

第 18 章

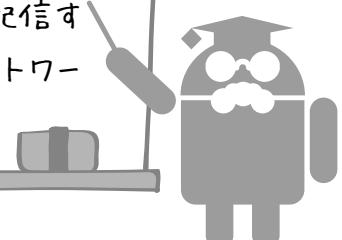
Web サーバー

著：宮川大輔

18-1 Webサーバーをつくる

著：宮川大輔

本節では、これまで"とは視点を変えて、Androidアプリにデータを提供するWebサーバー側の技術を学びます。Webサーバーとは何なのかについて考えた後、実際にデータを配信するサーバーを実装してみましょう。ここでは、第13章「ネットワークプログラミング」で学んだことを前提にしています。



この節で学ぶこと

- ・Webサーバーとは何か
- ・「Hello Worldサーバー」を実装する

この節で出てくるキーワード一覧

Webサーバー
クライアント
リソース
IaaS
PaaS
SaaS
Google App Engine (GAE)
Python
PyCharm Community Edition
webapp2
RequestHandler
HTTP
GETリクエスト
POSTリクエスト



18-1-1 アプリ内での代表的な映像 / 音声利用場面

現在は、Webブラウザを用いてWebページを閲覧するのが当たり前の時代ですから、Webという言葉についても、一般的なイメージがあるかもしれません。ごく簡単に言えば、蜘蛛の巣(英単語でWeb)のように、コンピューターを相互接続したネットワークが世界規模で張り巡らされていて、WebブラウザやAndroidアプリといった「クライアント」と「サーバー」が容易に接続できるのがWebの特徴です。

「Webサーバー」の「サーバー(Server)」という単語は、「ジュースサーバー」といった表現で日常会話でも登場します。IT業界以外でも「給仕する人、接客係」といった意味で使われる言葉です。

堅苦しく言えば、**サーバーは、サービスを要求する側に、そのサービスを提供します。**「サービスを要求する側」を「クライアント(Client)」と呼びます。クライアントという言葉自体は、例えば日本でも、コンサルティングを受けるお客様に対しても使われます。

特にWebサーバーの場合、そのサービスというのは何らかの資源、あるいは「リソース(Resource)」をクライアントに渡したり、クライアントの持つリソースを受け取って保存したりします。ジュース「サーバー」の場合、「クライアント」が要求する「リソース」はジュースでした。Web「サーバー」の場合、「クライアント」であるWebブラウザやAndroidアプリが要求する「リソース」はWebページや、その中の「お天気情報」、今回扱う「講義情報」などです。

なお、本章では説明を簡略化するため、「インターネット」と「クラウド」と「Web」の区別や、「Webサーバー」と「インターネット上の他のサーバー」の区別、「Webサーバー」と「Webアプリ」の区別については扱いません。みなさんがWebブラウザで見る向こう側を、全て「Webサーバー」「Webアプリ」という一括りで考えます。



18-1-2 自分のWebサーバーが必要な場面

Webサーバーとの関係で考えたとき、Androidアプリはおおまかに次の3種類に分けられます。

- ①そもそもネットワーク接続が不要ない
- ②誰かが用意したWebサーバーを利用する
- ③(本章のように)自分でWebサーバーを準備する

1つ目のタイプは今回、扱いません。本章では3つ目のタイプを実践することを通じて、Androidアプリ開発者が最も多く出会うであろう、2つ目のタイプについても役

立つ知見を得ることが目標です。

今回は、例えば「Google Play Game Services」(<https://developer.android.com/intl/ja/google/play-services/games.html>)の機能を使って、ゲームの「アチーブメント」情報をアップロードするといった場合も、2つ目に含めています。こういう場合には、しばしば17章「外部ライブラリーの利用と作成」で学んだ外部ライブラリーを用いることで、ネットワーク通信を意識しないで済ますようにすることができますが、ライブラリーの背後ではWebサーバーへのアクセスが発生しています。

3つ目のケースは、Androidアプリというよりは、サービス全体の設計も含めて考える職業の場合に発生します。例えば、ベンチャー企業で、全く新しいWebサービスを運営しようとして、Androidアプリとセットで開発することになった場合などがこれに相当します。また、すでに存在しているWebサービスをAndroidアプリと緊密に連携させるため、Webサーバー側も修正しつつ、Androidアプリを実装するという状況も少なくありません。



18-1-3 Web サーバーを支える技術

Webサーバーをゼロから作るのは大変なので、色々な技術や商用サービスを組み合わせて構成するのが一般的です。Web開発の話題になると、しばしば名前が挙がる技術やサービスをいくつかあげてみます。聞いたことがある名前が含まれているかもしれません。

- ・ **Google App Engine(今回利用するもの)**
- ・ **Amazon Web Services(略して「AWS」)**
- ・ **Virtual Private Server(略して「VPS」)**
- ・ **Ruby on Rails, Python Django, Heroku, ...**
- ・ **Apache Tomcat(Jetty, JBOSS, ...)**
- ・ **Microsoft Azure**
- ・ **Parse**
- ・ **自分でコンピュータを買ってきて回線契約を結んで運用する**

これらは提供するサービスの性質に応じて「SaaS」「PaaS」「IaaS」「オンプレミス」などと分類されることがあります。大まかに言うと、誰が物理的なコンピューターを管理して、OSそのものの機能をどこまで使うか、が異なります。本節はサーバーを含めた「ITインフラ」の講習ではないので、この辺りの事情は詳述しません。

本章ではその中で、Googleが提供するWebアプリケーションフレームワークのひとつである、「Google App Engine」を選択します。



18-1-4 Google App Engine とその Python 実装について

「Google App Engine」(略してApp EngineもしくはGAE)は、グーグルが提供するWebアプリケーションフレームワークのひとつです。これは上述した中では「PaaS」(Platform as a Service)のカテゴリに入ります。Webサーバーの機能を自分で作り込める一方、LinuxやWindows ServerといったOSについての深い知識を必要としません。

GAEでは現在Java、Python、PHP、Goの4種類の言語を使うことができます。本節ではこの中で「Python」(パイソン)言語による実装を用います。

Pythonは、Javaと同様に世界的にもよく利用されているプログラミング言語です。プログラミング言語の人気を測る指標ではしばしば上位を占め、マサチューセッツ工科大学(MIT)やカリフォルニア大学バークレー校といったコンピュータ科学教育で有名な大学で、プログラミングの入門コースで採用されるなど、プログラミング教育でも豊富な実績があります。

主な目的はWebサーバーのエキスパートになることではなく、仕組みを理解することです。PythonやGAEの習得が目的ではないので、演習はPythonを深く理解する必要がないように構成しています。

Androidと同様Java言語を採用したいところですが、準備がGAEのPython版と比べると複雑で、さらに既存のAndroid開発環境を壊してしまう可能性が懸念されたため、本講義ではGAEのJava版は採用しません。

Pythonについては18-3「付録」で基礎を解説しています。



18-1-5 本演習を行う上での注意事項

本演習では、実装したWebサーバーを全世界に公開します。Googleアカウントによるログインを除いて、個人情報やプライバシーに関わる情報を実装に含めるのは避けてください。

また、Androidアプリ開発とは関係がないソフトを複数インストールします。あらかじめ、元のシステムのバックアップを作成しておいた方が良いでしょう。Windows 8.1の場合について、付録18-3-3の「Windows 8.1でのシステムのバックアップ方法」で、その方法を説明しています。ユーザー名が日本語のWindows環境を使っている場合、先に18-3-4を参照してください。



18-1-6 Python 実行環境のインストール

GAEを用いてWebサーバーを開発するために、まずPython実行環境をPCへインストールします。インストールするバージョンはPython 2.7.8です。以下のURLからWindows向けPythonインストーラーを取得します。「Windows X86 MSI Installer (2.7.8)」を選択してください(図2)。Pythonには3で始まる新しいバージョンがありますが、今回はインストールしないでください。

64bit版として「Windows X86-64 MSI Installer(2.7.8)」も選択肢として表示されますが、今回は選ばないようにしてください。後述するGAEのSDKが32bit版しか提供していないためです。

<https://www.python.org/downloads/release/python-2.7.8/>
[\(https://www.python.org/ftp/python/2.7.8/python-2.7.8.msi\)](https://www.python.org/ftp/python/2.7.8/python-2.7.8.msi)

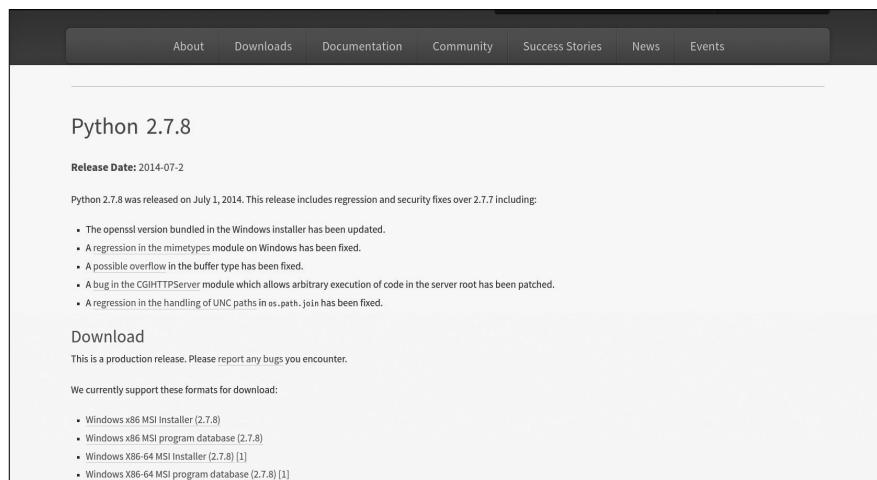


図2:「https://www.python.org/downloads/release/python-2.7.8/」からPythonのインストーラーをダウンロードする

ダウンロードした「python-2.7.8.msi」というファイルを実行すると、図3のような画面が表示されるはずです。ここでは「Install for all users」が選択された状態で「Next >」ボタンを押します。



図3:「Install for all users」を選んで「Next >」をクリックする

インストール先を確認されます。「C:¥Python27¥」となっていることを確認して「Next」を押します(図4)。

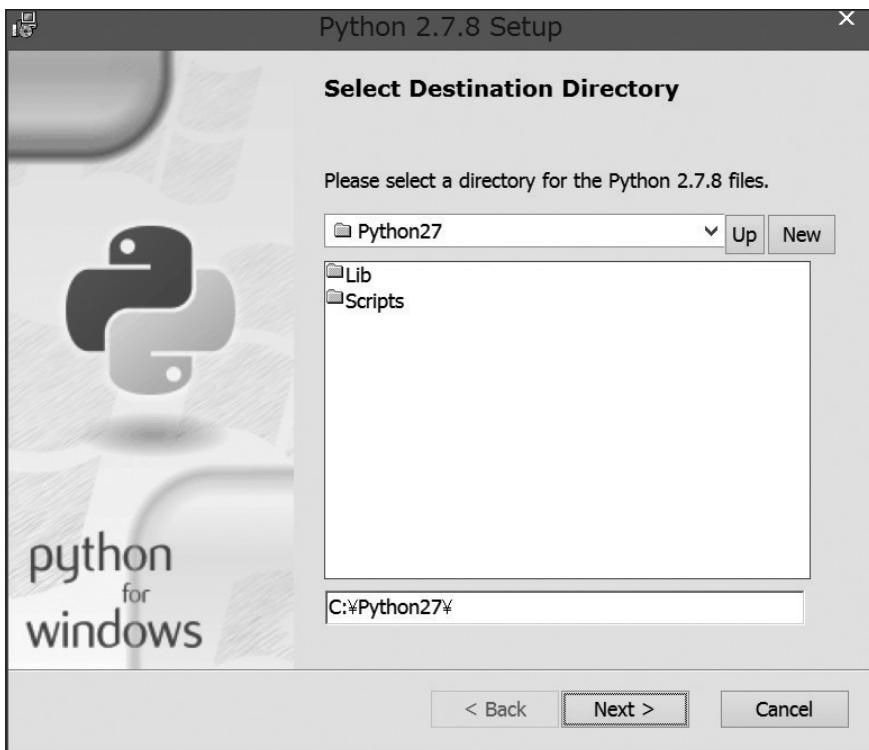


図4:インストール先の「c:\Python27\」を確認して「Next >」をクリックする

その後、図5のような画面が表示されたら「Add python.exe to Path」をクリックし、表示される2つの選択肢のうち「Will be installed on local hard drive」を選択します。

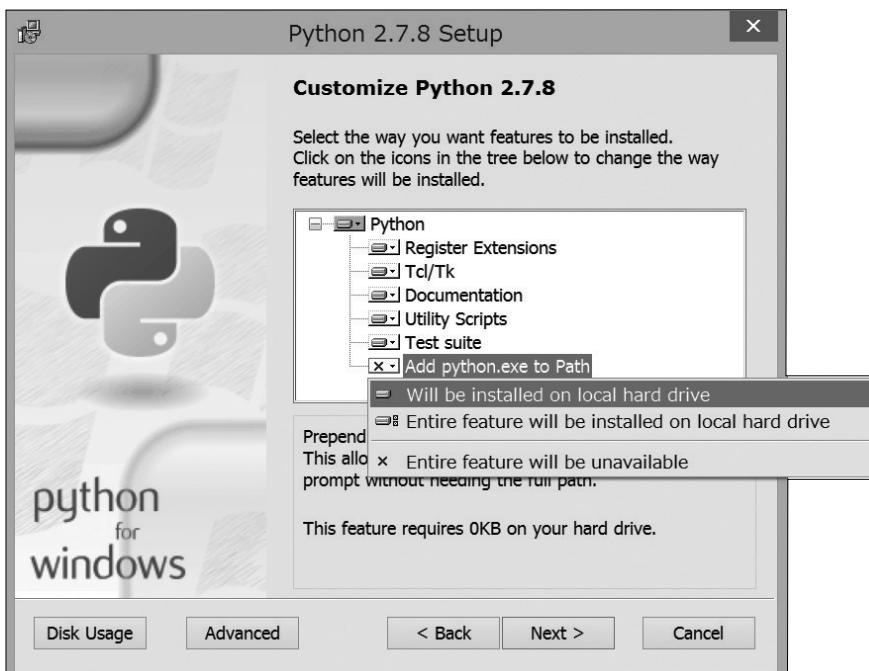


図5:Pythonをローカルなハードディスクにインストールする選択肢を選ぶ

すると×印がなくなるので「Next >」を押します(図6)。



図6:「×」の表示がなくなった状態で、さらに「Next >」をクリックする

図7のように再起動を要求されることがあります。このときは、PCを再起動してください。

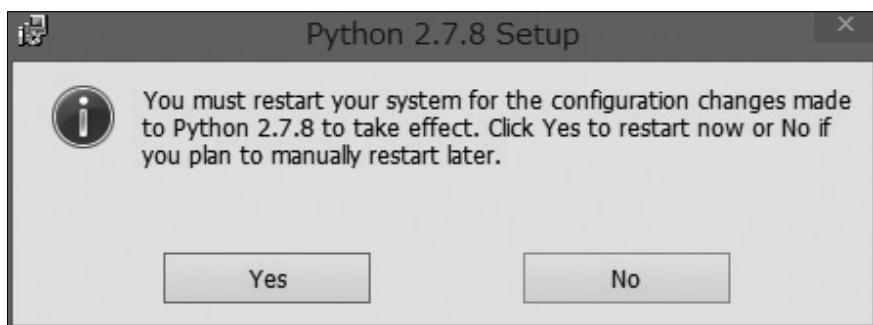


図7:再起動を要求されたら「Yes」をクリックしてPCを再起動する

以上の作業が終了したら、Python実行環境の動作を確認します。Windows画面左下のスタートボタンを右クリックして「コマンドプロンプト」を選択後、「Python」と入力して、Pythonの対話型環境が起動することを確認してください(図8)。

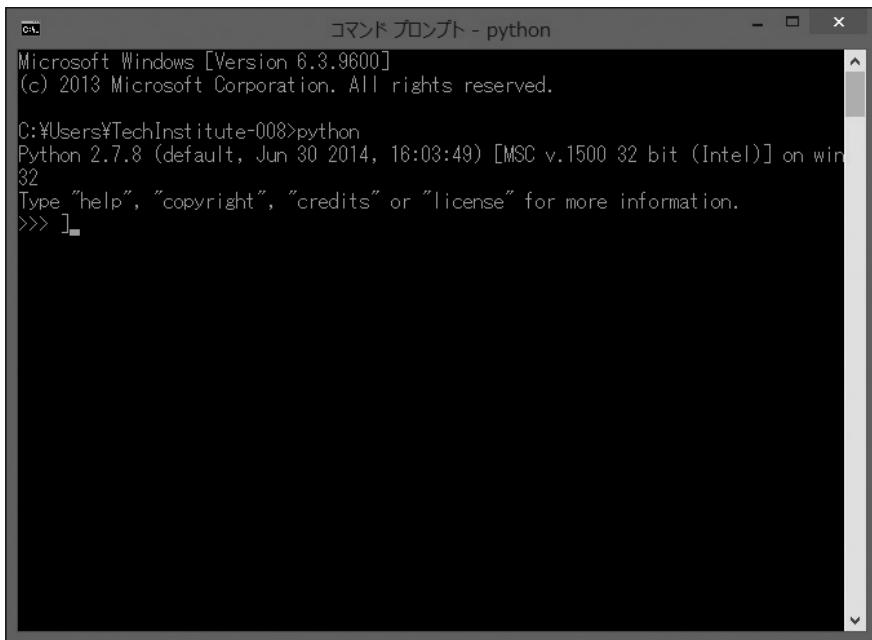


図8:コマンドプロンプトに「python」とタイプ入力して応答を確認する

18-1-7 Google App Engine SDK のインストール

次に、「Google App Engine」の「Python SDK」をインストールします。ブラウザで以下のURLを開いてください。

<https://developers.google.com/appengine/downloads>

そのページにある「Google App Engine SDK for Python」をクリックします(図9)。



図9:「GAE」のダウンロードページで「Google App Engine SDK for Python」をクリックする

さらに、適合するOSのインストーラーを選びます(図10)。

Note: The App Engine SDK is under **active development**; please keep this in mind as you explore its capabilities. See the SDK Release Notes for information on the most recent changes to the App Engine SDK. If you discover any issues, please feel free to notify us via our Issue Tracker.

Platform	Version	Package	Size	SHA1 Checksum
Windows	1.9.11 - 2014-09-11	GoogleAppEngine-1.9.11.msi	46.5 MB	933b516aef8556da2492ec7d4bc86b6330a7181d
Mac OS X	1.9.11 - 2014-09-11	GoogleAppEngineLauncher-1.9.11.dmg	54.0 MB	61a888ac202a05a2946048cec7a3293571b28783
Linux/Other Platforms	1.9.11 - 2014-09-11	google_appengine_1.9.11.zip	51.8 MB	f16468418433eb762aca4a509dc5a28ad77448f1

Installing on Linux

https://storage.googleapis.com/appengine-sdks/featured/GoogleAppEngine-1.9.11.msi

appengine-try-py...zip

スクリーンショットを追加しました Dropbox フォルダにスクリーンショットを追加しました。 すべてのダウンロードを表示

図10:Windows PCであればWindows版のmsiファイルを選ぶ

ダウンロード後、インストーラーを起動します。SDKインストーラーが「Python 2.7」を認識していることを確認した上で、「Next」ボタンをクリックします(図11)。

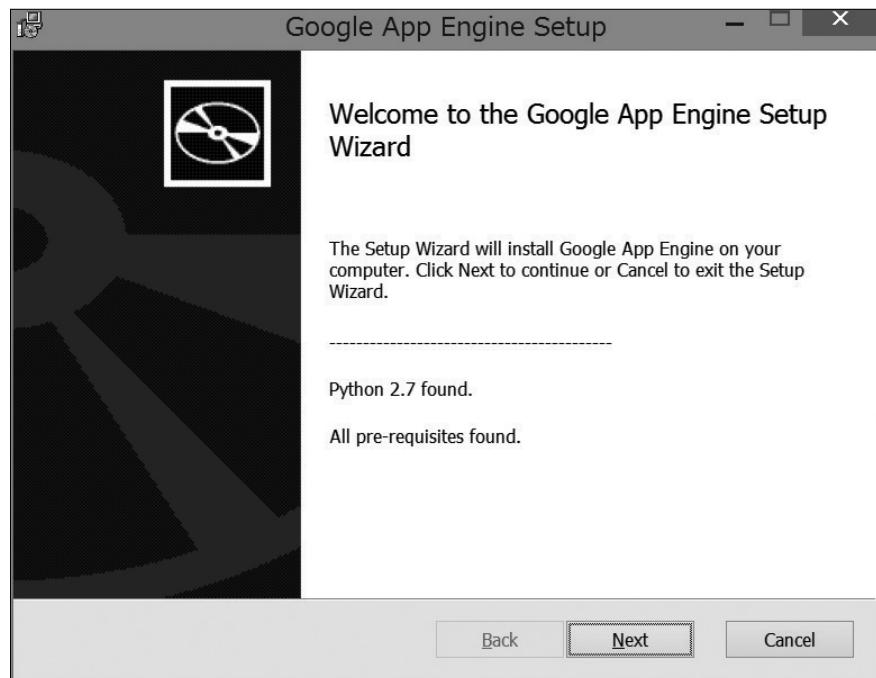


図11:「Python 2.7」の表示を確認して「Next」をクリックする

続いて表示される「End-User License Agreement」では、「I accept the terms in the License Agreement」にチェックを入れてから、「Next」を押します(図12)。

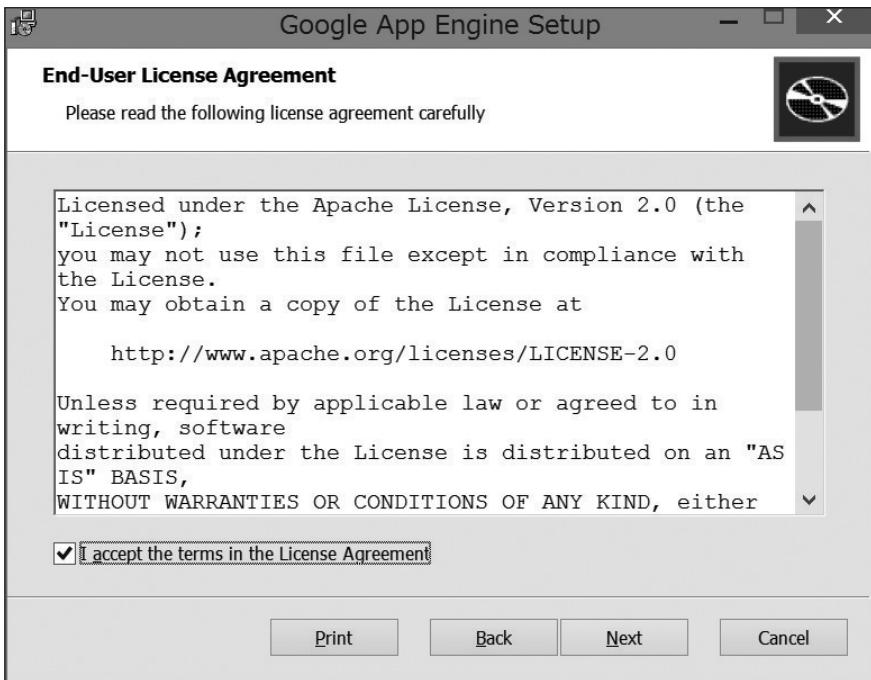


図12:ライセンス許諾条項を確認してから「Next」をクリックする

次のダイアログでは、インストール先のフォルダーは変更しません。表示される3つのチェックボックス全てにチェックが入っていることを確認して、「Next」を押します(図13)。

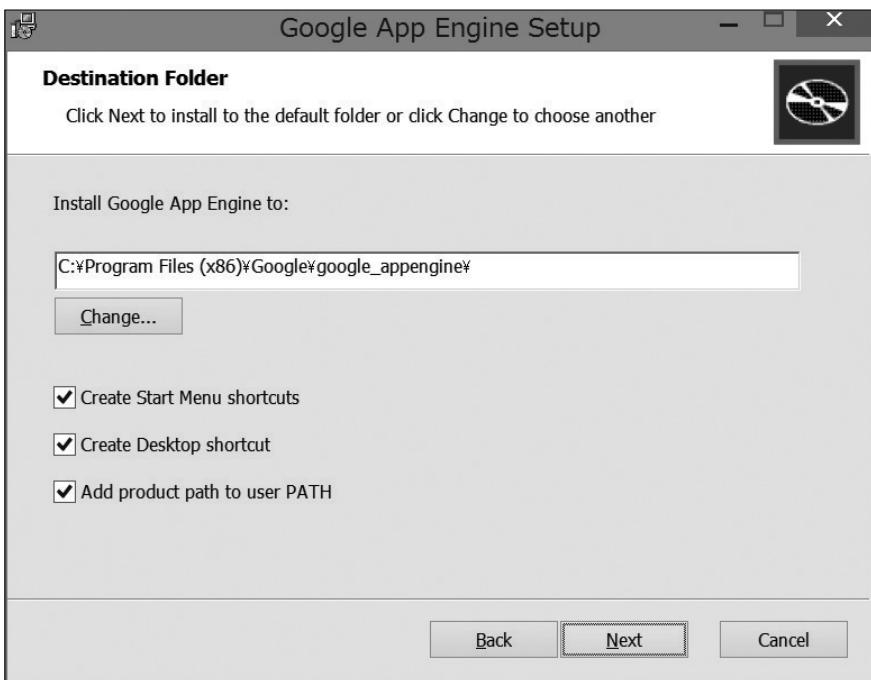


図13:3つのチェックボックスがすべてオンであることを確認して「Next」をクリックする

最後に「Install」ボタンをクリックします(図14)。

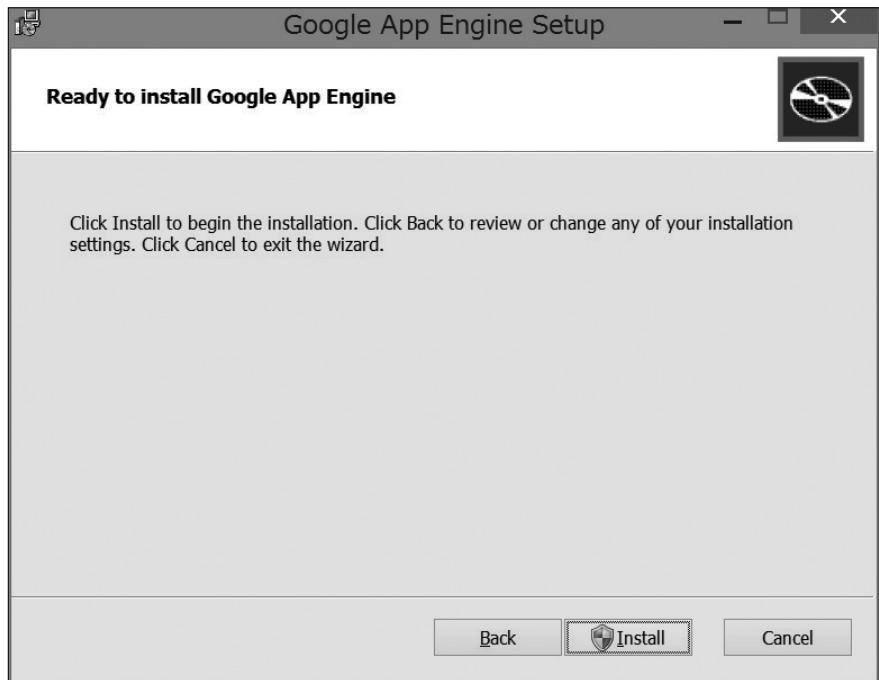


図14:「Install」をクリックする

GAEのインストール中には、ダウンロードしたソフトウェアを信用するかどうかを確認するシステムダイアログが表示されます。このときは、「確認済みの発行元」が「Google Inc」となっていることを確認した上で、「はい」を押します。

インストールが成功したら、「Run Launcher」ボタンを押して選択してから、「Finish」ボタンを押します(図15)。

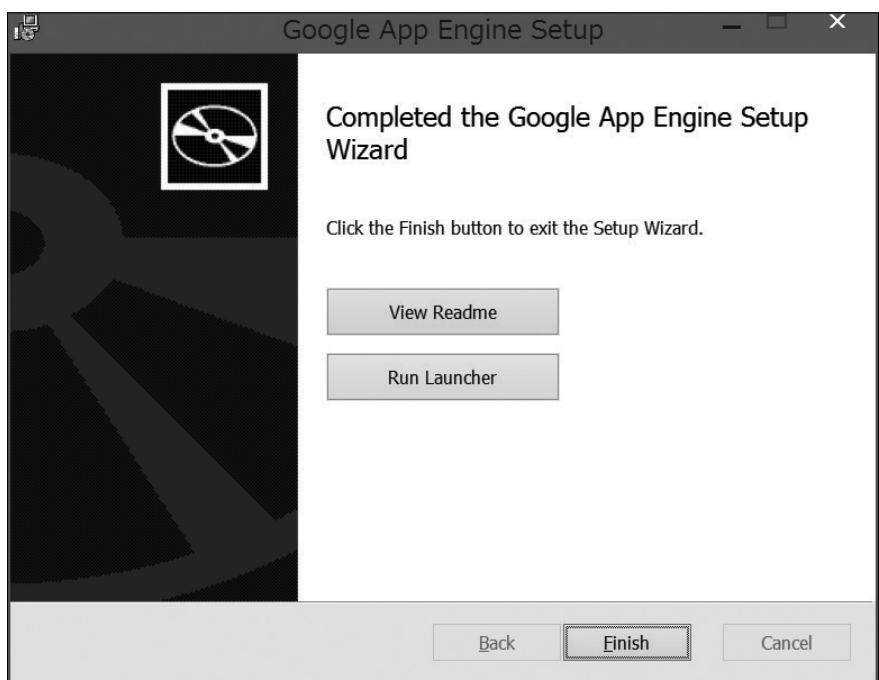


図15:「Run Launcher」を選択して、「Finish」ボタンをクリックする

この後、「Google App Engine Launcher」というウィンドウが開けば、「GAE SDK」のインストールに成功したことがわかります(図16)。

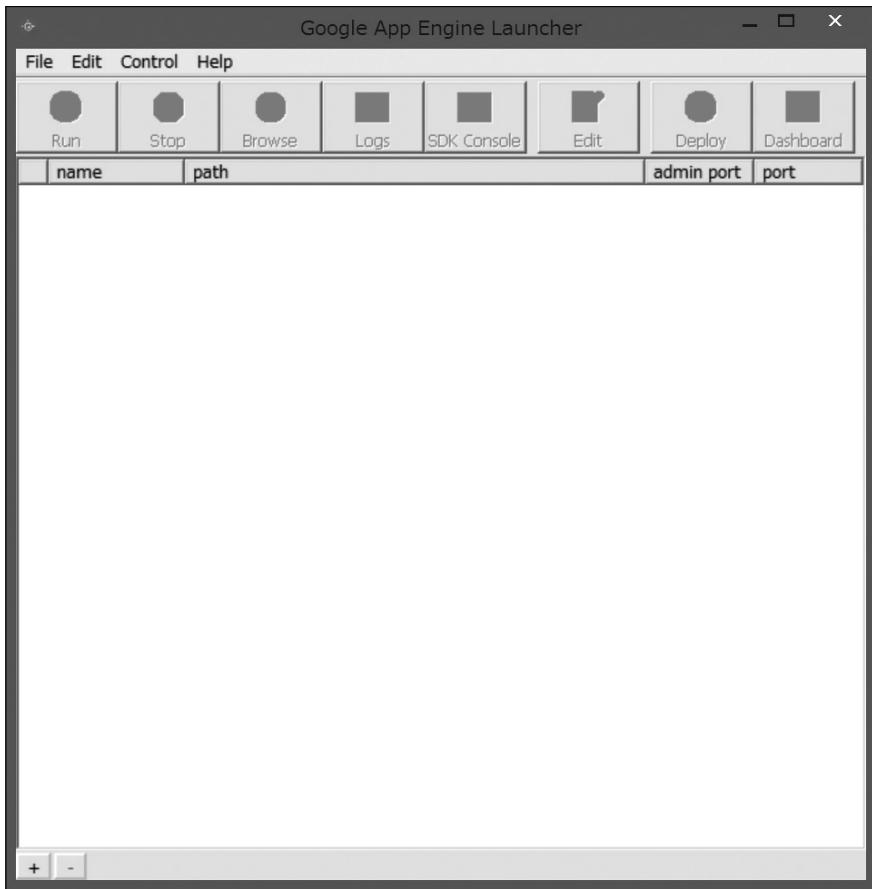


図16:「Google App Engine Launcher」ウィンドウが開いた

Windows 8等でユーザー名が日本語になっている場合、ここからさらに作業が必要です。付録の「ユーザー フォルダーに日本語が含まれる場合のGAEインストールの追加作業」を参照してください。



18-1-8 ローカルサーバーでHello World!

ここでは、今インストールしたGAEのプロジェクトを作成し、「Hello world!」と表示するだけのWebサーバーを作成しましょう。「Google App Engine Launcher」の「File」メニューから「Create New Application」を選択します。表示される「Add New Application」ダイアログに、以下の設定を加えます(図17)。

・Application Name を「helloworld」等に設定する

ここでは一部の文字、例えば半角大文字は使用できません。半角小文字の英字と数字だけで構成するのがベストです。

・Parent Directory を「デスクトップ」に設定する

日本語ユーザーである場合は、別途用意したフォルダーを設定します。

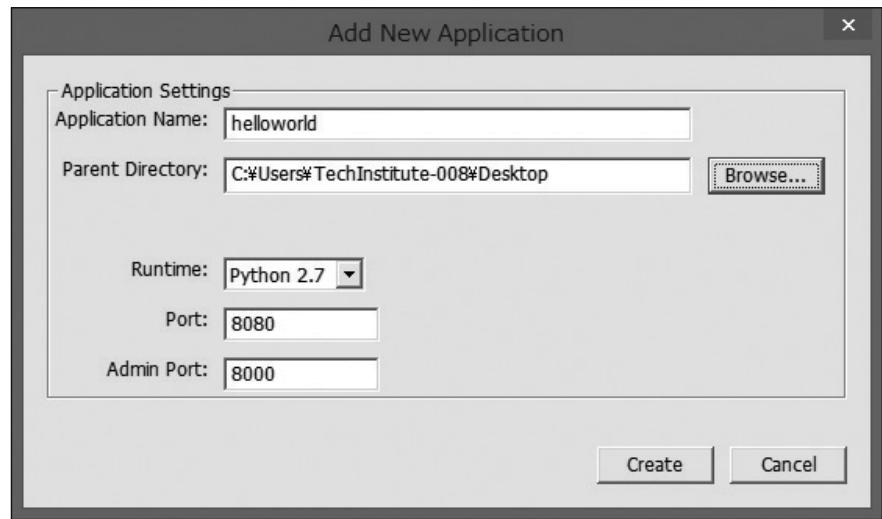


図17:アプリケーション名と保存ディレクトリを指定する

プロジェクトが作成できたら、「Google App Engine Launcher」でそのプロジェクトを選択し、「Run」をクリックします(図18)。

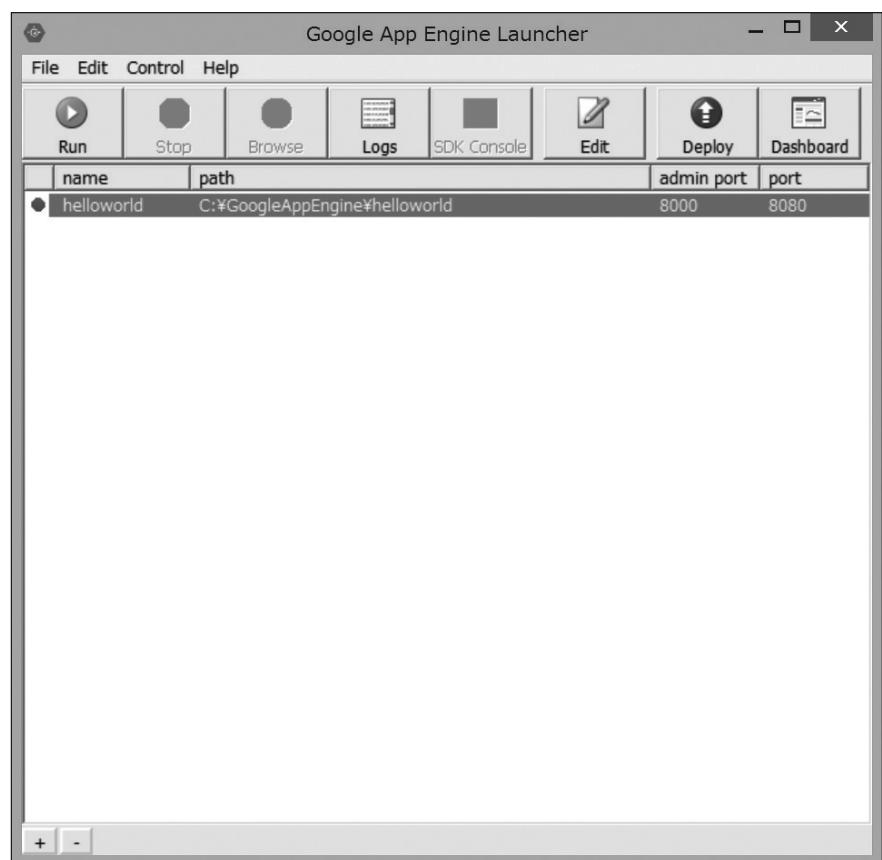


図18:作成した「helloworld」プロジェクトを選択して「Run」をクリックする

起動に成功すると、プロジェクトの左にある小さなアイコンが緑色の「再生ボタン」のようになります(図19)。

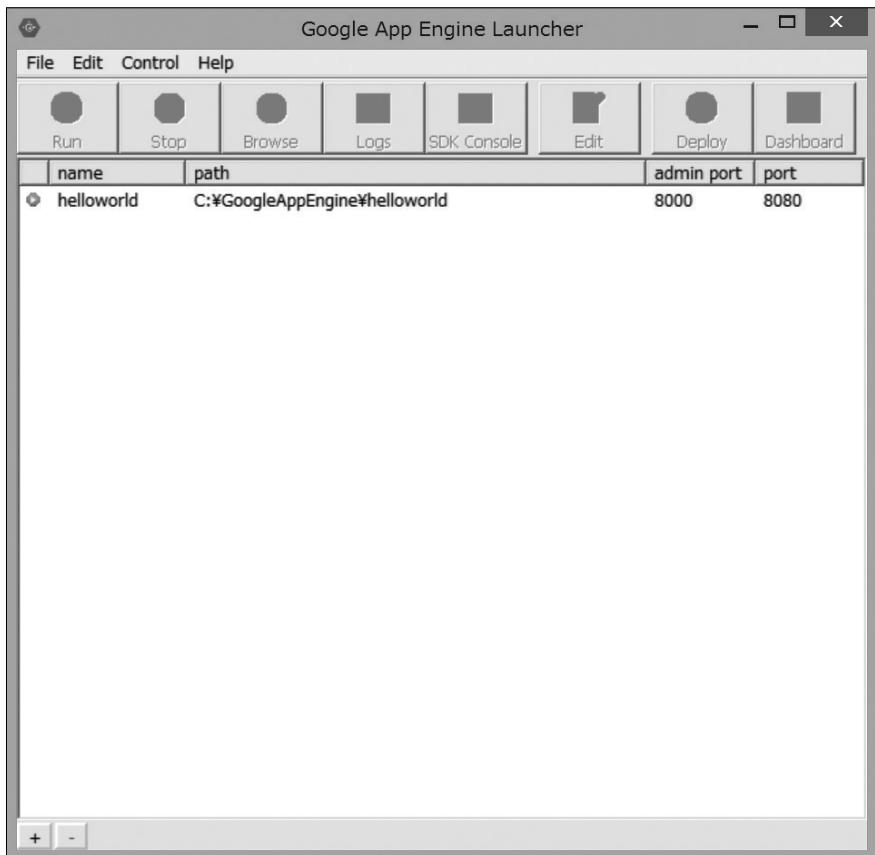


図19:「helloworld」の横のマークが緑の「再生ボタン」になれば成功

この状態でブラウザから「`http://localhost:8080`」を開きます。ウィンドウに「Hello world!」と表示されたら成功です(図20)。

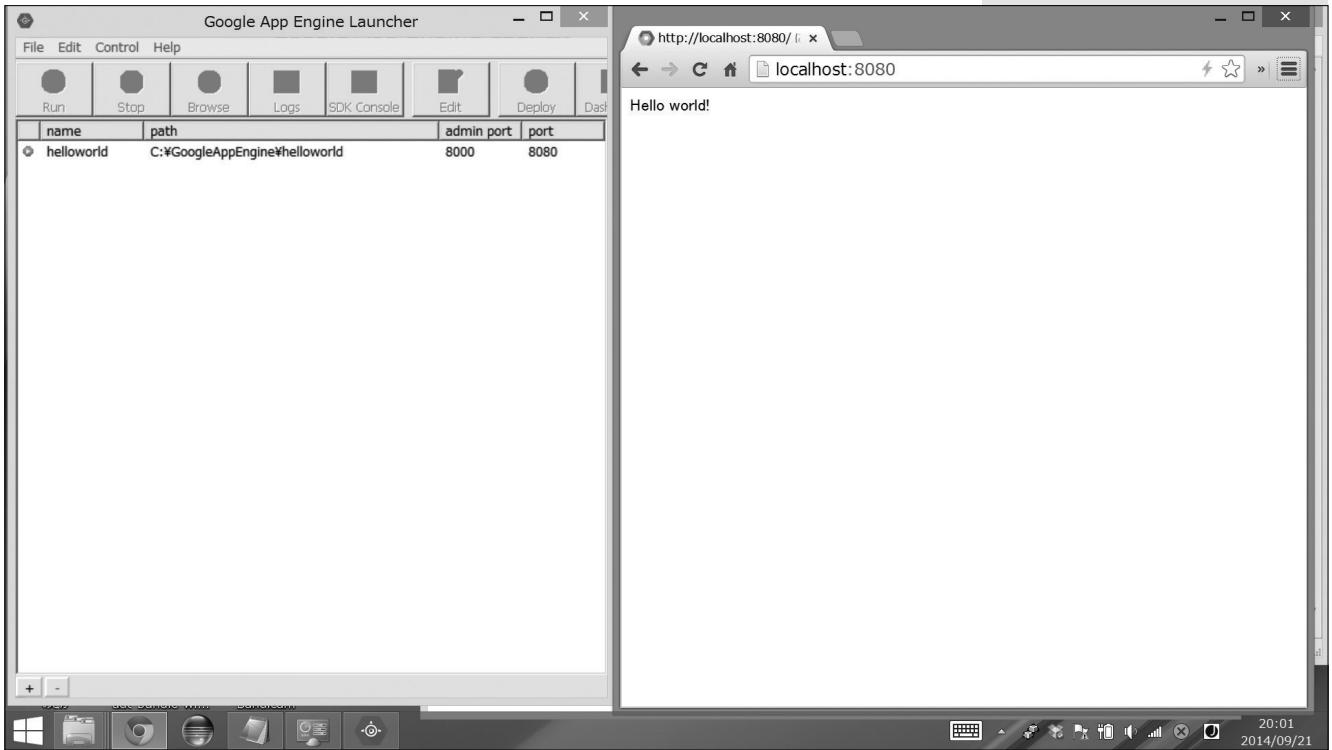


図20:Webブラウザで「`http://localhost:8080`」を開くと「Hello world!」と表示される

サーバーが利用するTCPポート番号は、複数のプロジェクトを生成すると変化する可能性があります。その場合、以降の説明の数字を変更してください。ポート番号そのものについては、本講座テキスト第14章「ネットワークプログラミング」の「14-1-5: ポート番号とは?」を参照してください。

ポート番号に8000番を指定すれば、このプロジェクトの管理用Webページを開くこともできます。

このサーバーは現在「実行中」の状態です。これを停止するには、「Google App Engine Launcher」の「Stop」ボタンを押します。「再生」ボタンが黒い丸印になれば、サーバープロセスが停止したことがわかります(図21)。

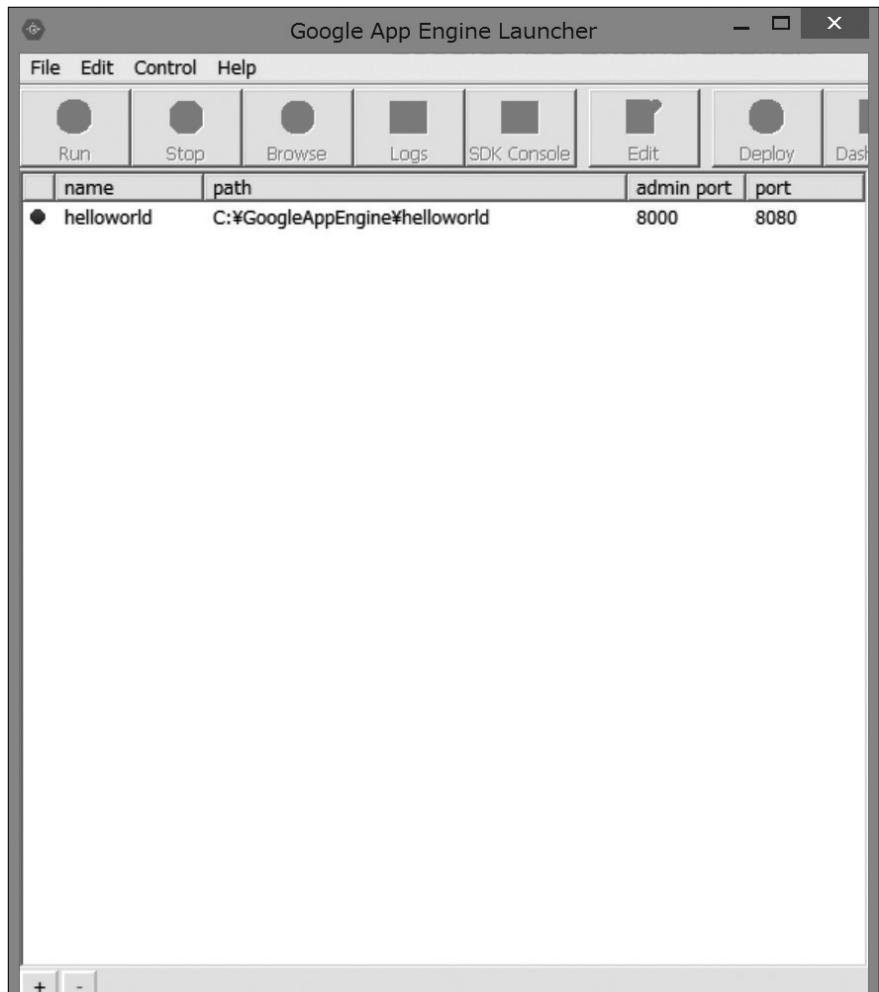


図21:サーバー停止時には「helloworld」の左のマークは黒い丸印になる

ブラウザに入力したURLについて、ここで説明しておきましょう。

- ・「**http**」はどのようにサーバーに接続するかを決めたもの(スキームと呼ばれる)
- ・「**localhost**」は「自分自身」で、つまりPCのことで、「ローカルサーバー」と呼んでいる
- ・**TCPポートの8080番でサーバーを作成している**

という意味になります

この「Hello world!」を表示するまでに、プログラムを1行も書く必要がありませんでした。これでは、ちょっと物足りなさすぎです。



18-1-9 PyCharm Community Edition のインストール

これから、Pythonでプログラムを書いてサーバーを拡張するのですが、Windows付属の「メモ帳」は、文字コードの扱い、特に改行コードの処理に難点があるため、Pythonの開発には不適切です。また、PythonでもEclipseのような統合開発環境(IDE)を利用できると、文法チェックなど、多くの面で便利でしょう。

ここでは、「PyCharm」と言う商用IDEの無料版「PyCharm Community Edition」をインストールします。

まず「<http://www.jetbrains.com/pycharm/download/>」をブラウザーで開きます(図22)。右側の「Community Edition」を選択してダウンロードし、そのインストーラーを実行して指示に従います。

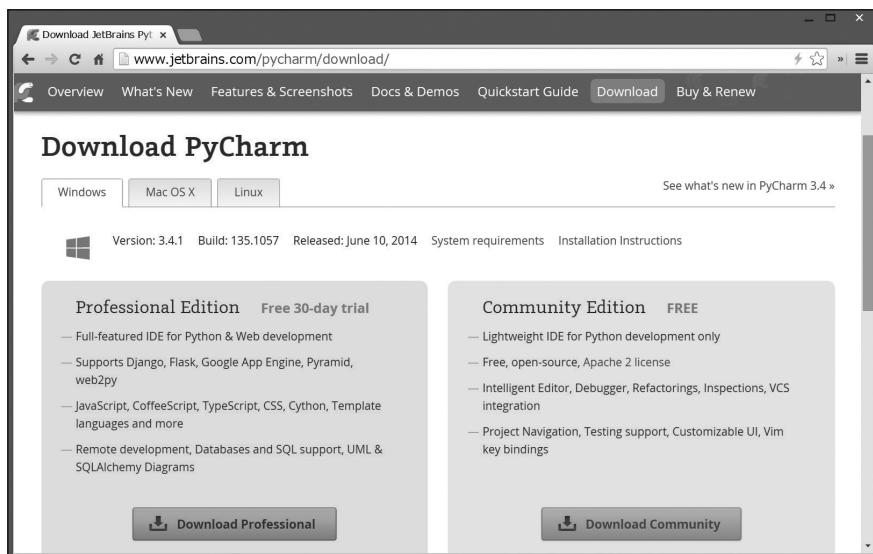


図22:PyCharmのダウンロードページ。右側の「Community Edition」をダウンロードする

途中で表示される「Create Desktop shortcut」(デスクトップにショートカットを作成する)のチェックボックスをオンにしておくと良いでしょう(図23)。

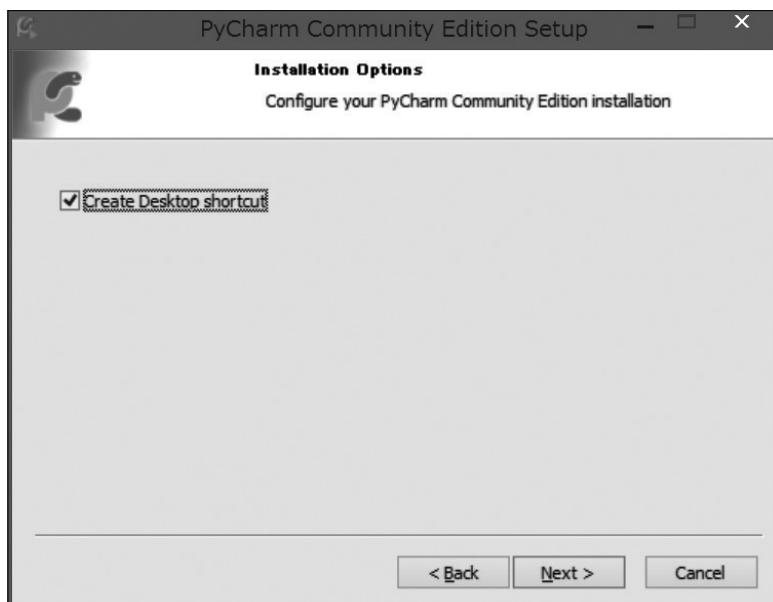


図23:「Create Desktop shortcut」をチェックしておくと、演習を効率的に行える

他に『秀丸』等の高機能なエディタがあればそれを利用しても構いません。その場合はPyCharmについての以下の説明は無視してください。

EclipseをPython開発環境として利用するための「PyDev」というEclipseプラグインが存在します。しかし、本講義において環境にインストールされているJavaのバージョンは若干古いため、そのままでは利用できません。Javaのバージョンを7にすれば「PyDev」の動作要件を満たしますが、Android開発環境を壊してしまう可能性もあるため、本演習では採用しません。

インストールの最後に表示されるダイアログで、「Run PyCharm Community Edition」をチェックして「Finish」ボタンを押しましょう(図24)。

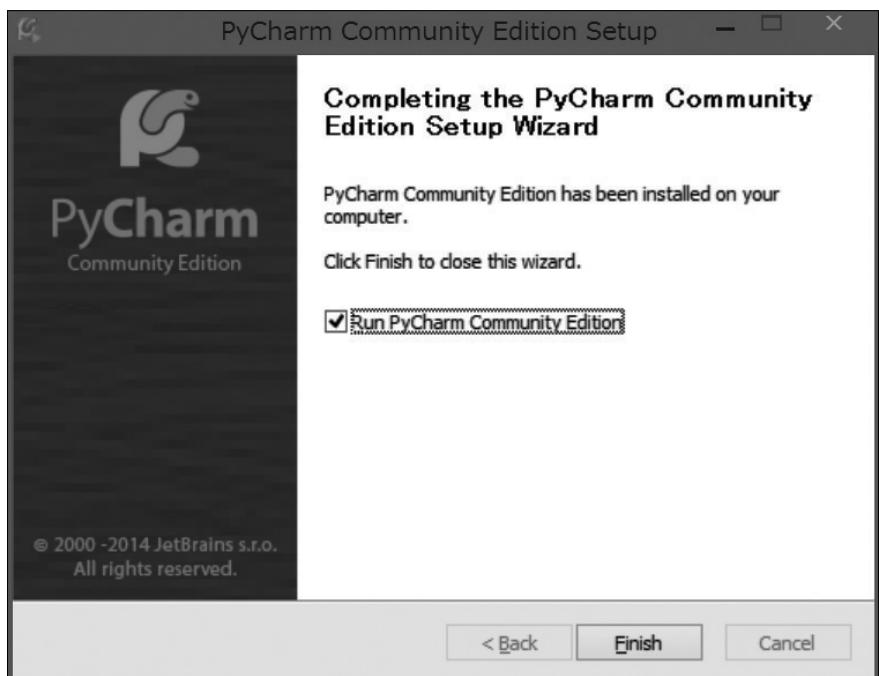


図24:「Run PyCharm Community Edition」をチェックして「Finish」をクリックする

初回起動時に図25のような表示が出ます。今回はそのまま「OK」を押します。



図25:初回起動時に表示される「Complete Installation」ダイアログ。そのまま「OK」をクリックする

さらに図26のような画面が表示されます。ここで、Eclipse風のキーバインドにしたい場合は「Keymap scheme」を適切に変更します。なお、この設定は後述する通り、変更可能です。

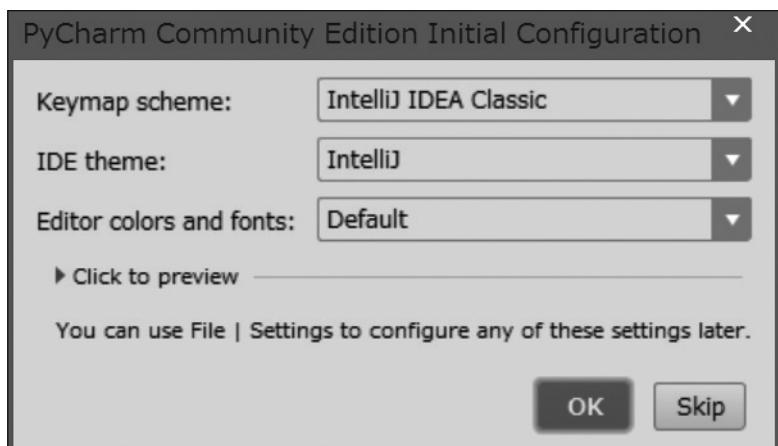


図26:初回起動時には、開発環境の初期設定をどうするか聞いてくる

ここで「OK」ボタンを押すとPyCharmが起動します(図27)。

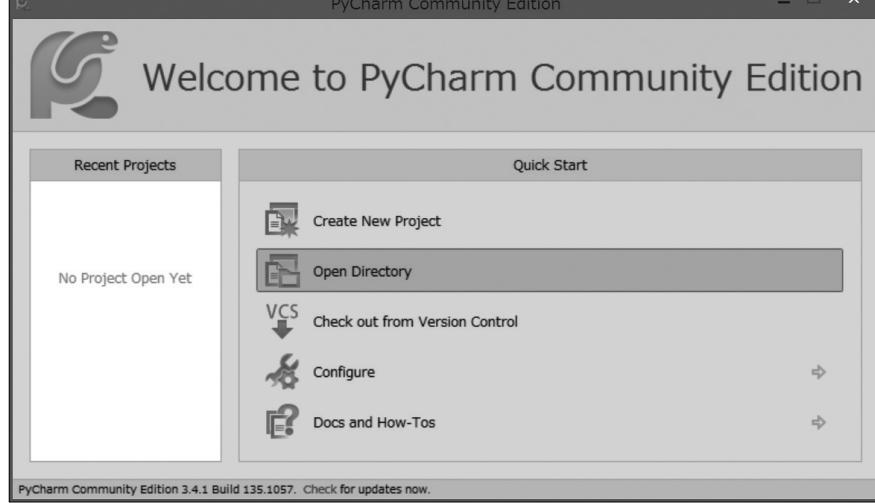


図27:PyCharmの起動が成功した

なお、Windowsの設定によっては、図28のような警告が表示されることがあります。ここではそのまま「アクセスを許可する」ボタンを押します。



図28:Windowsのファイアウォールの設定によっては警告が表示されることがある

既に「helloworld」プロジェクトを作っていますので、ここではそれをPyCharm上で編集してみましょう。図27のダイアログの「Open Directory」をクリックし、「helloworld」プロジェクトが存在するフォルダーを開きます(図29)。

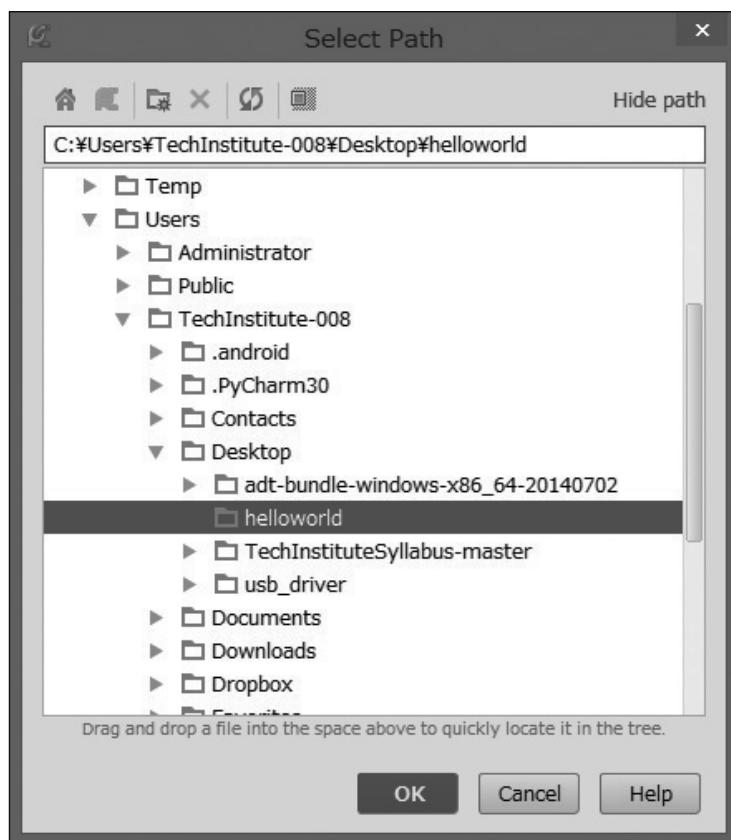


図29:「helloworld」プロジェクトの場所を指定する

その結果、図30のようなウインドウが表示されれば、うまくプロジェクトが開いたことがわかります。

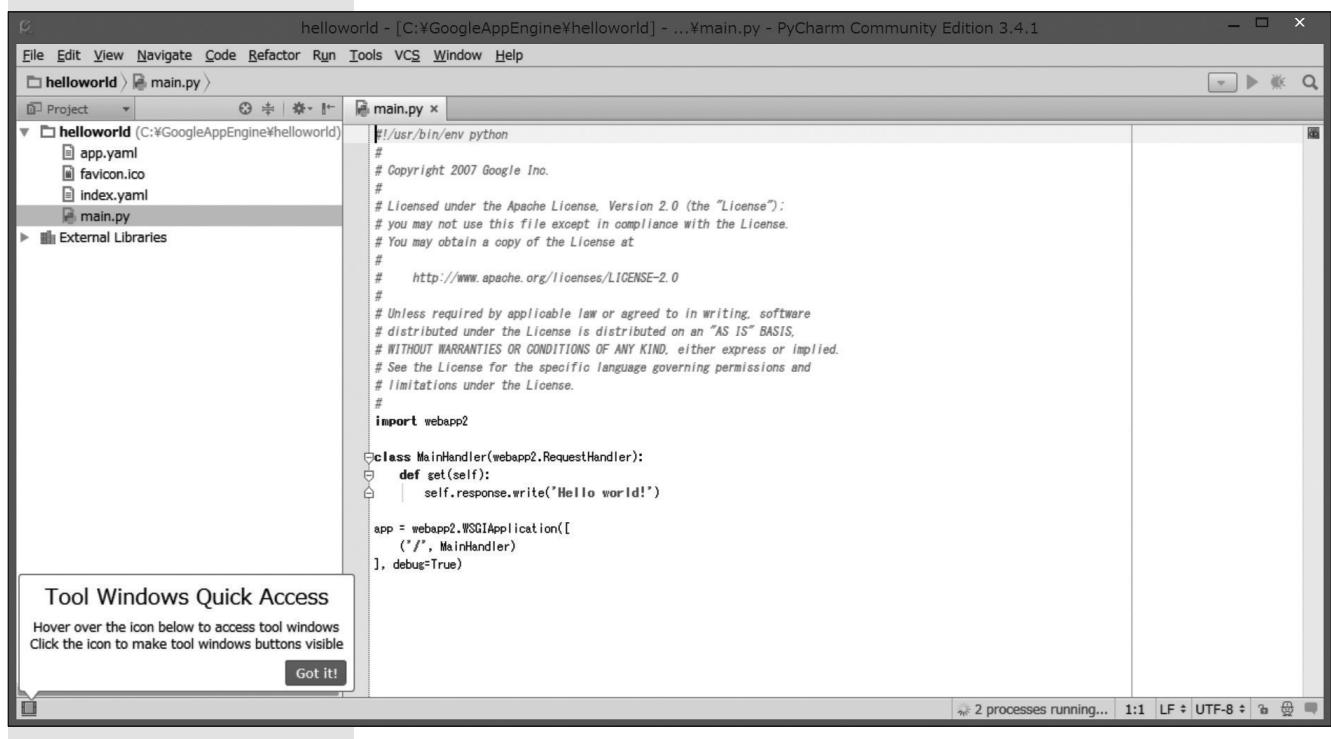


図30:「helloworld」プロジェクトをインポートしたウインドウが開いた

このとき、図31のようなダイアログが表示されることがあります。



図31:PyCharmのTipsを表示してくれるが、今回は不要なので閉じておく

ここでは左下の「Show Tips on Startup」(起動時にTips画面を表示する)のチェックを外して、「Close」ボタンを押します。

PyCharmは同じく統合開発環境であるEclipseと外見が似ています。しかし、使い勝手が全く同じというわけではありません。以下に、すぐに使い勝手を改善できるヒントを示しますので、興味があればさらにカスタマイズしてみましょう。

起動時にkeymapを変更していない場合「File」→「Settings」→「Keymap」でKeymapsを「Eclipse」に変更すると、Eclipse風になります(図32)。ただし全てがEclipseと同じになるわけではありません。

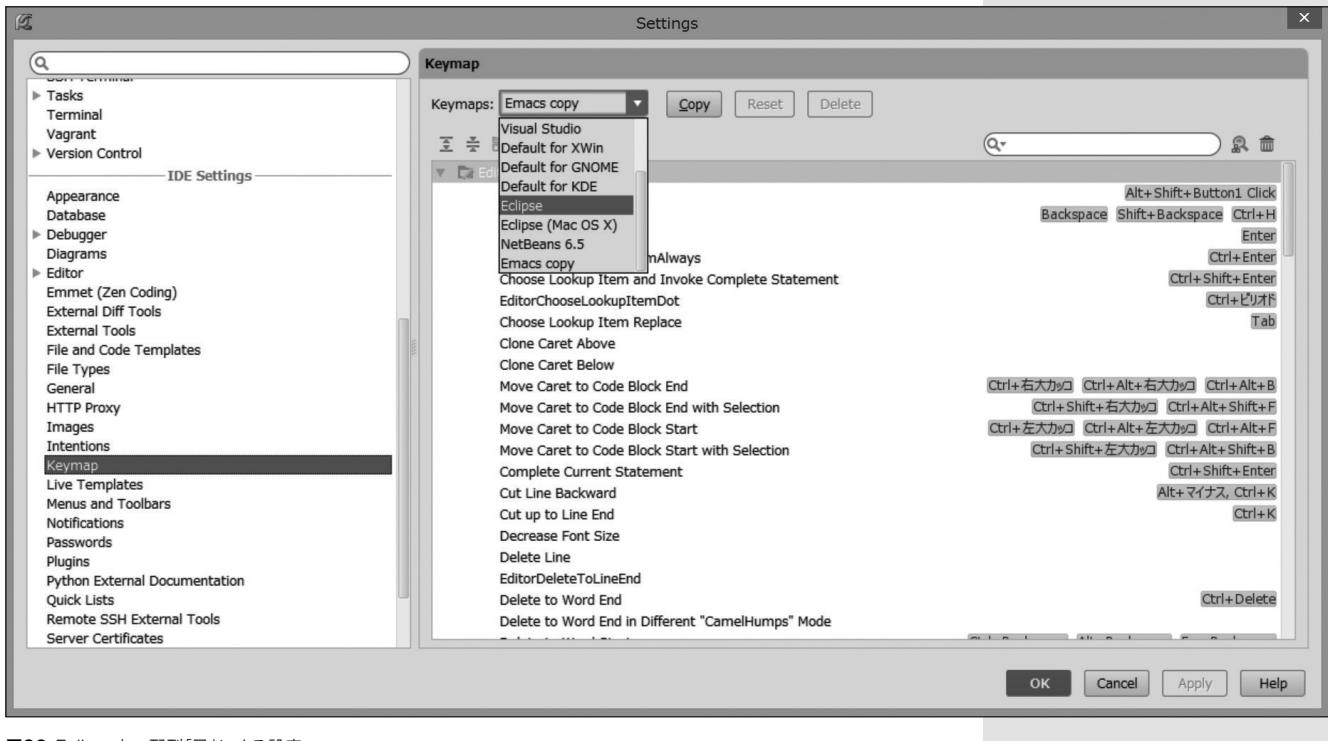


図32:Eclipseキー配列「風」にする設定

「File」→「Settings」→(左画面IDE Settingsの)「Editor」→「Editor Tabs」→「Mark modified tabs with asterisk」のチェックを入れておくと、Eclipseと同様に、保存されていないファイルのタブに「*」(スター、もしくはアスタリスク)が付きます(図33)。

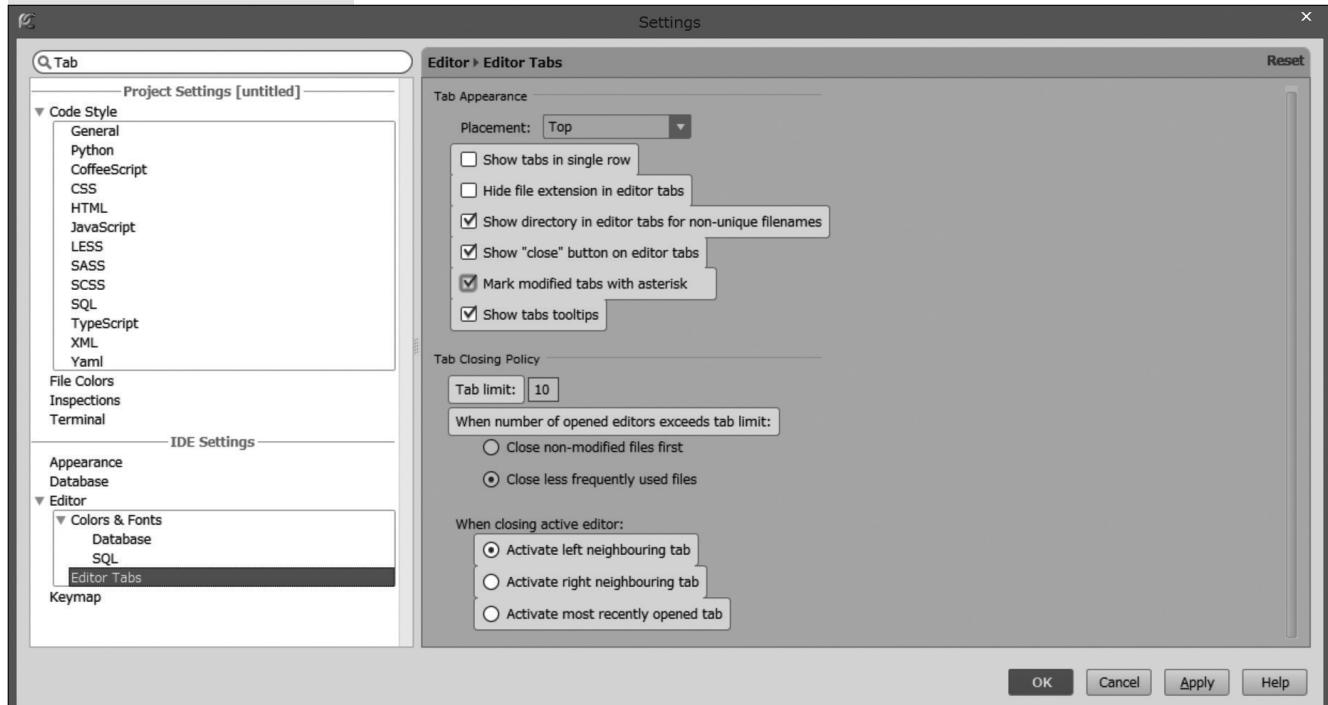


図33: ファイルの変更を保存していない時に気づきやすくなる設定

ここまででようやくPythonでプログラミングする準備が整いました。プログラミングに自信があれば、このままGAEによるWebサーバーの実装を進めましょう。

18-1-10 Hello World サーバーへブラウザからアクセスする

こんどは「Hello world!」以外の文字列を出力してみましょう。PyCharmの画面左の、「helloworld」となっている部分をクリックし、展開したファイル一覧を確認してください。以下のファイルが見えるはずです。

- app.yaml**
- favicon.ico**
- index.yaml**
- main.py**

それぞれのファイルの意味は以下のとおりです。

app.yamlとindex.yamlは、Androidで言えばAndroidManifest.xmlに相当するような、アプリの設定ファイルのことです。

favicon.icoはWebページを開いたときにタブに表示されるアイコンです。
main.pyはWebアプリケーション本体のソースコードです。
主に他の3つのファイルを編集することで機能を追加していきます。「main.py」のPythonコードは、コメントを除いて次に示します。

main.py:GAEを用いたWebサーバーの全文

```
import webapp2

class MainHandler(webapp2.RequestHandler):
    def get(self):
        self.response.write('Hello world!')

app = webapp2.WSGIApplication([
    ('/', MainHandler)
], debug=True)
```

「Hello world!」という文字列を変更したいだけなら、次に示す行だけを変更すればよさそうです。

mai.y.py(部分)

```
self.response.write("Hello world!")
```

「helloworld.py」の内容を変更してファイルを保存したら、「Google App Engine Launcher」で再びローカルサーバーを起動して変更を確認してみましょう。



18-1-11 Hello World! プログラムの意味

Pythonプログラミングに関する部分を除くと、今回のプログラムの意味はおおよそ次のとおりになります。

まず、「webapp2」というPython特有のライブラリーでWebサーバーを実装しています。Webサーバーを実装するためのフレームワークは、使用するプログラミング言語に関わらず似ている部分があります。今回は、細かな部分は気にせず、共通する部分に注目していきます。

Webサーバーを実装する上での「共通部分」とは何かと言えば、クライアントとサーバーの間でやりとりする通信のプロトコル(protocol)です。WebサーバーとクライアントのやりとりにはHTTP(Hyper Text Transfer Protocol)を用います。この言葉は14章「ネットワークプログラミング」でも説明されています。そこでは、HTTPにまつわる用語も多く登場しています。

「MainHandler」というクラスの「get()」関数が実際にWebサーバーで何を返すか見てみましょう。ここでは第14章「ネットワーク」で既に登場している、HTTPの

「プロトコル」という言葉は他の分野では「外交儀礼」という意味で使われたりします。たとえば他国の王族をもてなす際には一定の手順、すなわちプロトコルを踏まえるのがならわしです。この場合、プロトコルを違反すると、場合によっては外交問題になります。

実際には、GAEによって、自動的にこのクラスのオブジェクトが生成されおり、それがリクエストに応答します。

GETリクエストを処理するための「get()」関数を定義しています。HTTPには、目的に応じて何種類かリクエストの種類がありますが、クライアント(Androidアプリやブラウザ)がデータを受け取る場合はGETリクエストが一般的です。データを更新する場合はPOSTリクエストという別のリクエストを用います。POSTリクエストについては18-2で登場します。ここではいったんGETリクエストのみ実装します。GAEでは、実装していないリクエストは適切にエラー処理されます。それ以外のHTTPのリクエストの種類は今回は省略します。

「Hello World」サーバーでは、「app = webapp2.WSGIApplication(...)」という関数に渡されるリストによって、Webサーバーにクライアントからのリクエストが到着した際に、URLのどのパスをどのクラスが担当するかを決めています。

「<http://localhost:8080/>」のポート番号「8080」の後の部分が「パス」(path=経路)です。今回は「/」、つまりパスなしのケース「だけ」を「MainHandler」クラスが受け持つことになっています。

試しにブラウザに入力するURLを「<http://localhost:8080/example>」に変更して、サーバーにアクセスしてみてください。「404 Not Found」と出てくるはずです(図34)。

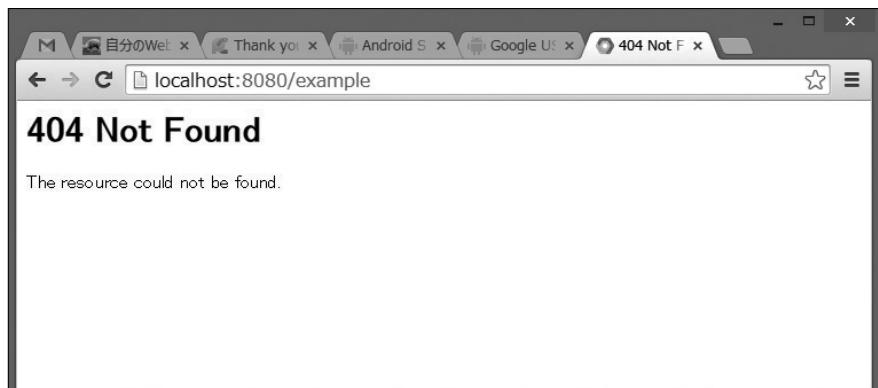


図34:「404 Not Found」というエラーが表示された

「全てのパス」をこの「MainHandler」クラスが受け持つように、実装を変更してみましょう。

mai.y.py(部分)

```
main.pyapplication = webapp2.WSGIApplication([
    ("/.*", MainHandler),
], debug=True)
```

このようにした場合、「404 Not Found」は表示されなくなり、「Hello world!」が再び表示されるようになるはずです。



18-1-12 HTTP のステータスコードについて

「404 Not Found」は、普段のWebブラウジングの際にも見たことがあるかもしれません。HTTPというプロトコル、つまりルールで、サーバーはそのリクエストに対応するリソースがない場合、404番という番号をクライアントに返すことを期待されています。

クライアントがサーバーにリクエストを送信した際、成功しても失敗しても、サーバーからは何らかの数字が返されます。

これをステータスコードと呼びます。Webブラウジングしているときには意識しませんが、Webページが正しく存在している時、サーバーからは「200番」という数字が返されます。

ステータスコードは左端の桁の数字が主要な理由を示しており、残りの2桁でより具体的な状況を説明しています。いくつか挙げてみます(表1)。

ステータスコード	意味
200 OK	リクエストで要求されたものが存在したので送ります
301 Moved Permanently	リクエストしたリソースは永久に別の場所へ移動しました
404 Not Found	そんなリソースは知りません
508 Internal Server Error	サーバー内部でエラーが発生しちゃった!

表1:主なステータスコードとその意味

なお、「404 Not Found」の場合でもコンテンツを返せないわけではありません。GitHubのWebサイトで存在しないページへアクセスした時の例を図35に示します。

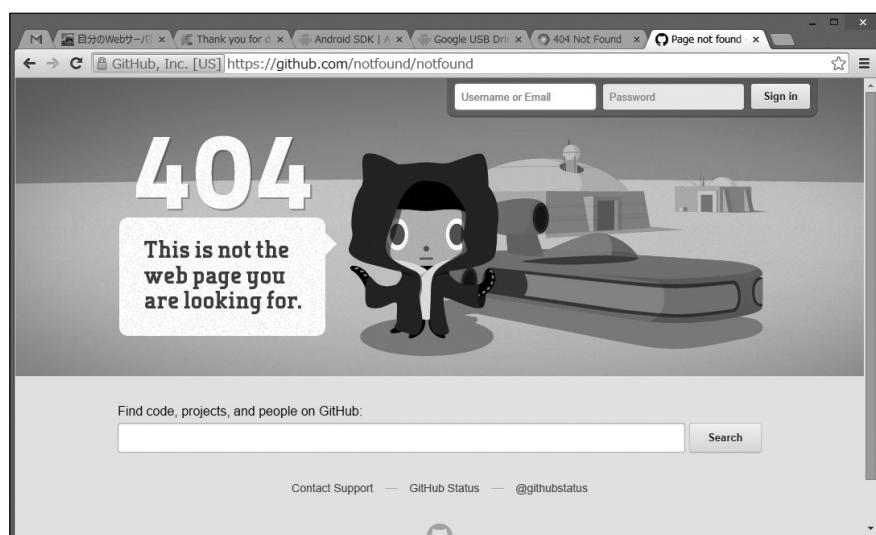


図35:GitHubの「404 Not Found」には絵が付いている

GAEでも、ステータスコードをPython実装上で指定したり、GitHubのように見た目の異なる404画面を表示させることもできますが、今回は割愛します。GAE Py

thonでのレスポンスを操作する際の詳細は、例えば「<https://cloud.google.com/appengine/docs/python/tools/webapp/redirects>」を参照してください。



18-1-13

世界に Hello World !

公開したサーバーは講義室内に限らず、インターネット上のユーザーから見ることができます。今回の演習では、開発段階から個人情報やプライバシーに関わる情報は含めないほうが良いでしょう。

「Google App Engine」の便利なところは、この時点ですでに、世界へWebアプリケーションを公開する準備がほぼできていることです。Googleの本番サーバーにアップロードし、世界にアプリケーションを公開してみましょう。

公開するには、Googleアカウントで開発者登録をした上で、Webアプリのための「アプリケーションID」を取得します。まずURL「<https://appengine.google.com/>」のページを開き、講義に用いるGoogleアカウントでログインします。その結果開くページで「Create Application」をクリックします(図36)。

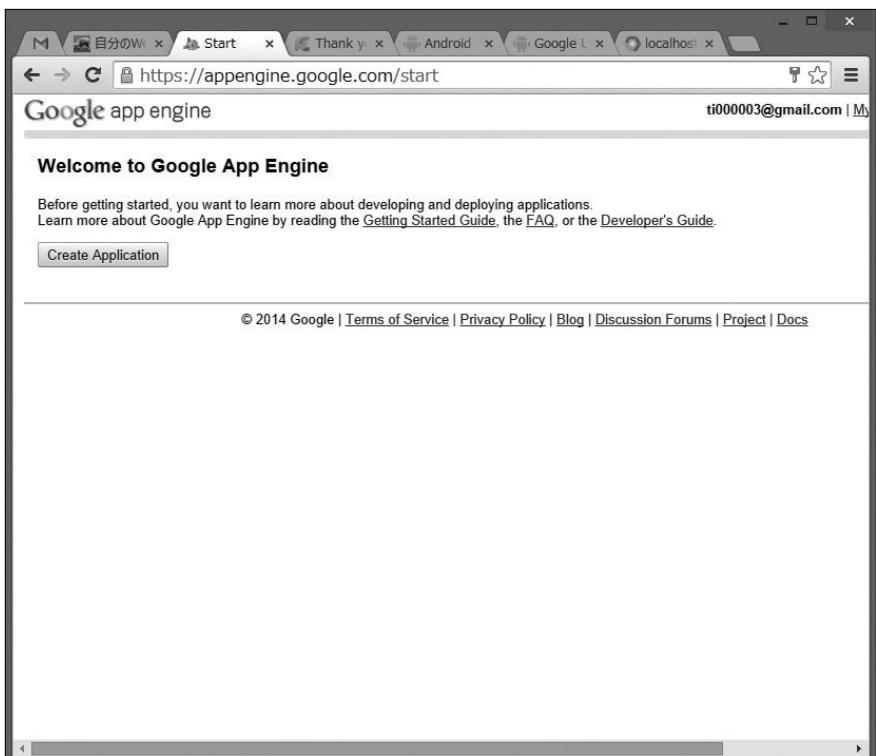


図36: ブラウザから「Create Application」を実行する

次に、図37のようなページが開くので、「Application Identifier」として「世界中で唯一」の名前を1つ設定します。他の人が利用している名前は選択できません。これはそのまま公開するサーバー名になるので、おかしな名前も避けたほうが良いでしょう。

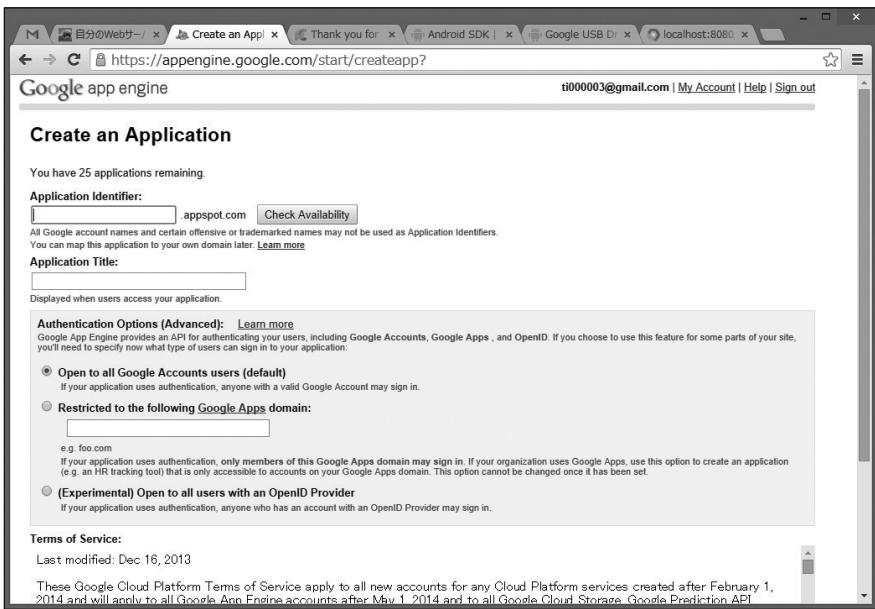


図37:他のユーザーが使っていないIDを決める。それがURLに含まれるので注意する

アプリケーションIDを考えたら、右側の「Check Availability」ボタンをクリックして、実際にそのアプリケーションIDを使用できることを確認します。「Google App Engine」の利用規約が表示されている場合は内容を確認し、「I accept these terms」をチェックした上で「Submit」を押します。そして最後に、最下段の「Create Application」ボタンを押します(図38)。

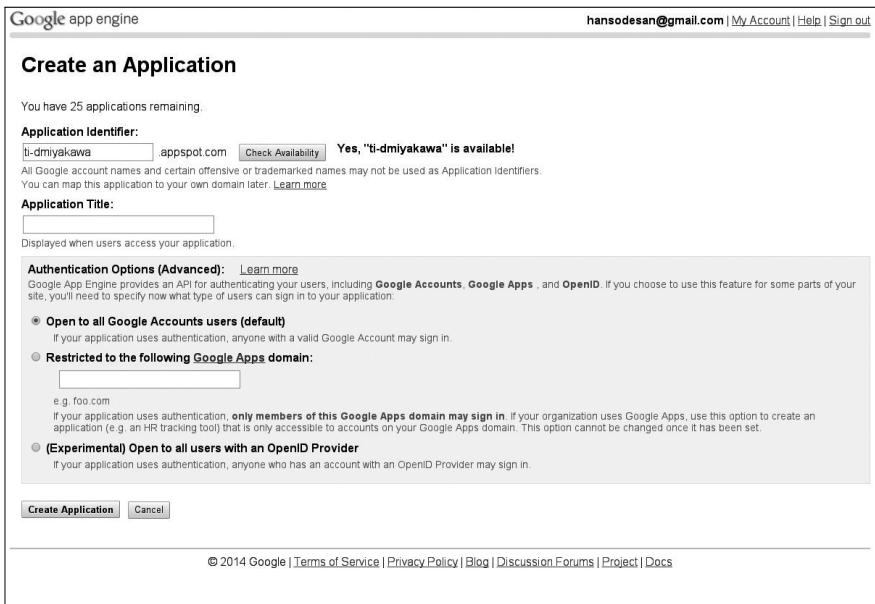


図38:「Create Application」ボタンをクリックする

アプリの作成に成功すると図39のようなページが表示されます。

The screenshot shows the Google App Engine application registration success page. At the top, it says "Application Registered Successfully". Below that, there's a note about the identifier being used. It also mentions the storage scheme (High Replication) and Google authentication. There are options to view the dashboard, upload code, or add administrators. At the bottom, there's a copyright notice for Google and links to Terms of Service, Privacy Policy, Blog, Discussion Forums, Project, and Docs.

図39: アプリケーションの登録が成功した

「Application Registered Successfully」と表示されたら、PyCharmで「app.yaml」を開き、最初の行の「application」の右側の文字列を自分が選択したものに変えます。

app.yaml(部分)

application: (ここを先ほど選択したIDに変更する)

「GAE Console」で「Deploy」ボタンを押すと、ユーザー名とパスワードを聞かれます。ここで、ユーザー名はGoogleアカウント名、パスワードはGoogleアカウントのパスワードを入力します。もし2段階認証プロセスを有効にしている場合は、アプリケーション固有のパスワードを準備してそれを使用します。

ログに「Deployment successful」と出力されたら成功です。

「[http://\(自分が生成したID\).appspot.com/](http://(自分が生成したID).appspot.com/)」へアクセスしてみましょう。せっかくですから、隣の人に自分のWebサーバーを見てもらってはいかがでしょうか。



18-1-14

ログ出力方法

PythonでもAndroidにおけるLogCatと似たようなログ機構を実装できます。以下に「logging」モジュールを用いた「Hello world!」の例を示します。

main.py: loggingを用いた例

```
import webapp2
import logging # この行を追加

class MainHandler(webapp2.RequestHandler):
    def get(self):
        logging.info('Hello world from logging!') # この行を追加
        self.response.write('Hello world!')
...
... 以下省略
```

今回のサーバー実装では「logging.info()」だけ使えば十分でしょう。そのモジュールを用いるため、Javaと同じように「import logging」という命令を予め入れておく必要があります。

Pythonのloggingについて詳細に学びたい人は、まず「<http://docs.python.jp/2/howto/logging.html#logging-basic-tutorial>」を読むことから始めるといいでしょう。

18-1-15 本番サーバーのダッシュボードとリソース割り当てについて

「GAE Launcher」には「Dashboard」というボタンもあります。このボタンを押すと、本番サーバーの管理者画面をブラウザに表示してくれます。ここから本番サーバーでのログ出力も見ることができます(図40)。

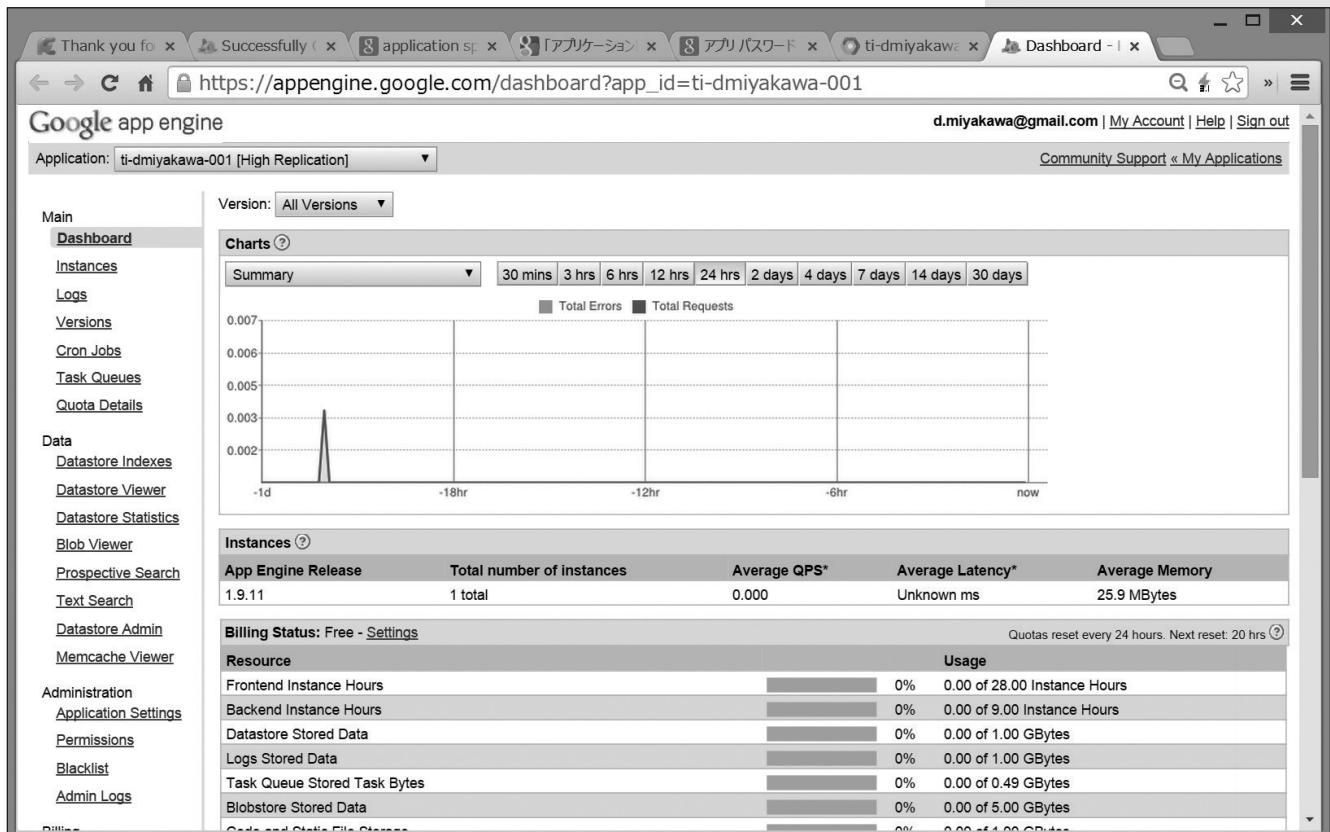


図40:GAEの本番サーバーのダッシュボード

「Google App Engine」は小規模なサーバーを起動する分には無料で利用できます。より正確には、それぞれのアプリに無料で利用できる一日あたりの割り当てるが決まっており、その範囲内のアクセスであれば料金はかかりません。割り当てとの消費量は「Quota Details」から見ることができます(図41)。

The screenshot shows the Google App Engine Quota Details page. On the left, there's a sidebar with links like Main Dashboard, Instances, Logs, Versions, Cron Jobs, Task Queues, Quota Details (which is selected), Datastore Indexes, Datastore Viewer, Datastore Statistics, Blob Viewer, Prospective Search, Text Search, Datastore Admin, Memcache Viewer, Application Settings, Permissions, Blacklist, and Admin Logs.

The main content area has a header: "The quota details for this application are grouped by API and are listed below. If your application exceeds 50% of any particular quota halfway through the day, it may exceed the quota before the day is over. To learn more about how quotas work, read [Understanding Quotas](#) and [Why is My App Over Quota?](#)". It also says "Quotas are reset every 24 hours. Next reset: 22 hours".

Requests

Resource	Daily Quota	Rate	
Requests	2	Okay	
Outgoing Bandwidth	0%	0.00 of 1.00 GBytes	Okay
Incoming Bandwidth	0%	0.00 of 1.00 GBytes	Okay
Secure Requests	0	Okay	
Secure Outgoing Bandwidth	0.00 GBytes	Okay	
Secure Incoming Bandwidth	0.00 GBytes	Okay	
Frontend Instance Hours	0%	0.05 of 28.00 Instance Hours	Okay
Backend Instance Hours	0%	0.00 of 9.00 Instance Hours	Okay

Storage

Datastore Write Operations	0%	0.00 of 0.05 Million Ops	Okay
Datastore Read Operations	0%	0.00 of 0.05 Million Ops	Okay
Datastore Small Operations	0%	0.00 of 0.05 Million Ops	Okay
Datastore API Calls	0	Okay	
Datastore Queries	0	Okay	
Blobstore API Calls	0	Okay	
Datastore Stored Data	0%	0.00 of 1.00 GBytes	Okay
Blobstore Stored Data	0%	0.00 of 5.00 GBytes	Okay
Data Sent to Datastore API	0.00 GBytes	Okay	
Data Received from Datastore API	0.00 GBytes	Okay	

図41:GAEの本番サーバーのQuota(割り当て)表示

第三者からの大量のアクセスがあったりした際に、この割り当てを使い尽くしてしまう可能性があります。その場合、すぐに課金が発生するわけではなく、ユーザーにはエラーが表示されます。明示的にクレジットカードを登録しない限り、課金されません。

今回の演習の範囲でこの問題を考える必要はないはずですが、仮にGAEで規模の大きなサービスを実装する場合、少なくとも以下の2点について対処する必要があります。

- ・ クレジットカードを登録し、アクセスに応じた課金を行えるようにする
- ・ アクセス負荷が小さくなるよう、GAEに適した形でアプリケーションの設計を考えなおす

電気代やインターネット接続料金を誰がどう支払うか、といった点を含めて、様々な料金体系のサービスがあるのです。

18-2 「シラバスアプリ」とWebサーバーの連携

著：宮川大輔

本節では、前節で実装したWebサーバーを拡張し、第10章で作った「シラバスアプリ」と連携できるようにします。Webブラウザから講義データをアップロードする機能も追加しながら、Webサーバーと通信する上でAndroidアプリ開発者が気をつけるべきことを学びましょう。



この節で学ぶこと

- ・HTTPのGETリクエストとPOSTリクエスト
- ・クロスサイト・スクリプティング(XSS)

この節で出てくるキーワード一覧

Content-Type

JSON

HTML

RFC

クロスサイト・スクリプティング(XSS)

テンプレートエンジン

jinja2

スタイルシート(CSS)

bootstrap

HTTPS



18-2-1

「シラバスサーバー」を考える

第10章「ユーティリティによる実践」でAndroidアプリを作成した際、Webサーバーに保存されたJSONデータを「Volley」というライブラリーを用いてダウンロードし、その内容を「ListView」に表示しました(図42)。この時に使用したデータ取得URLは、「https://dl.dropboxusercontent.com/u/1088314/tech_institute/2014/syllabus.json」でした。



図42:第10章で作成したシラバスアプリ

「Dropbox」
<https://www.dropbox.com/>

このURLは、講師が「Dropbox」というWebサービスを用いて公開している、静的な(アクセスの度に中身が変わったりしない)ファイルの場所を表しています。もし講義内容を変更したいとしたら、直接そのファイル自身を修正する必要があります。

本節では、この「シラバス」データを管理して公開する「シラバスサーバー」を実装することを考えましょう。具体的には、「Hello Worldサーバー」を拡張することで以下の機能を実現します。

- ・ JSON形式とHTML形式によって講義データを配布できるようにする
- ・ Webサーバーへブラウザから講義データを追加できるようにする
- ・ 最低限のセキュリティを確保する

サーバー側を実装する前にまず確認したいのは、「どのような方法でAndroidアプリとWebサーバーが通信するか」です。このルールがアプリとサーバー側で違つていると、通信はできません。どのような通信を行えば良いかを知るため、まず、上記のJSONデータをWebブラウザーから眺めてみましょう(図43)。

```
{"course": [
    {"No": "Chapter 1 ファーストステップ [全7回] 単元担当講師: 柴田文彦", "date": "", "day": "", "time": "", "chapter": "", "title": "", "teacher": "", "supporter-1": "", "supporter-2": "", "supporter-3": "", "supporter-4": "", "supporter-5": "", "detail": ""}, {"No": 1, "date": "2014-07-01T07:00:00.000Z", "day": "火", "time": "18:30-19:00", "chapter": "", "title": "開講式", "teacher": "", "supporter-1": "", "supporter-2": "", "supporter-3": "", "supporter-4": "", "supporter-5": "", "detail": ""}, {"No": "", "date": "", "day": "", "time": "19:00-21:00", "chapter": "1章", "title": "プログラミングとは何か", "teacher": "柴田文彦", "supporter-1": "", "supporter-2": "", "supporter-3": "", "supporter-4": "", "supporter-5": "", "detail": ""}, {"No": 2, "date": "2014-07-02T07:00:00.000Z", "day": "水", "time": "19:00-21:00", "chapter": "2章", "title": "コンピュータとスマホ、ストレージ、ネットワーク", "teacher": "嶋 是一", "supporter-1": "柴田", "supporter-2": "", "supporter-3": "", "supporter-4": "", "supporter-5": "", "detail": ""}, {"No": 3, "date": "2014-07-03T07:00:00.000Z", "day": "木", "time": "19:00-21:00", "chapter": "3章", "title": "プランニング (1)", "teacher": "嶋 是一", "supporter-1": "金澤創平", "supporter-2": "あくやん", "supporter-3": "中川茂樹", "supporter-4": "宮川大輔", "supporter-5": "", "detail": ""}, {"No": 4, "date": "2014-07-08T07:00:00.000Z", "day": "火", "time": "19:00-21:00", "chapter": "4章", "title": "開発環境セットアップ", "teacher": "柴田文彦", "supporter-1": "宮川大輔", "supporter-2": "下川敬弘", "supporter-3": "武藤繁夫", "supporter-4": "高橋憲一", "supporter-5": "", "detail": ""}, {"No": 5, "date": "2014-07-09T07:00:00.000Z", "day": "水", "time": "19:00-21:00", "chapter": "5章", "title": "JAVAって何だ? (1)", "teacher": "柴田文彦", "supporter-1": "宮川大輔", "supporter-2": "下川敬弘", "supporter-3": "齊藤弘太", "supporter-4": "", "supporter-5": "", "detail": ""}, {"No": 6, "date": "2014-07-10T07:00:00.000Z", "day": "木", "time": "19:00-21:00", "chapter": "5章", "title": "JAVAって何だ? (2)", "teacher": "柴田文彦", "supporter-1": "宮川大輔", "supporter-2": "下川敬弘", "supporter-3": "齊藤弘太", "supporter-4": "武藤繁夫", "supporter-5": "", "detail": ""}, {"No": 7, "date": "2014-07-11T07:00:00.000Z", "day": "火", "time": "19:00-21:00", "chapter": "5章", "title": "JAVAって何だ? (3)", "teacher": "柴田文彦", "supporter-1": "宮川大輔", "supporter-2": "下川敬弘", "supporter-3": "齊藤弘太", "supporter-4": "武藤繁夫", "supporter-5": "", "detail": ""}, {"No": 8, "date": "2014-07-16T07:00:00.000Z", "day": "水", "time": "19:00-21:00", "chapter": "6章", "title": "ためしてわかるAndroidのしくみ (1)", "teacher": "飯塚康至", "supporter-1": "高橋憲一", "supporter-2": "柴田", "supporter-3": "下川敬弘", "supporter-4": "武藤繁夫", "supporter-5": "日高正博", "detail": ""}, {"No": 9, "date": "2014-07-17T07:00:00.000Z", "day": "木", "time": "19:00-21:00", "chapter": "6章", "title": "ためしてわかるAndroidのしくみ (2)", "teacher": "飯塚康至", "supporter-1": "高橋憲一", "supporter-2": "金澤創平", "supporter-3": "下川敬弘", "supporter-4": "武藤繁夫", "supporter-5": "宮川大輔", "detail": ""}, {"No": 10, "date": "2014-07-22T07:00:00.000Z", "day": "火", "time": "19:00-21:00", "chapter": "7章", "title": "AndroidのAPIについて学ぶ (1)", "teacher": "小泉 勝志郎", "supporter-1": "高橋憲一", "supporter-2": "宮川大輔", "supporter-3": "柴田", "supporter-4": "武藤繁夫", "supporter-5": "大和田健一", "detail": ""}, {"No": 11, "date": "2014-07-27T07:00:00.000Z", "day": "水", "time": "19:00-21:00", "chapter": "7章", "title": "AndroidのAPIについて学ぶ (2)", "teacher": "小泉 勝志郎", "supporter-1": "高橋憲一", "supporter-2": "宮川大輔", "supporter-3": "柴田", "supporter-4": "武藤繁夫", "supporter-5": "大和田健一", "detail": ""}], "course": [{"No": "Chapter 2 Androidの仕組み [全6回] 単元担当講師: 小泉勝志郎", "date": "", "day": "", "time": "", "chapter": "", "title": "", "teacher": "", "supporter-1": "", "supporter-2": "", "supporter-3": "", "supporter-4": "", "supporter-5": "", "detail": ""}, {"No": 8, "date": "2014-07-16T07:00:00.000Z", "day": "水", "time": "19:00-21:00", "chapter": "6章", "title": "ためしてわかるAndroidのしくみ (1)", "teacher": "飯塚康至", "supporter-1": "高橋憲一", "supporter-2": "柴田", "supporter-3": "下川敬弘", "supporter-4": "武藤繁夫", "supporter-5": "日高正博", "detail": ""}, {"No": 9, "date": "2014-07-17T07:00:00.000Z", "day": "木", "time": "19:00-21:00", "chapter": "6章", "title": "ためしてわかるAndroidのしくみ (2)", "teacher": "飯塚康至", "supporter-1": "高橋憲一", "supporter-2": "金澤創平", "supporter-3": "下川敬弘", "supporter-4": "武藤繁夫", "supporter-5": "宮川大輔", "detail": ""}, {"No": 10, "date": "2014-07-22T07:00:00.000Z", "day": "火", "time": "19:00-21:00", "chapter": "7章", "title": "AndroidのAPIについて学ぶ (1)", "teacher": "小泉 勝志郎", "supporter-1": "高橋憲一", "supporter-2": "宮川大輔", "supporter-3": "柴田", "supporter-4": "武藤繁夫", "supporter-5": "大和田健一", "detail": ""}, {"No": 11, "date": "2014-07-27T07:00:00.000Z", "day": "水", "time": "19:00-21:00", "chapter": "7章", "title": "AndroidのAPIについて学ぶ (2)", "teacher": "小泉 勝志郎", "supporter-1": "高橋憲一", "supporter-2": "宮川大輔", "supporter-3": "柴田", "supporter-4": "武藤繁夫", "supporter-5": "大和田健一", "detail": ""}]]
```

図43: シラバスデータをWebブラウザーで見た結果

元のデータは人間が読めるようにフォーマットされていないため、大変読みづらいですが、本質的には次のリストに示すような構造をしています。

JSONフォーマットを整形したもの(抜粋)

```
"course": [
    ...
    {
        "No": 3,
        "date": "2014-07-03T07:00:00.000Z",
        "day": "木",
        "time": "19:00-21:00",
        "chapter": "3章",
        "title": "プランニング(1)",
        "teacher": "嶋 是一",
        "supporter-1": "金澤創平",
        "supporter-2": "あくやん",
        "supporter-3": "中川茂樹",
        "supporter-4": "宮川大輔",
        "supporter-5": "",
        "detail": ""
    },
    ...
    {
        "No": 65,
        "date": "2014-12-10T08:00:00.000Z",
        "day": "水",
        "time": "19:00-21:00",
        "chapter": "26章",
        "title": "レビューとプラッシュアップ(2)",
        "teacher": "嶋 是一",
        "supporter-1": "下川敬弘",
        "supporter-2": "武藤繁夫",
        "supporter-3": "中川茂樹",
        "supporter-4": "",
        "supporter-5": "",
        "detail": ""
    }
]
```

Eclipseを開いて、シラバスアプリの実装で「ListView」や「Intent」において使われていたデータ構造と比較してみてください。結果として、JSONで指定されたこれらのデータの中から、以下の4つの情報だけを使っていることが分かります。

- ・ **date**: 講義が行われる日付
- ・ **title**: その講義のタイトル
- ・ **teacher**: 講師の名前
- ・ **detail**: 講義の詳細 (サンプルのJSONファイルでは空)

それ以外のデータはまったく使っていないようなので、今回のWebサーバーでは、これら4種類のデータだけを扱うことにします。言い換えると、Webサーバーは、講義数分だけこの4種類のデータを繰り返し持つJSON形式のデータを、クライアントに返答すれば良いのです。



18-2-2

自分の Web サーバーから自動生成した JSON データを返す

「Hello World」サーバーの実装を変更し、講義情報が機械的に生成されるようにして、その結果をクライアント側に返送するWebサーバーにしてみます。

なお日本語を表示するためには、Pythonのルールとして「-*- coding: utf-8 -*-」というコメント行をファイル冒頭に含めておく必要があります。

helloworld.py

```
# -*- coding: utf-8 -*-

import webapp2

from datetime import datetime, timedelta
from json import dumps
import logging, pprint

class MainHandler(webapp2.RequestHandler):
    def get(self):
        self.response.headers['Content-Type'] = 'application/json; charset=UTF-8'
        courses = [] # 空のリスト
        for i in xrange(10): # 0から9までループ
            single_course = {'date': str(datetime.today() + timedelta(days=i)),
                             'title': 'たいとるその{}'.format(i),
                             'teacher': '第{}代目 高橋名人'.format(i),
                             'detail': ''}
            courses.append(single_course) # リストの末尾に追加している
        # 注意: 辞書のキー(左側)は単数形 course です。sは要りません
        data = dumps({'course': courses}, ensure_ascii=False)
        logging.info(pprint.pformat(data))
        self.response.write(data)

application = webapp2.WSGIApplication([
    ('/', MainHandler),
], debug=True)
```

ここでは、Python言語で生成したデータ構造を（「response.write()」関数で）JSONの文字列として返答しています。サーバーが返すデータがどのような形式のデータかを、ブラウザーやAndroidアプリは事前に理解する必要があるため、「Content-Type」という情報を指定しておく必要があります。あわせて日本語を表示するために文字コードも指定します。本講義の範囲では次のことが分かれれば十分です。サーバーから送られるデータがHTMLなのか、あるいはPDFや動画データといった別のデータなのか、ということです。それを区別するために、サーバーはユーザーに見えないところでクライアントとやり取りをしています。サーバーを実装する際には、これらを意識して送信しないと、ブラウザーの挙動が不適切になったりします。



18-2-3

生成された JSON データをブラウザーで確認する

ローカルサーバー(<http://localhost:8080>)から返されたJSON形式のデータを閲覧できることを、ブラウザーを用いて確認してください(図44)。

The screenshot shows a web browser window with the URL localhost:8080. The page content is a large block of JSON code. The JSON structure is as follows:

```
{"course": [{"teacher": "第0代目 高橋名人", "detail": "", "title": "たいとるその0", "date": "2014-09-25"}, {"teacher": "第1代目 高橋名人", "detail": "", "title": "たいとるその1", "date": "2014-09-26"}, {"teacher": "第2代目 高橋名人", "detail": "", "title": "たいとるその2", "date": "2014-09-27"}, {"teacher": "第3代目 高橋名人", "detail": "", "title": "たいとるその3", "date": "2014-09-28"}, {"teacher": "第4代目 高橋名人", "detail": "", "title": "たいとるその4", "date": "2014-09-29"}, {"teacher": "第5代目 高橋名人", "detail": "", "title": "たいとるその5", "date": "2014-09-30"}, {"teacher": "第6代目 高橋名人", "detail": "", "title": "たいとるその6", "date": "2014-10-01"}, {"teacher": "第7代目 高橋名人", "detail": "", "title": "たいとるその7", "date": "2014-10-02"}, {"teacher": "第8代目 高橋名人", "detail": "", "title": "たいとるその8", "date": "2014-10-03"}, {"teacher": "第9代目 高橋名人", "detail": "", "title": "たいとるその9", "date": "2014-10-04"}]}
```

図44:JSONデータをブラウザーで表示して見る

問題無さそうであれば、作成したサーバーアプリケーションを再び本番サーバーにアップロードします。本番サーバーのURLへブラウザーでアクセスし、やはり同様のJSONデータを取得できることを確認します。ここまでで、世界に公開された状態で講義表データを返すWebサーバーが完成しました！



18-2-4

シラバスアプリにサーバーの JSON データを読み込ませる

さっそくシラバスアプリでこのデータを表示させてみましょう。ここではAndroidアプリを編集するので、Pythonプログラムを書くPyCharmではなく、Eclipseを使用して作業を行います。混乱しませんように。

もしEclipse上にシラバスアプリが準備ができていない場合には、先にそちらを完成させるか、GitHubから最新のプロジェクトをダウンロードし、Eclipseにインポートして準備してください。18-3-2で、シラバスアプリをインポートする方法を説明しています。

まずシラバスアプリの「`MainActivity.java`」の「`syllabusUrl`」を自分の本番サーバーのURLに変更します。

MainActivity.java(部分)

```
private static final String syllabusUrl = "http://(自分が指定したID).appspot.com"
```

この変更を行った上で、これまでと同様にAndroidアプリをビルド、実行します。もしサーバーとAndroidアプリの通信がうまくいけば、サーバーが自動生成した講義が配信されるようになるはずです(図45)。



図45:シラバスアプリにサーバーが自動生成した講義名が表示された

シラバスアプリ上で、いつまでも「読み込中」を示すSpinnerが表示され続ける場合には、サーバーのログ画面とAndroidアプリのLogCatをよく見て問題を解決してください。例えば、返答するJSONで「course」となっているべきところが「courses」(sが余計)となっていた場合には、サーバーではエラーは出ず、LogCatで問題が報告されているはずです。

WebサーバーとAndroidアプリを交互に開発する際には、原因がWebサーバーにあるのか、Androidアプリにあるのかを見極める必要があります。サーバとブラウザ間の通信が成功していても失敗していても、Androidアプリの画面を見るだけでは、具体的にどのようなやり取りをしているのかは分かりません。AndroidアプリとWebサーバーをセットで開発している際、実際にクライアントとサーバーが何を通信しているのかを詳しく調べたいことがあります。以下に、chromeの開発者コンソールを用いた方法を簡単に紹介します。



18-2-5 Chrome の開発者コンソールを開いて通信の詳細を覗いてみる

現在はブラウザ自身に豊富な機能があるため、Webサーバーとクライアント間の通信を調べるには、まずこの機能を使うことが多いです。Chromeでは「デベロッパーツール」と呼ばれています。デベロッパーツールを起動するには、JSONデータを表示したChromeのウインドウ内で右ボタンクリックして、表示されるメニューから「要素を検証」を選択してください(図46)。すると、Chromeの下方にデベロッパーツールが表示されます。



図46:Chromeのデベロッパーツール

試しに「Network」タブを選択し、その状態でWebページを再読み込みしてみてください。すると、図47のように表示されるはずです。今回はただJSONデータを取得するだけなので、表示が1行増えるだけでしょう。

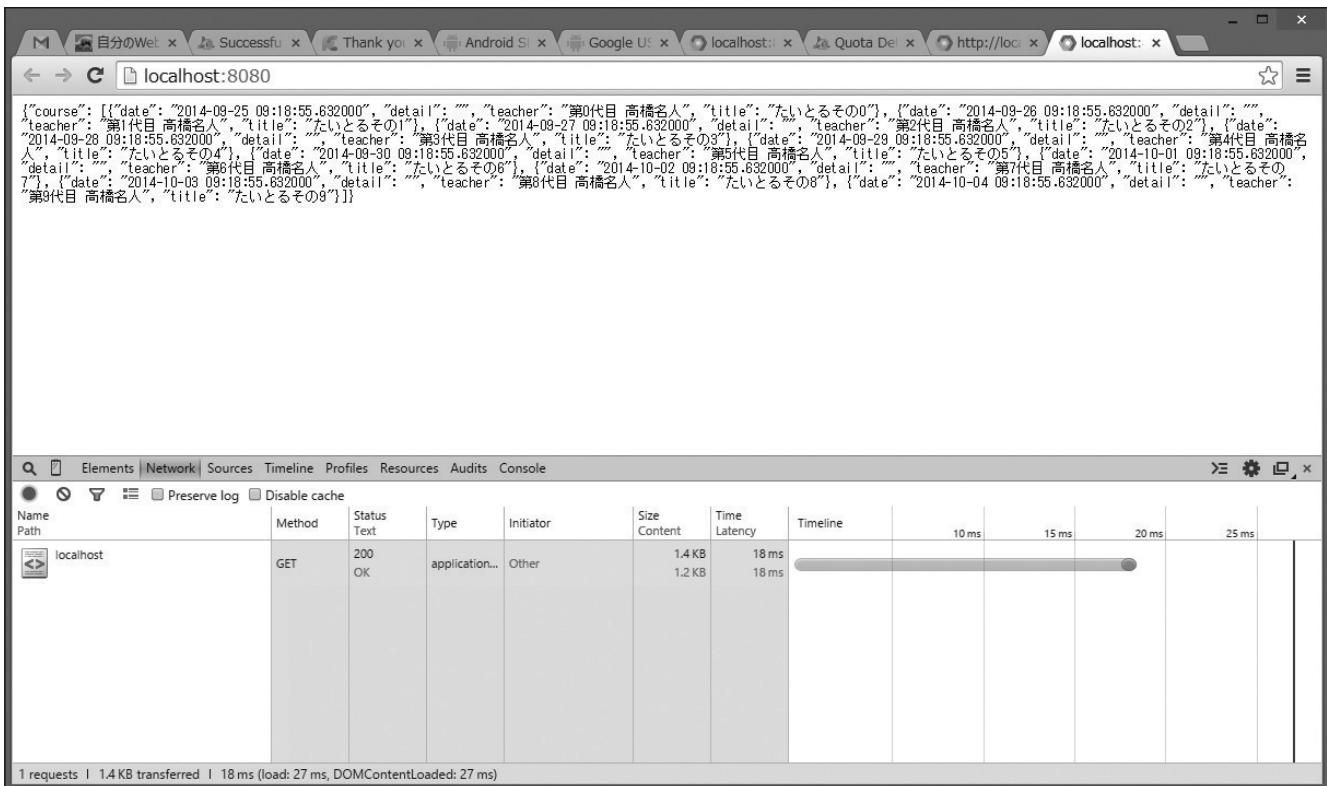


図47: デベロッパーツールで通信の内容を確認する

この例では、ブラウザーがGETリクエストをサーバーに送信し、サーバーから約18ミリ秒でステータスコード200とデータが返ってきたことが分かります。他のWebページで試してみると、さらに面白いでしょう。Webの1ページを表示するためにどれだけのGETリクエストが行われているかが分かるはずです。

Webサーバーとクライアント間をまたぐサービスを実装する場合、サーバー側のログとクライアントの画面表示のみならず、ネットワーク間の通信がどのように行われたかも把握する必要がしばしばあります。そういった場合、ブラウザー付属のツールを覚えておくととても便利です。「Chrome デベロッパーツール」などと検索すると多くの情報が得られるはずですので、デベロッパーツールについて詳細に学びたい方は調べてみてください。

ブラウザーの開発者向けツールよりも更に込み入った細かい情報を調べたい時があります。こういったときには、第14章「ネットワークプログラミング」で利用した「Putty」のような「生」のデータを扱えるツールが有用なことがあります。また、開発しているAndroidアプリの実際の通信をのぞき見することでデバッグすることもあります。

18-2-6 講義表を保存・変更できるようにする

ここまでで、Webサーバー上で生成したJSONデータをAndroidアプリに与えることに成功しました。しかし、このデータはPythonが勝手に生成したもので、本当に価値のあるデータとは言えません。

このWebサーバーをさらに発展させて、人の手で講義データを追加できるようにしてみましょう。本節では、以下の3種類のページを用意することにします。講義データを編集したり削除したりする機能の実装は、演習問題とします。

- ・講義一覧をHTMLで表示するトップページ
- ・JSON形式で講義を表示するページ
- ・講義の新規登録ページ

講義データを人の手で追加できるWebサーバーを実現するには、GAE上でデータを保存できるようにする必要があります。すでに議論したJSONデータの中にある4種類の情報をひとまとめにして、そのまとまりを複数保存できるようにする必要があります。

- ・**date**:講義が行われる日付
- ・**title**:その講義のタイトル
- ・**teacher**:講師の名前
- ・**detail**:講義の詳細（サンプルのJSONファイルでは空）

この4つのデータを、言ってみれば「1つのオブジェクト」のようにして保存したいわけです。まず、このデータ構造をサーバーに実装することにしましょう。今回はGAEの「Datastore」という仕組みを利用します。「helloworld.py」に次の実装を追加してください。

```
helloworld.pyへの追加

# (その他のimport文)

from google.appengine.ext import ndb

def course_list_key():
    return ndb.Key('CourseList', 'default_course_list')

class Course(ndb.Model):
    date = ndb.DateTimeProperty()
    title = ndb.StringProperty()
    teacher = ndb.StringProperty()
    detail = ndb.StringProperty()

# (MainHandlerの実装が続く)
```

保存する形式にはプログラミング言語と似たデータ型があります。GAEの「Datastore」で利用できるデータ型は「<https://developers.google.com/appengine/docs/python/ndb/properties>」に掲載されています。今回は「日付」に対応するデータと文字列データさえあれば事足ります。

「course_list_key()」関数は、GAE特有のおまじないとでも考えてください。この関数自体はすぐ後で使います。

GAEではここでさらに、データを順序立てて検索するために「インデックス」と呼ばれるデータを用意する必要があります。詳細は省略しますが、今回は「日付」で

ソートする前提で、「index.yaml」に次のように書いておきます。

index.yaml(app.yamlではないので注意)

```
indexes:
- kind: Course
  ancestor: yes
  properties:
  - name: date
```

この時点では、まだ保存するデータ構造を定義しているだけで、そのデータを表示するためには使っていません。さっそくJSONデータの一部として使ってみましょう。

helloworld.pyへの追加

```
# (import文やCourseクラス)

class MainHandler(webapp2.RequestHandler):
    def get(self):
        course_model_list = Course.query(ancestor=course_list_key()).order(Course.date)
        courses = [] # 空のリスト
        for course_model in course_model_list:
            single_course = {'date': str(course_model.date),
                            'title': course_model.title,
                            'teacher': course_model.teacher,
                            'detail': course_model.detail}
            courses.append(single_course)
        self.response.headers['Content-Type'] = 'application/json; charset=UTF-8'
        self.response.write(dumps({'course': courses}, ensure_ascii=False))

application = webapp2.WSGIApplication([
    ('/', MainHandler),
], debug=True)
```

まずローカルサーバーを起動してブラウザーで結果を確認します。ここではまだ講義データを一切登録していないので、JSONは以下のような「空」のコース一覧を表示します。

空の結果を持つJSON

```
{"course": []}
```

なおGAEの本番サーバーの仕様により、本番サーバーへ「index.yaml」を最初にアップロードした後、しばらくの間は、図48のように「NeedIndexError: The index for this query is not ready to serve. ...」（検索に使うインデックスが準備できていません）というエラーが表示されます。これが発生するのは、GAEでデータを取得する際にインデックス（索引）が必要なのに、それをGAE自身が用意するのに少し時間がかかるためです。だいたい数分程度待てばエラーはなくなります。

Internal Server Error

The server has either erred or is incapable of performing the requested operation.

```
Traceback (most recent call last):
  File "/base/data/home/runtimes/python27/python27_lib/versions/third_party/webapp2-2.5.2/webapp2.py", line 1535, in __call__
    rv = self.handle_exception(request, response, e)
  File "/base/data/home/runtimes/python27/python27_lib/versions/third_party/webapp2-2.5.2/webapp2.py", line 1529, in __call__
    rv = self.router.dispatch(request, response)
  File "/base/data/home/runtimes/python27/python27_lib/versions/third_party/webapp2-2.5.2/webapp2.py", line 1278, in default_dispatcher
    return route.handler_adapter(request, response)
  File "/base/data/home/runtimes/python27/python27_lib/versions/third_party/webapp2-2.5.2/webapp2.py", line 1102, in __call__
    return handler.dispatch()
  File "/base/data/home/runtimes/python27/python27_lib/versions/third_party/webapp2-2.5.2/webapp2.py", line 572, in dispatch
    return self.handle_exception(e, self.app.debug)
  File "/base/data/home/runtimes/python27/python27_lib/versions/third_party/webapp2-2.5.2/webapp2.py", line 570, in dispatch
    return method(*args, **kwargs)
  File "/base/data/home/apps/s~model-shelter-709/1.378819028227840532/helloworld.py", line 33, in get
    course_count = course_model_list.count()
  File "/base/data/home/runtimes/python27/python27_lib/versions/1/google/appengine/ext/ndb/utils.py", line 142, in positional_wrapper
    return wrapped(*args, **kwds)
  File "/base/data/home/runtimes/python27/python27_lib/versions/1/google/appengine/ext/ndb/query.py", line 1252, in count
    return self.count_async(limit, **q_options).get_result()
  File "/base/data/home/runtimes/python27/python27_lib/versions/1/google/appengine/ext/ndb/tasklets.py", line 325, in get_result
    self.check_success()
  File "/base/data/home/runtimes/python27/python27_lib/versions/1/google/appengine/ext/ndb/tasklets.py", line 368, in _help_tasklet_along
    value = gen.throw(exc.__class__, exc, tb)
  File "/base/data/home/runtimes/python27/python27_lib/versions/1/google/appengine/ext/ndb/query.py", line 1295, in _count_async
    batch = yield rpc
  File "/base/data/home/runtimes/python27/python27_lib/versions/1/google/appengine/ext/ndb/tasklets.py", line 454, in _on_rpc_completion
    result = rpc.get_result()
  File "/base/data/home/runtimes/python27/python27_lib/versions/1/google/appengine/api/apiproxy_stub_map.py", line 612, in get_result
    return self._get_result_hook(self)
  File "/base/data/home/runtimes/python27/python27_lib/versions/1/google/appengine/datastore/datastore_query.py", line 2697, in __query_result_hook
    yaml_index=yaml, xml_index=xml)
NeedIndexError: The index for this query is not ready to serve. See the Datastore Indexes page in the Admin Console.
The suggested index for this query is:
- kind: Course
  ancestor: yes
  properties:
  - name: date
    direction: desc
```

図48:本番サーバーで「NeedIndexError」が発生している

なお、似たようなエラーとして「NeedIndexError: no matching index found.」(一致するインデックスがありません)というメッセージが出ることがあります。この場合はPythonプログラムで指定しているソートの方法と「index.yaml」で生成しているインデックスが一致していないという意味です。つまり本番サーバーにアップロードしたプログラムか、「index.yaml」に何か間違いがあります。

空の講義データでは意味がないので、講義データの作成画面が欲しいところですが、その前にいったんHTMLで講義データを表示する実装もしておきましょう。

URLのパスの後に「?」をつけると、その後に「クエリ文字列」(Query String)という追加のデータを送ることができます。例えば(2014年9月現在)「<https://www.google.co.jp/search?q=html>」というURLをWebブラウザに入力すると「HTML」をGoogle検索することができますが、このとき「q=html」がクエリ文字列です。クエリ文字列は「=」と「&」を駆使することでJavaやPythonのマップのようなデータ構造を持つことができます。Google検索の場合は「q」というキーが「html」という値を持っていると考えることができるでしょう。

このクエリ文字列の仕組みを用いて、シラバスサーバーのトップページの挙動を

以下のように変更します。

- ・ クエリ文字列に指定がなければHTMLを返す(例「`http://localhost:8080/`」)
- ・ 「`output=json`」というクエリ文字列をつけたらJSONを返す(例「`http://localhost:8080/?output=json`」)

GAEでは「`self.request.get('output')`」といった表現で、クエリ文字列に与えられたキーと値の組み合わせを簡単に取得できます。この表現と「if」分岐を組み合わせてトップページでのHTML表示とJSON表示を分岐して表示する結果を次に示します。なお、既に実装したJSON側の各行のインデントが1段下がっている点にも注意してください。Pythonではインデントがブロックの範囲を表します。

helloworld.pyへの追加(HTMLページ形式も表示できるようにする)

```
# -*- coding: utf-8 -*-
from google.appengine.ext import ndb
import webapp2

from datetime import datetime, timedelta
from json import dumps

def course_list_key():
    return ndb.Key('CourseList', 'default_course_list')

class Course(ndb.Model):
    date = ndb.DateTimeProperty()
    title = ndb.StringProperty()
    teacher = ndb.StringProperty()
    detail = ndb.StringProperty()

class MainHandler(webapp2.RequestHandler):
    def get(self):
        course_model_list = Course.query(ancestor=course_list_key()).order(Course.date)
        output = self.request.get('output')
        # outputが「空」、もしくはhtmlなら
        if (not output) or (output == 'html'):
            lst = ['<html><body>']
            lst.append('<h1>講義数合計: {}</h1>')
            .format(course_model_list.count())
            lst.append('<ul>')
            for course_model in course_model_list:
                lst.append('<li>{}, {}, {}</li>')
                .format(course_model.title,
                        course_model.teacher,
                        course_model.detail))
            lst.append('</ul>')
            lst.append('<a href="/create">Create</a>')
            lst.append('</body></html>')
            self.response.headers['Content-Type'] = 'text/html; charset=UTF-8'
            for line in lst: self.response.write(line)
```

```

else:
    courses = [] # 空のリスト
    for course_model in course_model_list:
        single_course = {'date': str(course_model.date),
                         'title': course_model.title,
                         'teacher': course_model.teacher,
                         'detail': course_model.detail}
        courses.append(single_course)
    self.response.headers['Content-Type'] = 'application/json; charset=UTF-8'
    self.response.write(dumps({'course': courses}, ensure_ascii=False))

app = webapp2.WSGIApplication([
    ('/', MainHandler),
], debug=True)

```

なお、この「仕様変更」によって、講義表のJSON形式データを受け取るURLが変わりました。これ以降、Androidアプリ側で結果を取得するURLの末尾には「?output=json」をつけておく必要があります。

MainActivity.java(変更)

```
private static final String syllabusUrl = "http://(ID).appspot.com/?output=json";
```



18-2-7 講義データを追加するための画面を作る

講義データを作成するHTMLフォームを作り、シラバスサーバーを完成させましょう。ここで行うべきことが大きく分けて2つあります。

- ・ フォーム画面そのものを作る
- ・ フォーム画面から送信されたデータを受け取り、サーバーに登録する部分を作る

まずフォーム画面そのものを作ります。単純化のため、今はHTMLをPythonの複数行文字列としてプログラムに埋め込みます。18-2-9でテンプレートを用いたより適切な方法を紹介します。

helloworld.py(続き:これだけではデータを送信した後にエラーが発生する)

```
# (MainHandlerまで)

form_html = '''\
<html><body>
<h1>コース作成画面</h1>
<form action="/create" method="post">
講義日: <input type="date" name="date"><br>
講義名: <input type="text" name="title" size="40"><br>
講師: <input type="text" name="teacher" size="20"><br>
詳細: <textarea name="detail" rows="4" cols="50"></textarea><br>
<input type="submit" value="送信">
</form>
</body></html>
'''

class CreateCourse(webapp2.RequestHandler):
    def get(self):
        self.response.headers['Content-Type'] = 'text/html; charset=UTF-8'
        self.response.write(form_html)

# CreateCourseの行が増えている点に注意!
application = webapp2.WSGIApplication([
    ("/create", CreateCourse),
    ("/.*", MainHandler),
```

データを追加するフォーム画面を表示するため「CreateCourse」というクラスを新たに追加しました。このクラスは「`http://localhost:8080/create`」に対応するURLへユーザーがアクセスした時に、所定のHTMLを表示します。「/create」というパスと「CreateCourse」の対応は、「`webapp2.WSGIApplication()`」が受け取るリストに記述されています。

今回の実装では、HTTPのGETリクエストではなく、POSTリクエストを用いて、Webサーバーにデータを送信するよう、HTMLフォームに指定しています(上のリストの「`method="post"`」の部分に注意)。通常のWebサーバーでは、ユーザーがデータを閲覧する際にはGETリクエストのみ、ユーザーがデータを追加・更新する際にはPOSTリクエストのみが用いられるように実装していれば大丈夫です。なぜGETリクエストではなくPOSTリクエストを使うかについての詳細は、コラムを参照してください。

これでフォームがブラウザに表示されるようになりましたが、そのフォームから送られたデータを受け取るための実装はまだ作っていません。このまでは「405 Method Not Allowed」というエラーが出ます(図49)。



図49:まだPOSTの受け口を作っていないために表示されるエラー

というわけで、CreateCourseクラスにさらPOSTリクエストを受け取るメソッドpost()を追加します。

```
helloworld.py(続き:CreateCourseの全実装)

class CreateCourse(webapp2.RequestHandler):
    def get(self):
        self.response.headers['Content-Type'] = 'text/html; charset=UTF-8'
        self.response.write(form_html)

    def post(self):
        date_str = self.request.get('date')
        if date_str:
            date = datetime.strptime(date_str, '%Y-%m-%d')
        else:
            date = datetime.today()
        title = self.request.get('title')
        teacher = self.request.get('teacher')
        detail = self.request.get('detail')
        logging.info('Saving "{}", "{}", "{}", "{}"'
                    .format(date, title, teacher, detail))
        new_course = Course(parent=course_list_key(),
                             date=date,
                             title=title,
                             teacher=teacher,
                             detail=detail)
        new_course.put()
        self.redirect('/')
```

ここまでで、最初に提示した3つシナリオをすべて実装し終えました。以下を実行して、実装に誤りがないことを確認してください。

- ・「<http://localhost:8080/?output=json>」にアクセスして、JSON形式の出力が出ることを確認する
- ・AndroidアプリでURLの末尾に「/?output=json」がついていることを再確認する
- ・本番サーバーへアップロードする
- ・ローカルサーバーと本番サーバーはデータを共有しない
- ・本番サーバーで講義データを登録する
- ・Androidアプリを動かして、講義データが追加されたことを確認する

以上で講義情報を追加できるだけの「シラバスサービス」ができました。実際に使うのであれば編集と削除機能が必要なところですが、これについては演習課題としましょう。まだこのままでは機能の他に気がかりな点がいくつかあります。次にそれを見ていきましょう。

check!**なぜ更新はPOSTリクエストで行うのか**

クエリ文字列を用いればGETリクエストだけで更新も行えるように見えるため、実際そのような実装にしてしまいたくなることがあります。しかし、これは一般的には良い実装とは考えられていません。GETでサーバーデータの更新をさせるのは、HTTPの仕様で適切でないとされています。それに伴って、発見と修正が非常に面倒なバグに悩まされることすらあります。

HTTPの仕様はRFC(Request For Comments)という公開されたドキュメント群の一部として公開されています。RFCのドキュメントにはそれぞれに番号がついているのですが、その中で「RFC 7231」(<http://tools.ietf.org/html/rfc7231#section-4>)と呼ばれるドキュメントに、GETとPOST、そしてその他のリクエストメソッドの仕様が決められています。

「RFC 7231」では、要約すると「GETリクエストはサーバーの状態を変更せず(安全)、何度も行なっても結果が同じである(べき等)、そしてその結果はキャッシュ可能であるべきである」と定められています。言い方を変えると、GETリクエストでサーバーの

データを書き換えるのは少なくとも行儀の良い動作ではない、ということです。

例えば、インターネット上で通信する際、GETリクエストのこの性質を上手く用いて、結果をキャッシュするコンピューターやソフトが途中経路に存在しても、文句を言なことはできません。結果をキャッシュされると、Webサーバーに期待した更新リクエスト自体が送られてこないという事態が発生します。POSTではこのような挙動は仕様上許されませんので、その不適切な挙動を行うコンピュータについて少なくとも文句を言うことができます。

GETリクエストではURLにデータが埋め込まれます。そのため、URLの最大長の制約を受けます。またWebブラウザのキャッシュにデータそのものが残ってしまいます。それに対して、POSTリクエストはもともとデータを送るためのものなので、GETリクエストに関するこのような制約はありません。ただし、データ送信以外にPOSTを用いるのは不適切ですので、GETとPOSTを使い分けるのが適切、という結論になります。

**18-2-8 セキュリティ**

ここまでで、とりあえずシラバスサーバーができ上がりましたが、実際のサービスとしてこのWebサーバーを公開したら、他人が利用した時点で即座にトラブルに見舞われるはずです。実装が足りないというだけでなく、セキュリティの面で課題が多いからです。以降では、すぐに判明するセキュリティ課題の一部を検討してみましょう。

ログイン機能とアクセス制御を実装するには

現在は誰が来てもこの講義表を見るすることができます。これはまだ問題ありませんが、さらに誰でも講義を追加することができます。こちらは大問題です。

GAEではGoogleアカウントによるログイン処理を実装するのは簡単です。試しに、「Hello world!」の代わりにユーザーの名前を表示する方法を以下のリストに示します。これは、いったんシラバスアプリを離れて、ただの「Hello World」サーバーにログイン機能を追加した例です。

`helloworld.py`(ログインを実装する例)

```
# 「Hello world」サーバの実装にログイン機能を追加
def get(self):
    user = users.get_current_user()
    if user:
        self.response.headers['Content-Type'] = 'text/plain'
        self.response.write('Hello ' + user.nickname())
    else:
        self.redirect(users.create_login_url(self.request.uri))
```

この例では、ローカルサーバーでもユーザー名(メールアドレス)とパスワードを聞かれます。こちらはどのようなメールアドレスであっても動作しますが、本番サーバーでは、実際のGoogleアカウントでログインする必要があります(図50)。

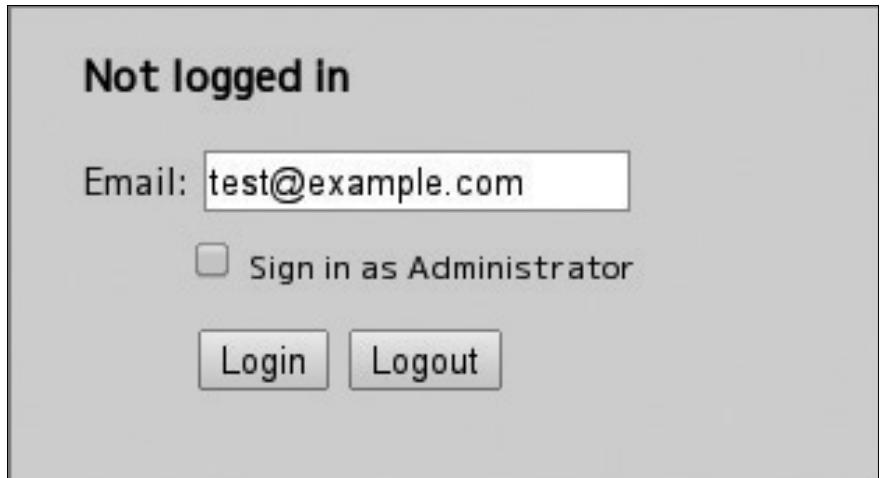


図50:ローカルサーバーの場合のログイン画面

講義表を公開するまでは良いとして、講義表を編集するページだけは、特定の人だけが利用できるようにするのが適切でしょう。例えば「user.user_id()」を用いて、自分がログインした際の「User ID」を把握し、それ以外のユーザーに対してはアクセスを拒否するといった実装が考えられます。実際に実装する部分は追加課題とします。

ログイン関連APIの詳細は以下のWebページに説明があります。

- ・ <https://developers.google.com/appengine/docs/python/users/>
- ・ <https://developers.google.com/appengine/docs/python/users/userclass>

クロスサイト・スクリプティング脆弱性とセキュアプログラミング

シラバスサービスが受け入れる講師名などの文字列に、HTMLタグを入力したらどうなるでしょうか。試しに「<script>alert("Hello");</script>」を講義タイトルとして入力して、保存してみましょう(図51)。

コース作成画面

講義日:

講義名:

講師:

詳細:

図51:「講義名」欄にHTMLタグを含む文字列を入力する

すると、その文字列が表示される代わりに、JavaScriptプログラムが実行され、アラートダイアログが表示されてしまいます(図52)。明らかにシラバスサービスが意図していない挙動です。



図52:クロスサイト・スクリプティング攻撃が成功している

これはいわゆる「クロスサイト・スクリプティング(XSS)」と呼ばれる立派なWebサイト攻撃方法の1つです。サービスが攻撃を受けるところまでは仕方ありませんが、その攻撃が成功してしまうのはいけません。

今回は一見すれば無害ですが、大手のWebアプリケーションでこのようなバグがあると、しばしば、パスワード漏洩を始めとする様々なセキュリティ問題の温床になります。一般にこのようなセキュリティに関するバグを「脆弱性」と言います。言い方を変えると、これまで実装したシラバスサービスには「クロスサイト・スクリプティング脆弱性」が存在しています。

今回のように単純に迷惑なプログラムを埋め込まれる、というだけでも十分問題ですが、場合によっては情報漏洩のきっかけになることもあります。埋め込んだプログラムがそうとは知らない第三者によって実行された際、そのプログラムはその人のログイン情報を取得できる可能性があります。それが仮に管理者アカウントだとすれば、ア

カウントの乗っ取りすら可能かもしれません。

ユーザーから提供されるデータをHTMLに埋め込む場合、最低でも以下の表に挙げる5種類の文字を適切に書き換える必要があると言われています。HTMLで特別な解釈をされるからです

文字	書き換え後の文字列
&	「&」(英語のampersandから)
"	「"」(英語のquoteから)
'	「'」(英語のapostropheから)
>	「>」(英語の greater than から)
<	「<」(英語の less than から)

表2:HTMLに埋め込む際に書き替えるべき文字

追加演習で登場する「テンプレート」の仕組みを用いる場合は、この処理は必要ありません。テンプレートエンジンが自動的にエスケープを行うからです。

この書き換えを自分で実装しても良いのですが、より安全なのは、他の人によって実装、検証された方法を使うことです。今回はPython自体に付属している「xml.sax.saxutils.escape()」関数を使えば良いでしょう。一般にこのような手続きを「エスケープ処理」と呼びます

以下にこの関数を利用した例を挙げます。

```
helloworld.py(追加:escape()を使う)

# 以下の行を追加するのを忘れないように
from xml.sax.saxutils import escape

# (略)

for course_model in course_model_list:
    lst.append('<li>{}, {}, {}</li>'
               .format(escape(course_model.title),
                       escape(course_model.teacher),
                       escape(course_model.detail)))
# (略)
```

Webアプリを開発する場合には、今回出会った「クロスサイト・スクリプティング脆弱性」以外にも、いくつか「定番」の脆弱性を作りこみがちです。IPA(情報処理推進機構)が「安全なウェブサイトの作り方」というドキュメントを無料公開しています(「<https://www.ipa.go.jp/security/vuln/websecurity.html>」)。本格的にWebアプリを作りこむ場合には、事前に目を通しておくことをおすすめします。

https

URLには「http」で始まるケースと「https」で始まるケースがあります。

「http」の場合、TCP接続して相手と通信する際、ネットワーク上の途中経路

に存在する全てのコンピューターから、通信しているデータを盗み見ることができます。ログイン機能を用いて適切なアクセス制御を実現していても、インターネット上に秘密のデータがそのまま平文(ひらぶん)で送受信されている状態なので、言ってみれば盗聴し放題です。

一方、「https」ではサーバーとクライアント間で通信を開始する際に、その2者しかわからない形で暗号化するので、データを盗み見られる可能性が小さくなります。特に送受信するデータを他人から見られたくない場合は、「https」のURLを用いてやりとりを検討してください。

今回用いるGAEの本番サーバーは、初めからhttpとhttpsの両方をサポートしています。これまでの本番サーバーのURLの「http」を「https」に変更すれば、ブラウザなどによる表示はそのままで、暗号化の恩恵を得られるでしょう。

ただし、多くの場合はhttpとhttpsを即座に切り替えられるわけではありません。事実、GAEで開発中に使う「`http://localhost:8080`」というURLでは、httpsを利用することはできません。その代わり、このURLはPC内からしか見ることをできなくすることによってセキュリティを確保しています。

盗み見られても全く害がない場合には、httpのURLを使っても問題がないこともあります。近年ではそのような分野は少なくなっています。皆に公開されているニュース等であれば良いのですが、例えば企業秘密をやりとりするといった場合には注意が必要です。

なお、一口にhttpsと言っても「どのように」暗号化通信を行うかについては、実は細かな違いがたくさんあります。Webサーバーを真面目に設計・開発・運用する際には、そういった部分も併せて考える必要があります。

不適切な方法で暗号化通信を行っている場合には、httpsの意味がないこともあります。

本文執筆時には「Heartbleed」や「POODLE」と呼ばれる脆弱性が相次いで報告されています。いずれもhttpsの暗号化通信を無効化される可能性のある厄介な問題で、適切なソフトウェア更新と設定が必要です。



18-2-9 HTMLテンプレートとスタイルシートを用いたWebサーバーの改善例

これまで、Webブラウザに表示させるHTMLをPythonに直接書き込んでいました。Androidアプリで言えば、XML形式のレイアウトファイルを書かずに、Javaを用いてActivityに直接レイアウトについての指示を書き込んでいるような状態です。また、デザインについての情報に至っては一切含めていませんので、普段見るWebサイトと比べると、だいぶ簡素です。

しかし、大きなWebアプリを作るのであればこの方法は色々と不都合です。Androidアプリの開発に描いて「UI基礎編」で既に述べられている通り、デザインとコードは分離できてしかるべきです。

Webアプリケーションの「デザイン」は主に、HTMLとスタイルシート(一般に「CSS」と呼ばれます)、そして本項では紹介していない「Webプログラミング」言語であるJavaScriptによって実現されます。これらは、今回実装したPythonのWebサーバー実装とは分離できます。

GAEでは、「jinja2」と呼ばれるテンプレートエンジンを用いて、デザインと実装の分離を実現することができます。ここではトップページの講義表にこの仕組みを使ってみましょう。また、デザインについては「Bootstrap」(<http://getbootstrap.com>)と呼ばれるソフトをHTMLに埋め込みます。

なお、今回はダウンロードしてプロジェクトに含める手続きを省略するため、Bootstrapの各種ファイルをオンラインで取得するようにHTML内で指示します。インターネット接続がない状態でローカルサーバーだけで開発している場合には見栄えがおかしくなるかもしれません。

まず、「jinja2」を使えるようにするために、「app.yaml」に以下を追加します。

app.yaml(「jinja2」に関する2行を追加)

```
libraries:  
- name: webapp  
  version: latest  
- name: jinja2  
  version: latest
```

その後以下のようないい「index.html」を新たに作成し、「HelloWorld」フォルダに置きます。

index.html

```
<!DOCTYPE html>  
<html lang="ja">  
  <head>  
    <meta charset="utf-8">  
    <meta http-equiv="X-UA-Compatible" content="IE=edge">  
    <meta name="viewport" content="width=device-width, initial-scale=1">  
    <title>講義表 with bootstrap</title>  
    <link href="//maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap.min.css" rel="stylesheet">  
  </head>  
  <body>  
    <div class="container">  
      <div style="text-align: center; padding-top: 40px;">  
        <h1>講義表ページ</h1>  
        <h2>総講義数: {{ course_count }}</h2>  
        {% if course_model_list %}  
          <table class="table" align="center"  
            style="width: 300px;">  
            <thead>  
              <tr><td>日付</td><td>講師</td><td>詳細</td></tr>  
            </thead>  
            <tbody>  
              {% for course_model in course_model_list %}  
                <tr>  
                  <td>{{ course_model.date.strftime('%Y-%m-%d') }}</td>  
                  <td>{{ course_model.title }}</td>  
                  <td>{{ course_model.teacher }}</td>  
                </tr>  
              {% endfor %}
```

```

        </tbody>
    </table>
    {% endif %}
    <div>
        <a href="/create">Create</a>
    </div>
</div>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.1/jquery.min.js"></script>
<script src="//maxcdn.bootstrapcdn.com/bootstrap/3.2.0/js/bootstrap.min.js"></script>
</body>
</html>

```

ほとんど通常のHTMLファイルのようですが、実は一部が「テンプレート化」されています。つまり、プログラムで後から特定の文字列を埋め込めるようになっています。ここでは以下のような2種類のテンプレートを使っています。

- `course_count`に後から講義の総数を埋め込む
- `course_model`にいくつかデータを入れる

Pythonコードは以下のようになります。HTMLを直接出力する必要がなくなるので、こちらの実装は簡潔になります。

helloworld.py(部分)

```

if output == 'html':
    course_count = course_model_list.count()
    template_values = {'course_count': course_count,
                       'course_model_list': course_model_list}
    template = JINJA_ENVIRONMENT.get_template('index.html')
    self.response.write(template.render(template_values))
else:

```

これで見栄えがだいぶ変わります。また、単に見栄えが良くなるだけではなく、実はセキュリティという面からも若干改善があります。テンプレートを用いたこのバージョンの場合、前述した「クロスサイト・スクリプティング攻撃」を避けるためのエスケープ処理をPython実装側で行う必要がありません。テンプレートエンジンは、与えられた文字列をチェックし、自動的にエスケープ処理を実行してくれるからです。もしエスケープを解除したいケースがあるようであれば、そのときだけ明示的にエスケープをしないように指定します。

プログラミングミスを全て回避することは熟練者でも難しいでしょう。「jinja2」テンプレートエンジンが採用している「していなければエスケープして安全よりに倒す」という方式の方が、安全なプログラミングとしては、より適切と言えます。

時間があれば、講義作成画面も、テンプレートを使って見栄えを変えてみましょう。



本章では、Androidアプリ開発者がWebサーバーを理解するための入り口と、より本格的に開発する上で必要な要素についても少し紹介しました。

Androidアプリ開発者としては、Webサーバーの全ての要素を理解する必要はありませんが、知っている方がサーバーとのやりとりを実装する上で落とし穴にはまりにくくなるでしょう。

参考まで、「Tech Booster」による「Android Open Textbook Project」(<https://github.com/TechBooster/AndroidOpenTextbook>)に、本章学習後に学ぶべきやや詳細なトピックを扱った内容を含めています。なお、最新版のPDF版やHTML版は、「<https://tcb.mowa-net.jp/griflet/github/TechBooster/AndroidOpenTextbook/>」から得ることができます(2014年10月現在)。

18-3 付録

著：宮川大輔

本節では、授業で用いる範囲で有用と思われる追加トピックを学びます。



この節で学ぶこと

- ・Pythonの基礎
- ・GitHubからのプロジェクトのインポート
- ・Windows 8.1でのシステムのバックアップ方法
- ・日本語フォルダー環境へのGAEのインストール
- ・Wi-Fi内でHello World!



18-3-1 Python を学ぶ

ここではPython言語そのものを学びます。

Windows 8.1を使用している場合、Python実行環境は付属していませんので、18-1-6「Python実行環境のインストール」を済ませてから読み進めてください。OS XにはPython実行環境は付属しています。

Pythonの対話型環境を起動する

対話型環境とは、ユーザの入力一行一行に対して応答を返してくれる実行環境のことです。文字通り「対話」しながらプログラム言語を試すことが出来ます。

18-1-9でインストールした統合開発環境PyCharmを利用している場合、「Tools」メニューの「Run Python Console」を選択することで、アプリ内で対話型環境を起動できます。試しに電卓のように使ってみましょう。

コマンド1

```
>>> 1 + 10  
11  
>>> 3 ** 10  
59049
```

PyCharmを利用してない場合でも、Python実行環境をインストール済であれば、Windowsのコマンドプロンプトから「python」と入力することでも起動できます。

「#」で始まる行はコメント
Pythonでは「#」以降の文字を行末まで無視します。Javaで言う「//」と同様です。本章では主にサンプルコードの解説を行うために使っています。
Javaでは「/*」と「*/」の組み合わせで複数行のコメントを実現出来ましたが、Pythonでは全く同じ機能を有するコメント機能はありません。

データ型について

- ・ 整数
- ・ 文字列
- ・ Unicode文字列
- ・ 真偽値(True／False。Javaのbooleanと異なり冒頭は「大文字」なので注意)
- ・ リスト(Javaの配列やList(ArrayList)に似た仕組み)
- ・ マップ型(JavaのMap(HashMap)に似た仕組み)
- ・ オブジェクト型

Javaのようなオブジェクト指向の仕組みはPythonにも存在しますが、本説では省略します。「class」で始まるブロックがあったら似たようなことが行われていると想像できれば十分です。

リストとマップ型は本編でも使われています。以下のコマンド実行例で、まず感覚をつかんでください。

コマンド2

```
>>> lst = [1, 2, 3]  
>>> print(lst)  
[1, 2, 3]  
>>> lst.append(5)  
>>> lst  
[1, 2, 3, 5]  
>>> d = {'a': 100, 'b': 200}  
>>> d  
{'a': 100, 'b': 200}  
>>> d['b']  
200  
>>> d['c'] = 1000  
>>> d  
{'a': 100, 'b': 200, 'd': 1000}
```

それぞれ「<http://docs.python.jp/2.7/library/index.html>」の「組み込み型 >> シーケンス型」と「組み込み型 >> マップ型」に、利用できる機能の全てが掲載されています。

Pythonでは変数の型指定は不要

Javaでは各変数を宣言する際、「型」を明示する必要がありました。Pythonではその必要はありません。

コマンド3

```
>>> x = 333
>>> y = 400
>>> x * y
133200
```

ただし、データが存在しないわけではありません。次の例は、整数と浮動小数点での割り算の挙動の違いを示しています。

コマンド4

```
>>> 1/2
0
>>> 1/2.0
0.5
```

定数や変数の型は「`type()`」で調べることができます。

コマンド5

```
>>> type(1)
<type 'int'>
>>> type(2.0)
<type 'float'>
```

型の名前と意味がJavaと似ていて異なることがあります。Pythonでは整数は`int`、浮動小数点は「`float`」のみが存在しますが、Pythonの「`int`」に最大数についての制限はなく、Pythonの「`float`」はJavaでいう「`double`」(倍精度浮動小数点)です。電卓のような用途では、JavaよりもPythonの方が数字を扱いやすいと言えるでしょう。

文字列について

Javaと同様、Pythonにも文字列があります。「」(ダブルクオート)もしくは「'」(シングルクオート)で囲った部分が文字列になります。Javaと異なり、Pythonでは「""」と「'''」で囲った場合で、文字列の挙動は基本的に変わりません。PythonにはC言語のchar型に相当する型は存在しません。

コマンド6

```
>>> "Hello World!"  
'Hello World'  
>>> 'Hello World!'  
'Hello World'
```

ダブルクオートで囲った文字列中でダブルクオートを使いたい場合は、バックスラッシュでエスケープする必要があります。

コマンド7

```
>>> print("Say \"Hello World!\"")  
Say "Hello World!"
```

シングルクオートで囲った文字列内では、ダブルクオートをエスケープする必要はありません。同様に、ダブルクオートで囲った文字列内では、シングルクオートをエスケープする必要はありません。

コマンド8

```
>>> print('Say "Hello World!"')  
Say "Hello World!"
```

ダブルクオートもしくはシングルクオートを3つ並べた場合、複数行の文字列を作成することができます。18-2-7「講義データを作るための画面を作る」では実際に、複数行に渡るHTMLを、まとめてソースコードに記述する際に使いました。

コマンド9

```
>>> print('''Hello  
... World  
... !!!!!'))  
Hello  
World  
!!!!!
```

Pythonでは文字列型と数値型は「+」演算子で結合できません。

コマンド10

```
>>> 'Hello ' + 20 + ' and ' + 10 + '!!!'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'int' objects
```

数字や他の文字列を埋め込む最も単純な方法は、数字などを埋め込みたい場所に「{}」を挿入した上で、文字列のformat()メソッドを呼ぶ方法です。

コマンド11

```
>>> print('Hello {} and {}'.format(20, 10))
Hello 20 and 10!!
```

「{}」という書式はPythonにおける「書式指定文字列」と呼ばれ、Javaの「System.out.printf()」に負けない非常に多彩な機能をもっています。「<http://docs.python.jp/2/library/string.html#formatstrings>」等で詳細な機能を確認することができます

Python文字列での日本語の扱い

Pythonの文字列についてしばしば混乱を招くのが、日本語のような、いわゆる「非アスキー文字」の扱いです。Pythonについて学ぶ際、文字列の前に「u」をつけて「u'日本語'」のように表記する例を見ることがあるかもしれません。これは「ユニコード文字列」と呼ばれる文字列で、本来の文字列と異なります。

コマンド12

```
>>> type('str')
<type 'str'>
>>> type(u'str')
<type 'unicode'>
>>> type('str') == type(u'str')
False
>>> 'str' == u'str'
True
>>> '日本語' == u'日本語'
__main__:1: UnicodeWarning: Unicode equal comparison failed to convert both arguments to Unicode
- interpreting them as being unequal
False
```

Python 3では文字列の扱いが大幅に変わっています。

最後の例（「'日本語' == u'日本語'」）では、不穏な警告まで表示されます。その上で文字列とユニコード文字列の比較では、結果は「False」です。Python言語では日本語の扱い方に注意が必要、とだけ覚えておきましょう。

ソースコードの記述されたファイルを実行する

PyCharm等で作成したプログラムファイルを、Pythonの実行環境に実行させることができます。例えば次のリストのようなプログラムを作成し、「simpleprogram.py」として保存したとします。

simpleprogram.py(簡単なプログラムの例)

```
x = 10
y = 20
z = 30
x = y * z - x
y = z * x - y
z = x * y - z
print(x + y + z)
```

5-1-2「Javaプログラムを実際に動かしてみよう」に登場した、Javaの「System.out.println()」と同等です。

PyCharmで、このPythonファイルを右クリックし「Run 'xxx'」を選択すると、結果がPyCharm内に表示されます。ファイルに保存して実行するときは、表示させたい結果は「print()」で囲ってください。

対話型環境で、次のように毎回1行1行入力することもできますが、この場合はプログラムがどこにも保存されていないため、対話型環境を再起動したりした場合には、最初から入力し直す必要があります。

コマンド13

```
>>> x = 10
>>> y = 20
>>> z = 30
>>> x = y * z - x
>>> y = z * x - y
>>> z = x * y - z
>>> print(x + y + z)
(出てくる数字はいくつ?)
```

Pythonではインデントに意味がある

ここで、Pythonを学ぶ初学者が陥りがちな罠を紹介しましょう。Pythonではインデントに意味があります。「if」文の書き方を通じてJavaとPythonを比較してみましょう。まずJavaの「if」文の例を以下に示します。

Javaのif文

```
if (x == 0) {  
    System.out.println("x is 0");  
} else {  
    System.out.println("x is not 0");  
}  
System.out.println("After if.");
```

次にPythonで同様にif文を書いた場合は以下のとおりです。

Pythonのif文

```
if (x == 0):  
    print("x is 0")  
else:  
    print("x is not 0")  
print("After if.")
```

ブロックの開始は「:」(コロン)で指定していますが、終了を明示するものはありません。その代わり、行頭に空白が存在することで**ifプロックの中であることとその外に存在することを区別します。**

Pythonでは、次のように書いてはいけません。

Pythonのif文(間違った例)

```
# これは正しくない  
if (x == 0):  
print("x is 0")  
else:  
print("x is not 0")  
print("After if.")  
# これは正しくない
```

これはエラーになります。また、「x is not 0」と「After if.」の行についてはブロックの内外の区別ができません。インデントつまり行頭の空白が、各行がどのブロックに属しているかを示します。次の例を見てみましょう。

Pythonのif文(正しい例)

```
if (x > 0):  
    print("x > 0")  
    if (x < 30):  
        print(u"0 < x < 30")  
    else:  
        print(u"x >= 30")  
print("After if.")
```

これを次のように書くとprint()文と実際のxの中身が一致しなくなります。

Pythonのif文(間違った例)

```
# これは正しくない
if (x > 0):
    print(u"x > 0")
    if (x < 30):
        print(u"0 < x < 30")
else:                                # この行以降のインデントが壊れたとする。
    print(u"x >= 30")                 # x が例えば -30 であってもこれが表示される。
print("After if.")
# これは正しくない
```

例えば、コピー＆ペーストをした際に、上記のような意図しないインデントが発生しがちです。

「if」

既にifは登場していますが、説明していないことがあります。Pythonでは「if」の評価文前後の丸括弧は必要ありません。

ifの例

```
if x > 0: # ()がない
    print(u"x is greater than 0")
```

「if」の後が1行であれば、次のようにも書けます。

ifの例

```
if x > 0: print(u"x is greater than 0")
```

Javaの「if」文は「boolean」型しか評価の対象としませんでしたが、Pythonではデータ型は概ね何でも受け取ります。真偽を表す「True」と「False」(いずれも先頭が「大文字」です)以外の場合、おおまかにそのデータが「空でないか」をチェックします。空の意味は、以下のように変数に実際に保存されているデータの型で変わります。

- ・ 整数なら 0
- ・ 文字列なら "もしくは"
- ・ リストなら []
- ・ マップなら {}

真偽値以外を指定して、以下のように書くこともできます。

ifの例

```
if x: print("x is not empty")
```

なお、このような書き方を許す言語はPythonに限らず多数ありますが、逆にそれが混乱を招く可能性があります。例えば、Pythonと良く比較対象になるRuby言語では、整数の0を真と判定するか、偽と判定するかについて、Pythonと異なります。Rubyでは0は真で、Pythonでは上述の通り0は偽です。よって、Rubyとそっくりだと思って「if x:」と書いた場合、その結果として分岐が完全に入れ替わる可能性があります。

Rubyのコードだとyay!が出る(0は「真」)

```
(rubyの対話環境)> x = 0
(rubyの対話環境)> if x; "yay!"; end
=> "yay!"
```

「for」

forはJavaと文法が異なります。例えば0から4まで表示するプログラムは以下のとおりです。

コマンド14

```
>>> for i in range(5): print(i)
... (何も入力せずにエンター)
0
1
2
3
4
コマンドここまで>
```

「関数」

Pythonの関数は次のように書きます。インデントの有無で関数の開始と終了を明示します。ifやforと同様、中括弧は必要ありません。

関数の例

```
def sum(a, b):
    return a + b
# インデントをなくして関数の終わりを明示する。
print(sum(1, 9)) # 10を表示する
```

標準ライブラリーとimport

Pythonではインストールした段階で豊富なライブラリーが付属しています。JSONデータを扱う「json」モジュールを例に、標準ライブラリーに含まれるモジュールの使い方を説明します。Pythonでは外部ライブラリーは明示的に利用することを宣言する必要があります。jsonモジュールを使うには「import json」とします。これ以降、対話型環境およびそのソースコードファイルではjsonモジュールを使うことができます。

試しにPythonマップ型データをJSONデータに変換してみます。

コマンド15

```
>>> import json  
>>> json.dumps({'a': 1, 'b': 2})  
'{"a": 1, "b": 2}'
```

一見、入力したデータと「json.dumps()」関数の結果がそっくり同じに見えますが、よく見ると関数の出力結果は「文字列」です。Pythonのデータ構造とJSONフォーマットの文字列データは一見してよく似ています。

Python 2.7の持つライブラリの内容は「<http://docs.python.jp/2.7/library/index.html>」から参照できます。特に、jsonモジュールの説明は「<http://docs.python.jp/2.7/library/json.html>」にあります。

モジュール内的一部だけインポートしたいことがあります。本教科書の範囲で言うと、18-2-8「セキュリティ」で「xml.sax.saxutils」モジュールの「escape()」関数を使う例がありますが、毎回「xml.sax.saxutils.escape()」と書くのは面倒なので、以下のようにimport文を記述して、プログラム本体は「escape()」だけを書けば良いようにしています。

モジュール部分の指定を省略できるようにしている

```
from xml.sax.saxutils import escape
```

Pythonモジュールのインポート方法の詳細は、「<http://docs.python.jp/2/tutorial/modules.html>」などを参照してください。



18-3-2 シラバスアプリのインポート

第10章「ユーティリティによる実践」でシラバスアプリが完成していない場合にだけ本作業を行ってください。この作業を行う場合にも、同章に記述された「Volley」ライブラリの準備は必要です。10-2-1の「ネットワーク経由でデータを取得する」で「Volley」ライブラリの準備方法は扱われていますので、ここでは割愛します。

まず「<https://github.com/ktaka/TechInstituteSyllabus>」へアクセスします。図53のようなWebページが表示されたら、右下の「Download ZIP」ボタンをクリックします。

The screenshot shows a web browser window with the GitHub interface. The URL in the address bar is <https://github.com/ktaka/TechInstituteSyllabus>. The repository name is **TechInstituteSyllabus**. The repository details section shows 4 commits, 1 branch (branch: master), 2 releases, and 1 contributor. Below this, a list of files is shown:

- libs: Minimum requirements. (a month ago)
- res: Added progress bar and view holder for ListView. (23 days ago)
- src/jp/techinstitute/syllabus: Added progress bar and view holder for ListView. (23 days ago)
- .gitignore: Initial commit (a month ago)
- AndroidManifest.xml: added network access function. (28 days ago)
- LICENSE: Initial commit (a month ago)
- README.md: Initial commit (a month ago)
- ic_launcher-web.png: added network access function. (a month ago)

At the bottom right of the main content area, there is a button labeled "Download ZIP". To its left, a tooltip says "Download the contents of ktaka/TechInstituteSyllabus as a zip file".

図53:GitHubから「シラバスアプリ」をダウンロードする

ダウンロードしたzipファイルを解凍し、解凍したフォルダー「TechInstitute Syllabus-master」の場所を憶えておきます。今回は「デスクトップ」へ配置いたします。

Eclipseの「File」メニューから「New」→「Project...」を選択します(図54)。

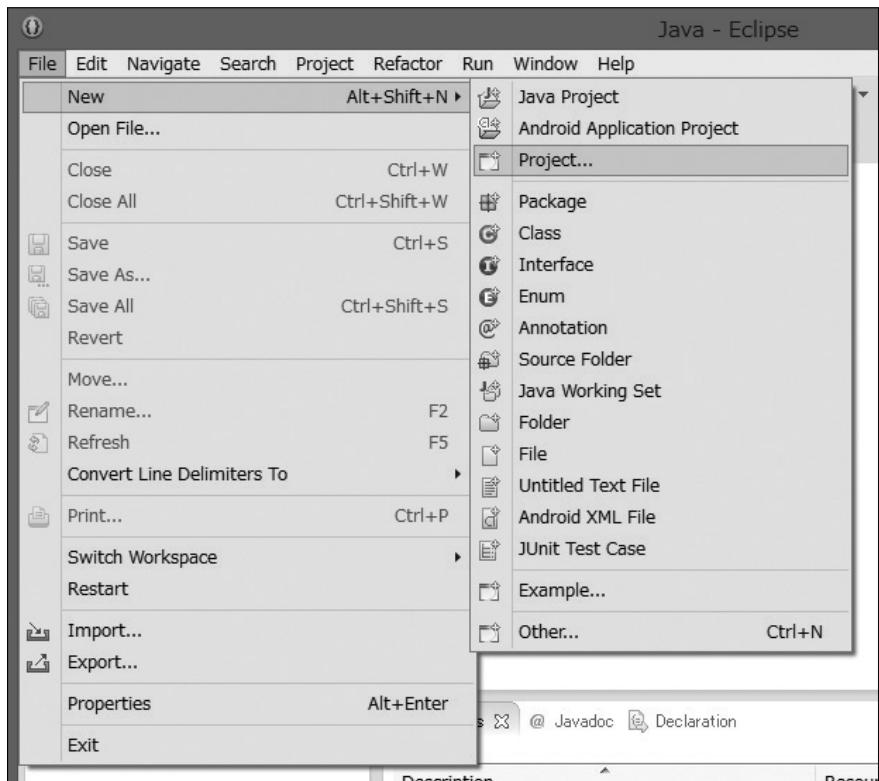


図54:Eclipseでプロジェクトをインポートするコマンドを選択する

表示されるダイアログで「Android」の中の「Android project from Existing Code」(既存のAndroidプロジェクト)を選択します(図55)。

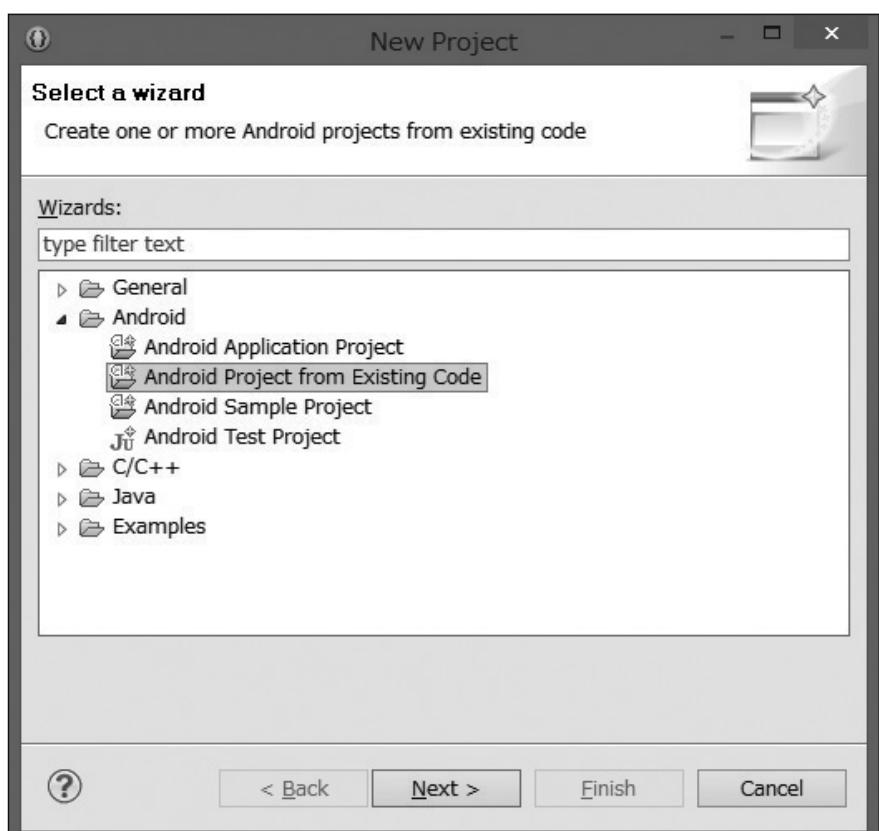


図55:Androidプロジェクトをインポートするオプションを選ぶ

すると「Import Projects」というダイアログが表示されるので、右上の「Browse...」ボタンを押します。

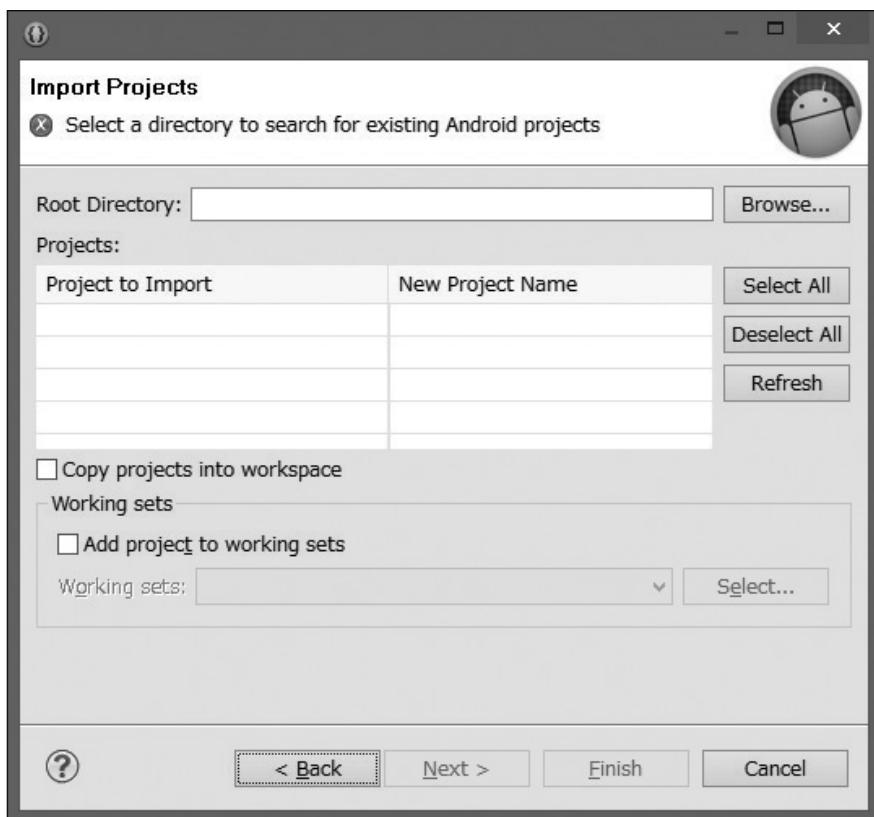


図56:「Browse...」ボタンをクリックする

そこで、解凍した「TechInstituteSyllabus-master」を選択して「Finish」を押します(図57)。

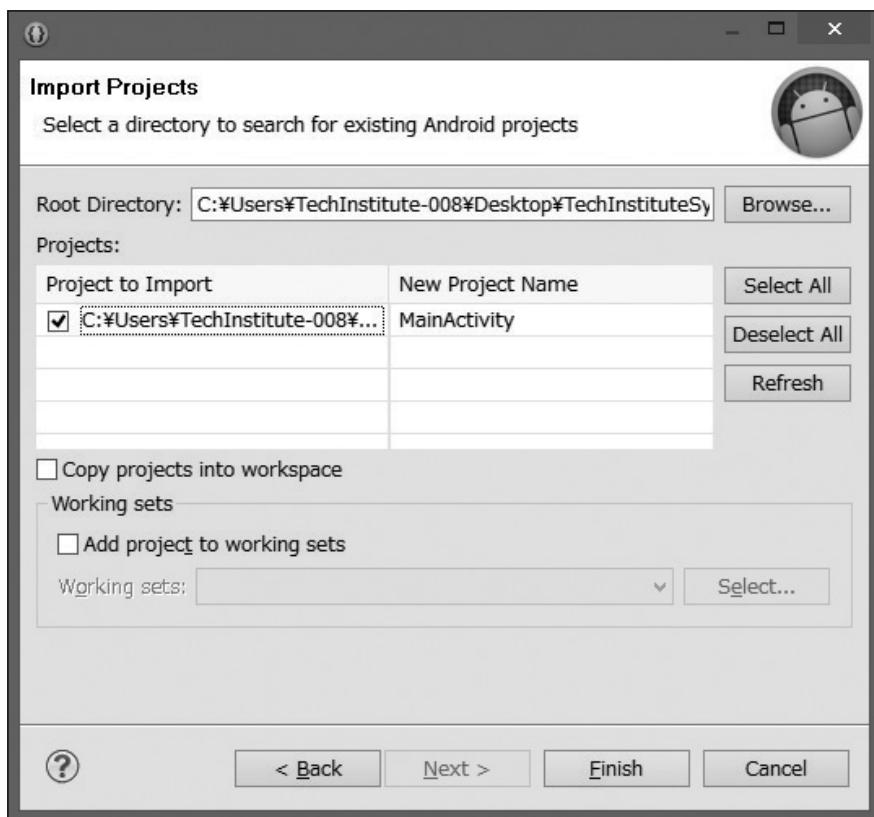


図57:「TechInstituteSyllabus-master」を選択して「Finish」をクリックする

このプロジェクトは「MainActivity」という名前でEclipseに認識されます。ここで、インポートしたプロジェクトがビルドエラーが発生している場合があります。この場合、そのプロジェクトを右クリックし、「Properties」を選択します(図58)。

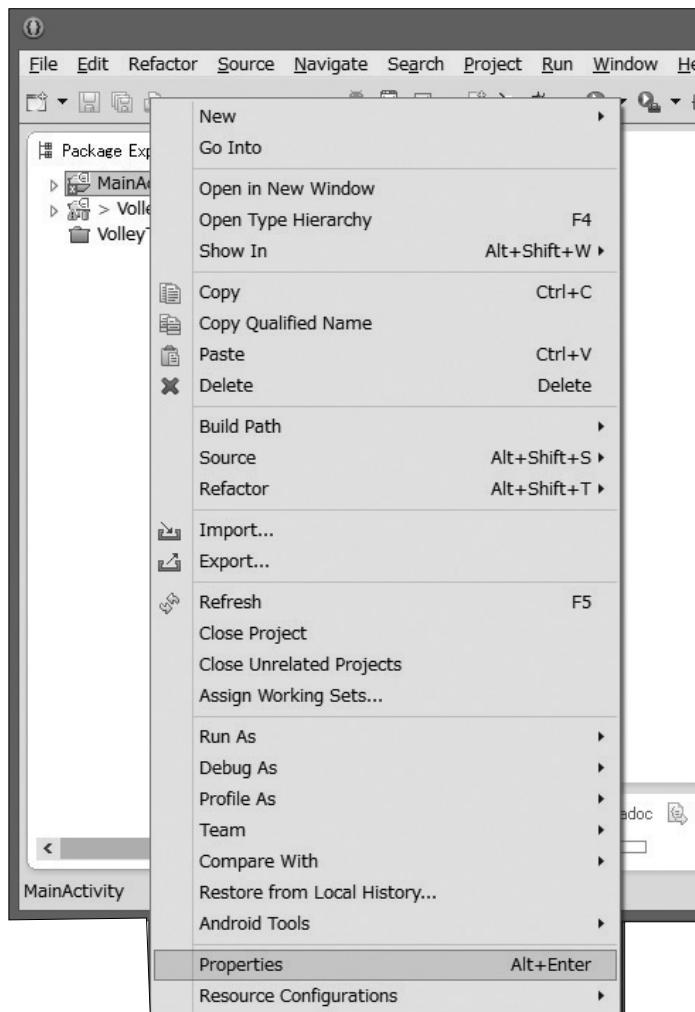


図58:ビルドエラーが発生したプロジェクトの上で右クリックして「Properties」を選ぶ

プロパティダイアログの左ペインで「Android」を選択します。もしビルドに失敗している場合、次にいずれかの状態になっているはずです。

- ・右側の「Project Build Target」のチェックがどこにもついていない
- ・右下のLibrary部分で「volley」という行の左に赤い「×」がついている

1番目の問題が発生している場合、授業で案内されているバージョン(多くのケースでAndroid 4.4.2)を選択します。2番目の問題の(赤い「×」印がついている)場合(図59)、右側のRemoveボタンをまず押し、次にAddボタンを押します。

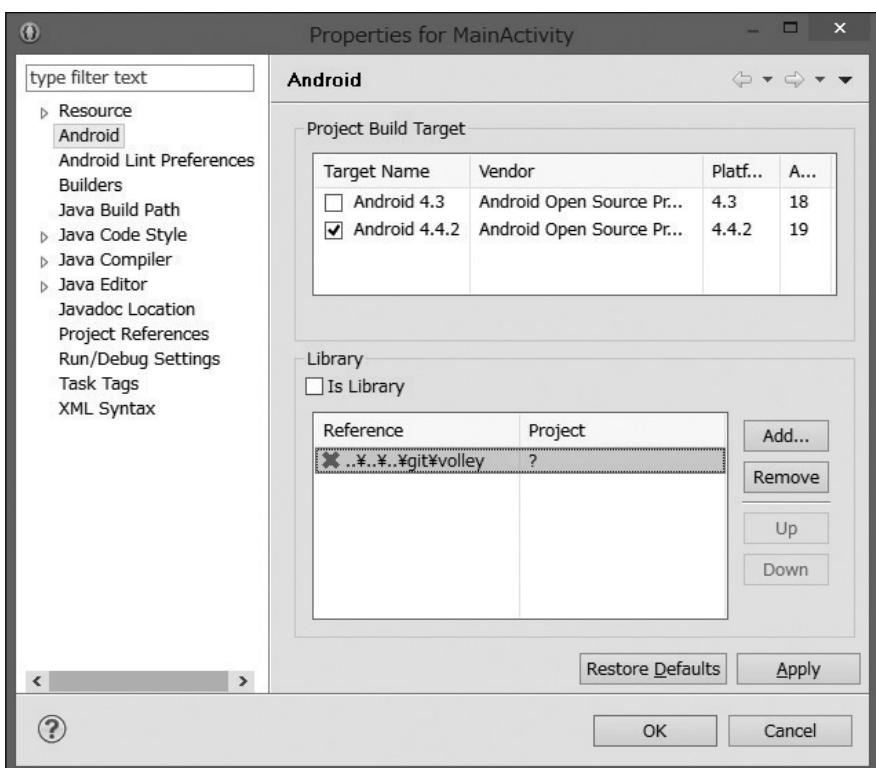


図59:「Library」の「volley」に赤い「×」が付いている

「Project Selection」というウィンドウが開いたら、「Volley」を選択して、「OK」を押します(図60)。

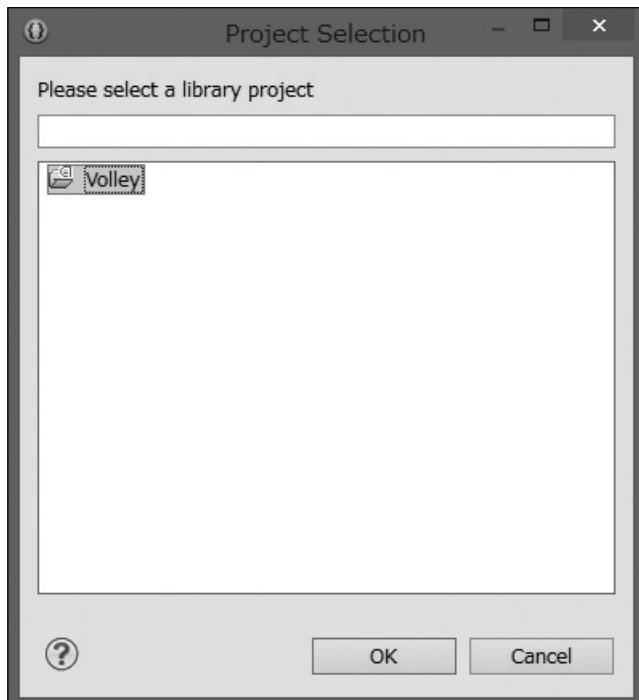


図60:「Volley」を選んで「OK」をクリックする

その結果、図61のように赤い×印が緑のチェックマークになったことを確認したら「OK」を押して「Properties for MainActivity」を閉じます。その後Eclipseの

メイン画面の「Project」メニューから「Clean...」を選択して「Clean all projects」を実行してください。

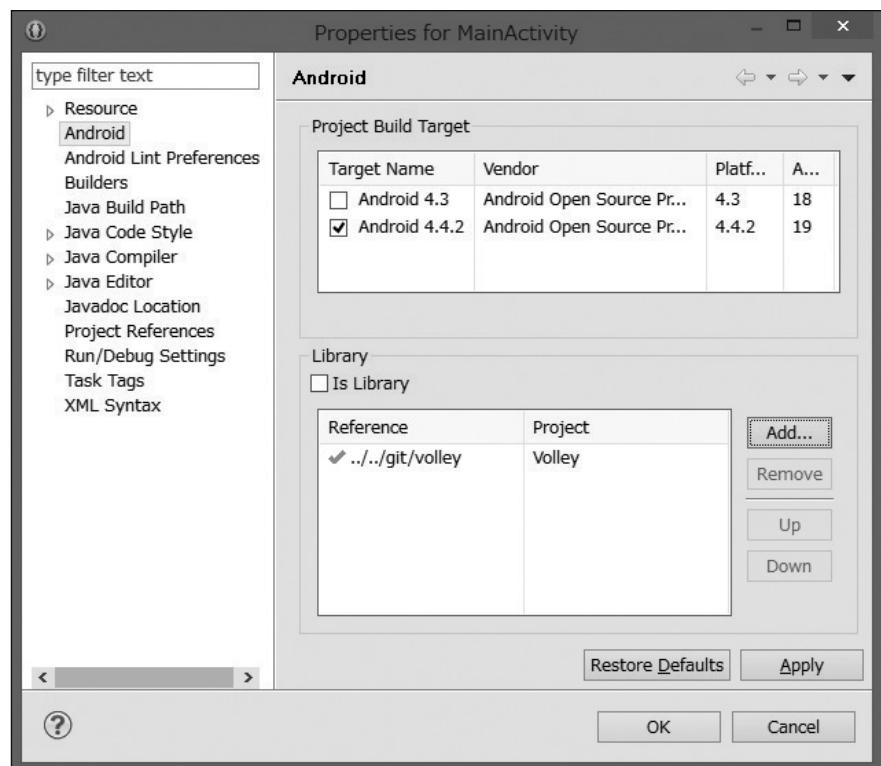


図61:「Library」の「volley」が緑のチェックマークになった

これで、第10章「ユーティリティによる実践」のシラバスアプリの完成版を利用できるようになります。



18-3-3 Windows 8.1 でのシステムのバックアップ方法

ここではWindowsにおける「復元ポイント」の作成方法を説明します。本演習でインストールしたソフトウェアや中間生成物を一切残したくない時に、演習終了後にその復元ポイントに戻れば、それまでのPC内の変更を取り消すことができます。

なお、ユーザーのディレクトリ内（デスクトップなどに作成したファイル）に作成したファイルはそのまま残りますが、その間に行われた他のシステム関連の作業についてもリセットされてしまいます。「復元ポイント」自体にディスク容量も必要なため、実際にこれを実行するかどうかは各自判断してください。

まず左下のスタートボタンを右クリックし「システム」を選択します（図62）。



図62:左下のスタートボタンを右クリックして「システム」を選ぶ

その結果表示される「システム」ウインドウ左側の「システムの保護」をクリックします(図63)。



図63:「システム」ウインドウで「システムの保護」をクリックする

「システムのプロパティ」ウィンドウ下方の「作成...」をクリックします(図64)。



図64:「システムのプロパティ」で復元ポイントの「作成...」をクリックする

「復元ポイントの作成」において分かりやすい名前を入力します。図65では「manual-recovery」と入力していますが、覚えやすいものであれば、他の文字列でも構いません。名前を入力したら「作成」ボタンを押します。



図65:復元ポイントの名前を入力して「作成」をクリックする

復元ポイントの作成には、PC内のデータ量に応じた時間がかかります。最後に

図66のように表示されれば成功です。

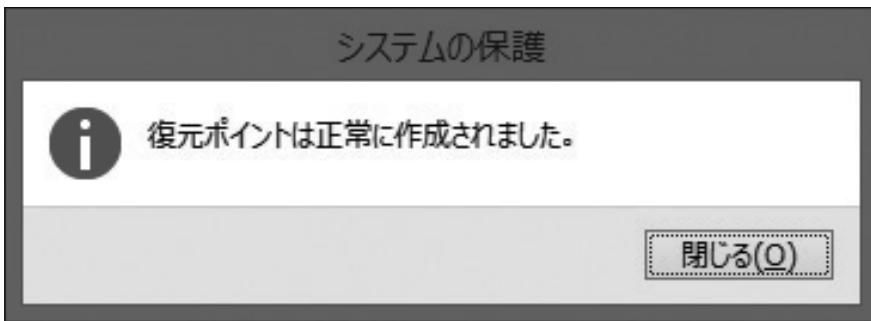


図66:復元ポイントの作成に成功した

実際に復元したくなった場合には、図64の「システムの復元...」ボタンを押して、指示に従います。

18-3-4 ユーザーフォルダーに日本語が含まれる場合の GAE インストールの追加作業

GAEインストール時にこの作業が必要なのは、「C:\ユーザー\（ユーザー名）」の「ユーザー名」部分が日本語になっている場合だけです。「Google App Engine Launcher」が日本語を含むフォルダーの扱いを正しく処理できないため、この作業が必要となります。そのような例を図67に示します。この現象は、Windows 8 の初期設定で、日本語名のユーザーを作成した際に発生します。

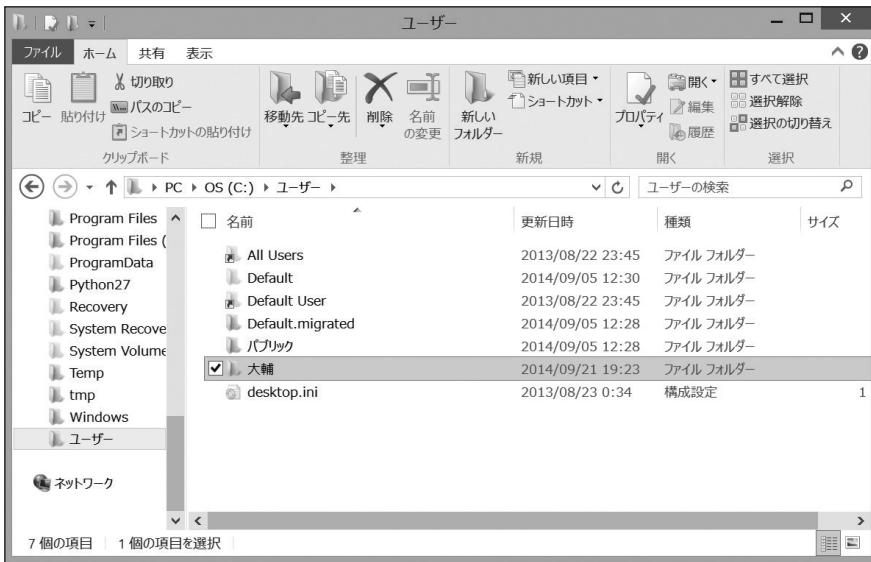


図67:ユーザー名が「大輔」で日本語になっている

このケースに当てはまる場合、以下の作業を行なってください。

- ・「C:\GoogleAppEngine」のような日本語を含まないフォルダーを作成する
- ・TMP環境変数に上記のアドレスを指定する

環境変数は第4章「開発環境セットアップ」P.61でPATHを指定した時と同じ方法で設定します。今回、変数名は「TMP」とし「変数値」を上記のフォルダーとします。既に存在している場合にはいったん値を削除してください。

・プロジェクトを上記フォルダーに作る

この作業を行わずにGAEの各種作業を行おうとすると、以下のような問題が発生します。

- ・テスト用の「sqlite DB」が作成できないという旨のエラーが発生する
- ・GAEが動作しなくなる
- ・GAEが起動しなくなる

特に、最後の挙動に至った場合、PCの再起動等では復帰しません。そのPCの「ユーザー」フォルダーに「Google」というフォルダーができているはずなので、そのフォルダーの中にある「GAE Launcher」が作成する一時ファイルを全て削除すると復帰します。



18-3-5 Wi-Fi 内で Hello World!

同じWi-Fiネットワーク内の他の人にだけサーバーを見せたい、というケースがあるかもしれません。その場合には、まず14-1-2「IPアドレスとは?」に登場した「ipconfig」(Mac等では「ifconfig」)コマンドを用いて、自身のIPアドレスを確認します。注意点として「127.0.0.1」といったIPアドレスではない、もうひとつのIPアドレスを探してください。

コマンド

```
> ipconfig  
...  
10.0.90.181  
...
```

このようになっていたら、「GAE Launcher」の「Edit」メニューから「Application Settings...」を選び、「Extra Command Line Flags」に「--host=10.0.90.181」と追加します(図68)。IPアドレス部分は状況に応じて変更してください。

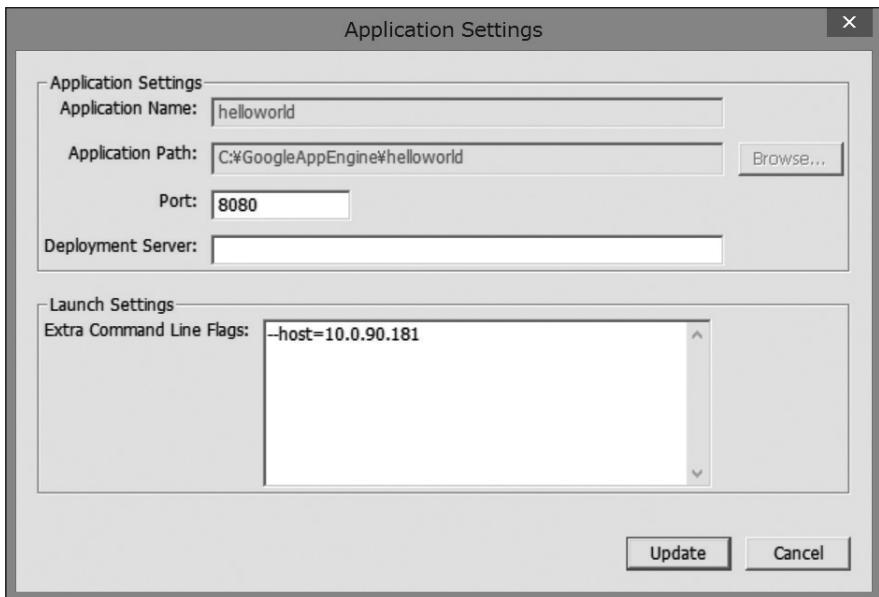


図68:自分のIPアドレスを「Extra Command Line Flags」に設定する

このあとに起動すると、「`http://(自分のIPアドレス):8080/`」でローカルサーバーに接続するようになります。このようにホスト側IPアドレスを指定してサーバーを起動すると、「`http://localhost:8080`」ではWebページは表示されなくなります。代わりに、指定したIPアドレスを含むURLでだけ表示できます。

このIPアドレスとそのURLは、Wi-Fiを共有している人たちの間でだけ共有することができます。「`http://(自分のIPアドレス):8080/`」を隣の人に教えれば、隣の人のPCやAndroid端末から、そのWebページを見られるようになります。この方法は、世界中にサーバーを公開せず、近くの人と挙動を確認しあうのに使えます。18-2で使用するシラバスアプリとも、Wi-Fiネットワークの範囲内であれば、通信を行えます。

教室や自宅で現在利用されている多くのネットワークでは、NAT(Network Address Translation)と呼ばれる技術によって、インターネットと内部のネットワークが分断されています。そういった状況でのIPアドレスはインターネットからは見えないため、「世界に公開」していることにもなりません。



●シラバスサーバーの拡張

- ・テンプレートをフォーム画面にも適用してみましょう。
- ・講義データの変更、削除に対応してみましょう。
- ・Android アプリから講義データの管理を行う方法を調べてみましょう。
- ・受講生のアカウントから講義の質問や感想を送れるようにしてみましょう。

●過去の授業のサーバーサイド実装

第13章「センサー」、第14章「ネットワークプログラミング」などで、講師が用意したサーバーが登場しています。同じ機能を持つWebサーバーを自力で実装してみましょう。