。

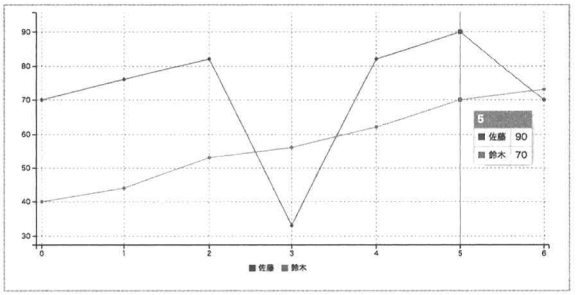
```
C3.js の最新版ダウンロード
https://github.com/masayuki0812/c3/releases/latest
```

本書執筆時点の最新版 0.4.10 を利用しました。ファイルをダウンロードすると、いくつかのファイルがありますが、ここでは、「c3.min.css」と「c3.min.js」の二つのファイルをコピーしておきます。

C3.js を利用するには、以下のようなコードを記述します。

```
<!-- CSS の取り込み -->
<link href="lib/c3/c3.min.css" rel="stylesheet" type="text/css">
<!-- D3.js の取り込み -->
<script src="lib/d3.min.js"></script>
<!-- C3.js の取り込み -->
<script src="lib/c3/c3.min.js"></script>
```

簡単な折れ線グラフ（ラインチャート）を描画してみましょう。



C3 で描画した例

C3.js が便利な点は、グラフを描画するときに、配列変数でデータを与えるだけで、グラフが描画されるという点です。まずは、ソースコードを見てみましょう。

● file: src/ch08/04-ext/c3-test.html

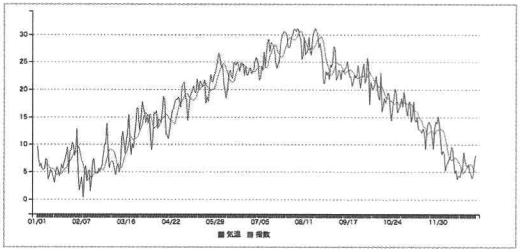
```
<!DOCTYPE html>
<html><head><meta charset="utf-8">
  <link href="lib/c3/c3.min.css" rel="stylesheet" type="text/css">
  <script src="lib/d3.min.js"></script>
  <script src="lib/c3/c3.min.js"></script>
</head><body>
<div id="chart" style="width:800px; height:400px;"></div>
<script>
// サンプルデータ
var sample_data = [
  ['佐藤', 70, 76, 82, 33, 82, 90, 70],
  ['鈴木', 40, 44, 53, 56, 62, 70, 73]
];

// グラフの描画 ---- (※1)
var chart = c3.generate({
  bindto: '#chart',
  data: {
    columns: sample_data
  },
  grid: {
    x: { show: true },
    y: { show: true }
  }
});

</script>
</body></html>
```

プログラムの (※1) の部分に注目してみましょう。c3.generate() というメソッドを一つ呼び出すだけでグラフの描画が完成してしまいます。これまで見てきた、どのグラフ描画ライブラリよりもシンプルです。

それでは、c3.generate() メソッドに与える引数のオブジェクトに注目してみましょう。bindto では、HTML 内のどの要素にグラフを描画するのかを指定します。そして、data.columns に実際のデータを与えます。grid.x.show と grid.y.show に true を与えると、グラフの背面のグリッドに線を描画するようになります。

C3.js で気温を表示する

次に、C3.js に 2014 年の一年分の平均気温データを読み込ませてみます。ここでは、7 章で作成した 2014 年の平均気温と SMA の値をグラフに描画してみようと思います。描画すると以下のようになります。



気温と SMA の値をグラフにしたところ

では、さっそく HTML ファイルを見てみましょう。

● file: src/ch08/04-ext/c3-kion.html

```
<!DOCTYPE html>
<!DOCTYPE html>
<html><head><meta charset="utf-8">
  <!-- ライブラリの取り込み -->
  <link href="lib/c3/c3.min.css" rel="stylesheet" type="text/css">
  <script src="lib/d3.min.js"></script>
  <script src="lib/c3/c3.min.js"></script>
  <!-- 気温データ -->
  <script src="2014kion-sma.js"></script>
</head><body>
<div id="chart" style="width:800px; height:400px;"></div>
<script>
// 気温データを変換 ----- (※1)
var tempdata_c3 = [],
    labels = ["x"], temp = ["気温"], sma = ["指数"];
for (var i in kion_data) {
  var x = kion_data[i];
  labels.push(new Date("2014/"+x[0]));
  temp.push(x[1]);
  sma.push(x[2]);
}
tempdata_c3 = [labels, temp, sma];

// グラフの描画 ---------- (※2)
var chart = c3.generate({
  bindto: '#chart',
  data: {
    x: 'x',
    columns: tempdata_c3
  },
  axis: { // ------------(※3)
    x: {
      type: 'timeseries',
      tick: { format: '%m/%d' }
    }
  },
```

```
  grid: {
    x: { show: false },
    y: { show: true }
  },
  point: { show: false }
});
</script>
</body></html>
```

プログラムですが、(※1)の部分で、気温データを変換しています。というのも、ここで取り込んだ気温データは、7章でGoogle Chartsのために作成したデータですので、次のようなデータとなっているのです。

```
var kion_data = [
// 日付,  気温, SMA
  ["1/1", 9.6, null],
  ["1/2", 7.3, 6.5],
  ["1/3", 5.9, 6.1],
  ...
]
```

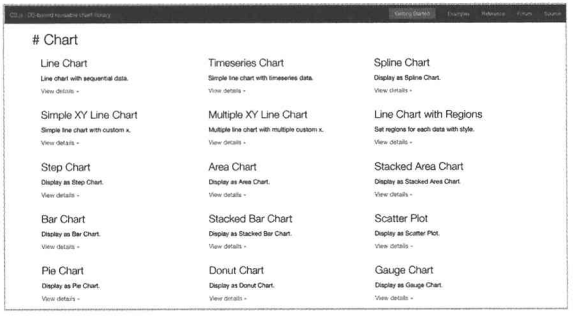
しかし、C3.jsでグラフにするためには、これを回転した形、つまり次のようなものにしなくてはなりません。

```
var tempdata_c3 = [
  ["x",     "1/1", "1/2", "1/3" ... ],
  ["気温",  9.6,   7.3,   5.9   ... ],
  ["SMA",   null,  6.5,   6.1   ... ]
]
```

この操作さえ行えば、C3.jsのための大半の作業は完了したと言えるでしょう。

プログラムの(※2)を見てみましょう。先ほどと同じで、グラフを描画するのに必要なのは、c3.generate()メソッドを一つ呼ぶだけです。とは言え、その分、オプションでどのような描画を行うかを指定しなければなりません。まず、データの一行目のデータが日付であることをC3.jsに教えなくてはなりません。そのため、プログラムの(※3)のaxisプロパティのtypeを'timeseries'とし、tickに表示フォーマットを指定します。

このように、C3.jsを使うと、非常に手軽にグラフを描画することができます。ちなみに、C3.jsのドキュメントのexampleには、さまざまな項目をどのように指定したら良いのか、豊富なグラフの表示例が用意されています。これを参考にしてパラメーターを設定すれば、思い通りのグラフを描画することができるでしょう。



C3.jsのマニュアル

C3.js のマニュアル
http://c3js.org/examples.html

この節のまとめ

- この節では、D3.js を基礎ライブラリとして利用した二つのライブラリ「NVD3.js」と「C3.js」を紹介しました。
- これらのライブラリを使うなら、素早くデータの視覚化を行うことができます。
- 基礎ライブラリの D3.js の機能がそのまま利用できるので、データの読み込みなどを行うのも非常に手軽であることがわかります。状況に応じて、これらのライブラリを使うことを選ぶと良いでしょう。

Appendix

APPENDIX 01

WindowsやOS X上に開発環境を構築しよう

CentOSの仮想マシンを利用した設定を紹介しましたが、やはり、WindowsやMac OS Xに直接インストールしたいという場合もあります。そこで、簡単に、WindowsやMac OS XにJavaScriptのエンジンをインストールする方法を紹介します。

「Node.js」のインストール

Node.jsのWebサイトを訪れると、各OS用のインストーラーが提供されています。インストーラーを利用して、Node.jsをセットアップすることができます。

INSTALLボタンを押すとインストーラーをダウンロードできます

```
* node.jsのWebサイト
http://nodejs.org/
```

Windowsの場合 *

上記Webサイトより、インストーラーをダウンロードします。インストーラーをダブルクリックしたら、後は、「次へ」ボタンを数回クリックすればインストールが完了します。

---

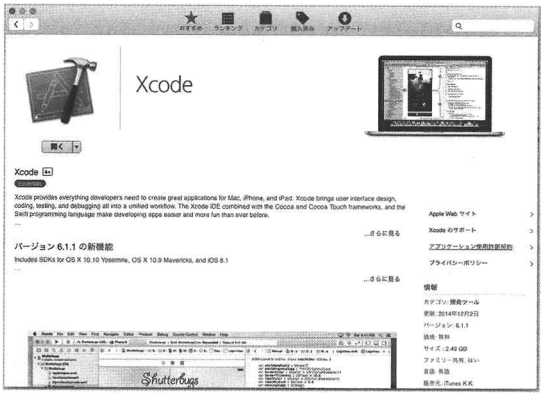
* 補足ですが、WindowsでNode.jsのモジュールをインストールするとき、ソースコードからバイナリをコンパイルする必要がある場合、PythonやWindowsのSDKが必要になることがあります。
例えば、SQLite3のモジュールをインストールする場合、以下のリンクを参考にしてみてください。

● Windows上でNode.jsからSQLite3を使う方法
http://qiita.com/kujirahand/items/f262b05123c25353e93d

Mac OS X の場合

Windows の場合と同様に、インストーラーを使う方法もあります。また、パッケージマネージャーの Homebrew を使ってインストールする方法もあります。本書では、さまざまなコマンドラインツールを利用したプログラムを紹介します。Mac OS X 環境に各種開発ツールを用意します。そこで、Mac OS X に Xcode、および、コマンドラインツールをインストールしておきましょう。

まず、Mac の App Store から Xcode を探してインストールします。



xcode をインストールする

続いて、「ターミナル .app」を起動し、以下のコマンドを実行します。これにより、Command Line Tools がインストールされます。

```
xcode-select --install
```

Homebrew をインストールします。このために、ターミナルから以下のコマンドを実行します。このコマンドは、Homebrew の Web サイトからコピー＆ペーストできます。

```
$ ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/
master/install)"
```

Homebrew（日本語）の Web サイト
http://brew.sh/index_ja.html

Homebrew がインストールされたら、ターミナルから以下のコマンドを実行します。以下のコマンドを実行すると「node.js」がインストールされます。

```
$ brew install node
```

「Rhino」のインストール

Rhinoの実行エンジンは、Java SE6から標準でJavaに搭載されていますが、Mozillaのページから、Rhinoの最新版を入手することができます。より柔軟に構文が記述できるようになっています。本書執筆時点での最新版は、Rhino 1.7R5(2015-01-29)です。下記のダウンロードサイトから、jarファイルをダウンロードします。Rhinoの本体は、アーカイブに含まれるJARファイルの「js.jar」です。

```
Rhino download archive
https://developer.mozilla.org/en-US/docs/Mozilla/Projects/Rhino/Download_
Rhino
```

また、Rhinoを動かすためには、Javaのランタイムが必要です。インストールされていなければ、Javaのサイトからダウンロードしてインストールしてください。

```
Javaのサイト
http://java.com/ja/
```

Windowsの場合

Windowsでは、Rhinoが手軽に実行できる以下のようなバッチファイルを用意すると良いでしょう。本書では、このバッチファイルがパスの通ったところにコピーしてあることを前提にしています。

● file: src/App/rhino.bat

```
java ^
  -cp "c:/bin/*" ^
  org.mozilla.javascript.tools.shell.Main %*
```

ここでは、上記からダウンロードしたRhinoの本体「js.jar」と上記の「rhino.bat」をCドライブ直下の「c:\bin」にコピーします。

バッチファイルとJARファイルをコピー

そして、パスを通す方法ですが、Windows Vista以降では、次のようにします。まず、コマンドプロンプトを管理者権限で起動します。そのために、スタートメニューあるいは、スタート画面のコマンドプロンプトのアイコンを右クリックし「管理者で実行」を実行します。ユーザーアカウントの確認ダイアログが出るので「はい」で実行を許可します。

管理者でコマンドプロンプトを実行

続いて、以下のコマンドを実行します。これで、先ほどの bin フォルダにパスが通ります。

```
> SETX /M PATH "%PATH%;c:\bin"
```

以上でインストールができました。うまくインストールできたかどうかテストしてみましょう。以下のプログラムを実行すると、Rhino コマンドが実行された後、11 が表示されるでしょう。

```
> rhino -e "print(3 + 8)"
11
```

Mac OS X の場合

Mac OS X では、Homebrew を利用して、Rhino コマンドがインストールできます。

```
$ brew install rhino
```

「Nashorn」のインストール

Nashorn は、Java8 以降の JRE(Java 実行エンジン) あるいは、JDK(Java 開発キット) に含まれています。Java をインストールするには、Java のサイトから最新の Java をインストールできます。Nashorn は Java に含まれています。

```
Java の Web サイト
https://java.com/ja/
```

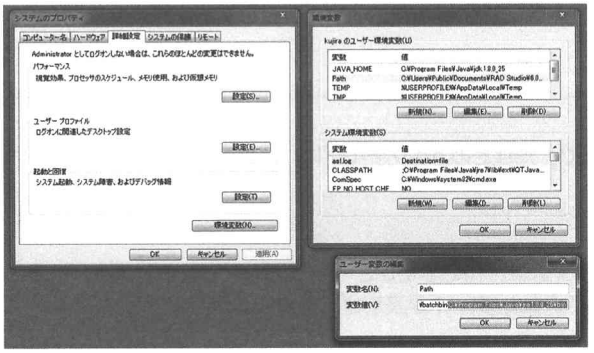
JRE/JDK 共に、インストールされたパスの「bin」ディレクトリ以下に、jjs という名前の実行ファイルがあります。これが、Nashorn のメイン実行ファイルです。

Windows の場合

Windows では、Program Files フォルダ以下にある「Java/jre***/bin」フォルダにある「jjs.exe」が Nashorn の本体となります。jjs を利用するために、このフォルダにパスを通しておく必要があります。

パスを通すには、上記、Rhino と同じ方法で、管理者権限でコマンドプロンプトを実行しパスを SETX コマンドでパスを追加します。あるいは、コントロールパネルより「システムとセキュリティ＞システム＞システムの詳細設定＞環境変数」より環境変数を設定できます。

Path の項目を探して、[編集] ボタンを押します。そして、そこに、bin フォルダのパスを追加します。



環境変数の Path に bin フォルダを追加します

Mac OS X の場合

Mac OS X では、Java のホームディレクトリ以下、「bin/jjs」が Nashorn の実行ファイルとなっています。Java のホームディレクトリを調べるには、ターミナルから以下のようなコマンドを実行すると調べることができます。

```
$ /usr/libexec/java_home
```

上記のコマンドを実行すると、筆者の手元の OS X では、Library/Java ディレクトリ以下のようなパスが表示されました。この実行結果は、環境によって異なるでしょう。

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_25.jdk/Contents/Home
```

そこで、jjs コマンドが手軽に実行できるように、以下のようにしてシンボリックリンクを張っておくと、手軽に Nashorn(jjs コマンド) を実行することができるようになります。

```
$ ln -s `/usr/libexec/java_home`/bin/jjs /usr/local/bin/jjs
```

APPENDIX 02

HTML/XMLのパースを簡単に行う

Web サイトから Node.js を利用してダウンロードしたデータから , 任意の値を素早く取り出すのに役立つライブラリがあります。それが「cheerio」モジュールです。2章でも紹介しているのですが、ここではもう少し詳しく使い方を紹介します。

「cheerio」と「cheerio-httpcli」について

「cheerio」モジュールを使うと、HTML/XML のデータに対して、jQuery ライクに任意のオブジェクトを操作することができます。

これを利用したモジュールが「cheerio-httpcli」です。こちらは、cheerio に Web ページの取得機能を付け加えたものです。ただページを取得するだけでなく、面倒な文字コードの変換まで自動でやってくれます。

そのため、Web ページを取得するところから記述する場合には「cheerio-httpcli」モジュールを利用し、別途、Web API などで取得したデータから任意のデータを取り出したい場合などは、「cheerio」モジュールを使います。

「cheerio-httpcli」モジュールについては、すでに2章で紹介したので、ここでは、「cheerio」モジュールの使い方を詳しく見ていきましょう。

基本的な使い方

cheerio モジュールのインストールは、以下のようにコマンドラインから npm コマンドを実行します。

```
$ npm install cheerio
```

XML データから、書籍のタイトル一覧を表示してみましょう。

● file: src/App/01-cheerio/xml-title-list.js

```
// モジュールの取り込み
var cheerio = require('cheerio');

// サンプル XML データ
var xml = "<books>" +
  "<book><title> リンゴの山 </title><author> 山田 </author></book>" +
  "<book><title> ミカンの歌 </title><author> 佐藤 </author></book>" +
  "<book><title> バナナの里 </title><author> 市川 </author></book>" +
  "</books>";

// cheerio に XML データを読み込ませる ---- (※1)
$ = cheerio.load(xml);

// 本のタイトル一覧を表示 --- (※2)
$("title").each(function(i, e) {
  var title = $(e).text();
  console.log(title);
});
```

プログラムを実行するには、コマンドラインから以下のようなコマンドを入力します。

```
$ node xml-title-list.js
リンゴの山
ミカンの歌
バナナの里
```

プログラムの (※ 1) にあるように、load() メソッドで XML を読み込ませます。

そして、プログラムの (※ 2) にあるように、$("title") で <title> タグの一覧を取得します。each() メソッドは取得した要素のそれぞれに対して処理を行う場合に利用します。そのため、<title> タグを列挙し、それぞれの要素について、テキスト部分を取り出してコンソールに出力するという働きをします。

また、列挙した要素の、text() メソッドを呼び出すことで、テキストを取得します。このとき、間違え安いのが、each() メソッドの引数で得られたオブジェクト変数 e 自体には、text() メソッドがないということです。そのため、$() メソッドを利用して、$(e).text() と記述する必要があります。

```
$("title").each(function(i, e) {
  console.log(e.text());     // ← エラーとなる
  console.log($(e).text()); // ← OK
});
```

取得したオブジェクト以下をさらに検索する

ここから少しいろいろなプログラムを書いて動作を検証してみましょう。プログラムを検証するのに都合の良い、サンプルの XML ファイルを利用します。次のような XML ファイルで「test.xml」という名前で保存します。

● file: src/App/01-cheerio/test.xml

```
<books>
  <book id="b1000">
    <title>リンゴの山</title>
    <author>山田</author>
    <price value="1500"/>
    <options>
      <color>red</color>
    </options>
  </book>
  <book id="b1001">
    <title>ミカンの歌</title>
    <author>佐藤</author>
    <price value="1800"/>
    <options>
      <color>orange</color>
    </options>
  </book>
  <book id="b1002">
    <title>バナナの里</title>
    <author>市川</author>
    <price value="2400"/>
    <options>
```

```
      <color>yellow</color>
    </options>
  </book>
</books>
```

このXMLで、タイトル、筆者を取得するプログラムは、次のようになります。

● file: src/App/01-cheerio/show-books.js

```
// モジュールの取り込み
var cheerio = require('cheerio');
var fs = require('fs');

// サンプルXMLデータを得て、cheerioに読み込ませる
var xml = fs.readFileSync("test.xml", "utf-8");
$ = cheerio.load(xml);

// 本の情報を表示
$("book").each(function(i, e) {
  // <book>タグの子要素から値を取得 ---- (※1)
  var title = $(e).children('title').text();
  var author = $(e).children('author').text();
  console.log(title + " - " + author);
});
```

コマンドラインから、プログラムを実行してみます。

```
$ node show-books.js
リンゴの山 - 山田
ミカンの歌 - 佐藤
バナナの里 - 市川
```

ポイントとなるのは、プログラムの(※1)の部分です。ここでは、<book>要素の直下の子要素に対して、children()メソッドを使って取り出しています。

属性値を取り出す

次にタグの属性値を取り出す方法を紹介します。

● file: src/App/01-cheerio/show-book-price.js

```
// モジュールの取り込み
var cheerio = require('cheerio');
var fs = require('fs');

// サンプルXMLデータを得て、cheerioに読み込ませる
var xml = fs.readFileSync("test.xml", "utf-8");
$ = cheerio.load(xml);

// 本のIDと値段を表示
$("book").each(function(i, e) {
  // ID属性を取得 --- (※1)
  var id = $(e).attr("id");
  // <price>タグのvalue属性を取得 --- (※2)
  var price = $(e).children("price").attr("value");
  console.log(id + ":" + price);
});
```

以下のコマンドを実行してみましょう。

```
$ node show-book-price.js
b1000:1500
b1001:1800
b1002:2400
```

cheerio でタグの属性値を取得するには、attr() メソッドを指定します。プログラム (※ 1) で、attr("id") と書くと id 属性を取得します。プログラムの (※ 2) は、子要素 <price> タグの value 属性を取得するというプログラムです。

子要素以下にあるタグを検索する

それでは、取り出した要素の子要素の子要素、つまり孫要素にあたるタグを抽出したいときにはどうしたら良いでしょうか。children() メソッドは子要素を取り出しますが、孫要素を検索対象にしたいときには、find() メソッドを利用します。

以下のプログラムでは、book > options > color という孫要素の値を取り出します。

● file: src/App/01-cheerio/show-book-color.js

```
// モジュールの取り込み
var cheerio = require('cheerio');
var fs = require('fs');

// サンプル XML データを得て、cheerio に読み込ませる
var xml = fs.readFileSync("test.xml", "utf-8");
$ = cheerio.load(xml);

// 本のタイトルと色を表示
$("book").each(function(i, e) {
  // タイトルを表示
  var title = $(e).children("title").text();
  // 色を表示 ---- (※1)
  var color = $(e).find("color").text();
  console.log(title + " - " + color);
});
```

プログラムを実行してみましょう。孫要素にあたる color タグの値も取り出すことができています。

```
$ node show-book-color.js
リンゴの山 - red
ミカンの歌 - orange
バナナの里 - yellow
```

プログラムの (※ 1) の部分に注目してみましょう。孫要素にあたる color タグを探すために使っているのは、find() メソッドです。

テーブル内の情報を取得する

次に、よくあるシチュエーションとして、HTML ファイル内のテーブルファイルからデータを取り出すというプログラムを作ってみます。ここでは、次のような HTML ファイルを対象にしてみます。

● file: src/App/01-cheerio/table.html

```
<!DOCTYPE html>
<html><body>
<table id="tbl">
  <tr><th> 商品名 </th><th> 値段 </th></tr>
  <tr><td> ニンジン </td><td>220</td></tr>
  <tr><td> ジャガイモ </td><td>140</td></tr>
  <tr><td> ダイコン </td><td>190</td></tr>
</table>
</body></html>
```

このファイルからテーブルを解析してデータを取り出してみましょう。

● file: src/App/01-cheerio/read-table.js

```
// モジュールの取り込み
var cheerio = require('cheerio');
var fs = require('fs');

// ファイルを cheerio に読み込ませる
var html= fs.readFileSync("table.html", "utf-8");
$ = cheerio.load(html);

// テーブルを取得
var data = readTable("#tbl");
console.log(data);

// テーブルのセルを全部読む
function readTable(query) {
  var data = [];
  var table = $(query);
  var tr_list = $(table).children("tr");
  for (var i = 0; i < tr_list.length; i++) {
    var cells = tr_list.eq(i).children();
    var cols = [];
    for (var j = 0; j < cells.length; j++) {
      var v = cells.eq(j).text();
      cols.push(v);
    }
    data.push(cols);
  }
  return data;
}
```

プログラムを実行してみましょう。コマンドラインで次のように入力します。

```
$ node read-table.js
[ [ '商品名', '値段' ],
  [ 'ニンジン', '220' ],
  [ 'ジャガイモ', '140' ],
  [ 'ダイコン', '190' ] ]
```

このプログラムですが、テーブルを読み取る機会は多いと思うので、関数にまとめてみました。readTable() 関数がテーブルの読み込みをしている部分です。セルの内容を読み取って二次元配列変数として返します。

children() メソッドの戻り値は、複数の要素を持っています。そのため、length メソッドがあり、いくつの要素が含まれているのかを確認できます。そのため、for 構文を使って、length 分だけ要素を取り出します。指定番目にある要素を取得するには、eq() メソッドを使います。

cheerio のまとめ

このように、cheerio を使うと、手軽に HTML/XML の要素を取り出すことができます。使いこなすことができれば、とても便利です。GitHub の cheerio のページでは、各メソッドのマニュアルがありますので、こちらも一通り目を通しておくと良いでしょう。

```
GitHub > cheerio
https://github.com/cheeriojs/cheerio
```

索引

［著者略歴］

クジラ飛行机（くじらひこうづくえ）

中学時代から趣味でやっていたプログラミングが楽しくていろいろ作っているうちに本職のプログラマーに。現在は、ソフト企画「くじらはんど」にて、Windows から Android アプリまで「楽しく役に立つツール」をテーマに作品を公開している。代表作は、ドレミで作曲できる音楽ソフト『テキスト音楽「サクラ」』や『日本語プログラミング言語「なでしこ」』など。2001 年にはオンラインソフトウェア大賞に入賞、2004 年度 IPA 未踏ユースでスーパークリエイターに認定、2010 年に OSS 貢献者賞を受賞。日本中にプログラミングの楽しさを伝えるため日々奮闘中。

カバー・本文デザイン：坂本真一郎（クオルデザイン）
編集協力：片野美都
DTP：G2UNIT inc.

JS+Node.jsによる Webクローラー / ネットエージェント 開発テクニック

2015 年 9 月 14 日　初版第 1 刷発行
2016 年 5 月 20 日　初版第 5 刷発行

著　者　クジラ飛行机
発行人　片柳 秀夫
編集人　佐藤 英一
発行所　ソシム株式会社
http://www.socym.co.jp/
〒 101-0064 東京都千代田区猿楽町 1-5-15
猿楽町 SS ビル 3F
TEL　03-5217-2400（代表）
FAX　03-5217-2420
印刷・製本 株式会社暁印刷

定価はカバーに表示してあります。
落丁・乱丁は弊社販売部までお送りください。送料弊社負担にてお取り替えいたします。
ISBN978-4-88337-993-4

ISBN978-4-88337-993-4
C2055 ¥3200E
定価：本体3200円(税別)
9784883379934
1922055032005

CONTENTS