コラボレイティブ開発特論

産業技術大学院大学 中鉢欣秀

2016 年度

目次

1 Part 1: ガイダンスとモダンな道具達

1.1 第 1 章ガイダンス

1.1.1 連絡事項

■連絡事項 (1)

1. 資料等の入手先
   - GitHub の下記リポジトリにまとめておきます
     - https://github.com/ychubachi/collaborative_development
   - 資料は随時 update するので，適宜，最新版をダウンロードしてください
2. Twitter のハッシュタグ
   - Twitter ハッシュタグ -> #enpit_aiit

■連絡事項 (2)

1. 仮想環境（Vagrant）
   - 各自の PC に仮想環境をインストールしておいてください
     - PC を持ってない方には貸出も可能です（数量限定）
   - インストール方法については下記を参照
     - https://github.com/ychubachi/cldv_preparation

1.1.2 授業の全体像

■学習の目的

- ビジネスアプリケーションを構築するための基礎力
- 分散型 PBL を実施する上で必要となる知識やツールの使い方
- これら活用するための自己組織的なチームワーク

■学習の目標

- 分散ソフトウェア開発のための道具を学ぶ
  - 開発環境（Ruby），VCS とリモートリポジトリ（GitHub）
  - テスト自動化，継続的インテグレーション，PaaS
- これらのツールの **設計思想**に対する本質理解

■前提知識と到達目標

1. 前提とする知識
   - 情報系の学部レベルで基礎的な知識を持っていること
2. 最低到達目標
   - 授業で取り上げる各種ツールの基本的な使い方を身につける
3. 上位到達目標
   - 授業で取り上げる各種ツールの高度な使い方に習熟する．

■授業の形態

1. 対面授業
   - 担当教員による講義・演習
2. 個人演習
   - 個人によるソフトウエア開発
3. グループ演習
   - グループによるソフトウエア開発

1.1.3 授業の方法

■講義・演習・課題

1. 講義
   - ツールの説明
   - ツールの使い方
2. 演習
   - 個人でツールを使えるようになる
   - グループでツールを使えるようになる

■成績評価

1. 課題
   - 個人でソフトウエアを作る
   - グループでソフトウエアを作る
2. 評価の方法
   - 課題提出と実技試験

3. 評価の観点
    - 分散 PBL で役に立つ知識が習得できたかどうか

1.1.4 自己紹介

■自己紹介

1. 名前
    - 中鉢欣秀（ちゅうばちよしひで）
2. 出身地
    - 宮城県仙台市
3. 肩書
    - 産業技術大学院大学産業技術研究科
      情報アーキテクチャ専攻准教授

■連絡先

E-Mail　yc@aiit. . .
Facebook　ychubachi
Twitter　ychubachi（あんまり使ってない）
Skype　ychubachi（あんまり使ってない）

■学歴

1991 年	4 月	慶應義塾大学環境情報学部入学
1995 年	**10** 月	同大大学院政策・メディア研究科 修士課程入学
1997 年	10 月	同大大学院政策・メディア研究科 後期博士課程入学
2004 年	10 月	同大大学院政策・メディア研究科 後期博士課程卒業 学位：博士（政策・メディア）

■職歴

1997 年	10 月	合資会社ニューメリック設立 社長就任
2005 年	4 月	独立行政法人科学技術振興機構 PD 級研究員 （長岡技術科学大学）
2006 年	4 月	産業技術大学院大学産業技術研究科 情報アーキテクチャ専攻准教授

■起業経験

1. 社名
    - 合資会社ニューメリック
2. 設立
    - 1997 年
3. 資本金
    - **18** 万円

■起業の背景

1. 設立当時の状況
    - Windows 95 が普及（初期状態でインターネットは使えなかった）
    - 後輩のやっていたベンチャーの仕事を手伝って面白かった
2. 会社設立の理由
    -「やってみたかった」から
    - 少しプログラムがかければ仕事はいくらでもあった
    - 後輩にそそのかされた・笑

■起業から学んだこと

- 実プロジェクトの経験
- 使える技術
- お金は簡単には儲からない

■教育における関心事

1. 情報技術産業の変化
    - 情報技術のマーケットが変化
    - ユーザ・ベンダ型モデルの終焉
2. モダンなソフトウエア開発者
    - 新しいサービスの企画から，ソフトウエアの実装まで何でもこなせる開発者
    - このような人材の育成方法

1.1.5 「学びの共同体」になろう

■共に学び、共に教える「場」

- 教室に集うメンバーで 学びの共同体になろう
- 困った時には助けを求める
- 他人に教えること＝学び

■チーム演習での問題解決（理想の流れ）

1. 困った時はメンバーに聞く
2. わからなかったらチーム全員で考える
3. それでもダメなら他のチームに相談
4. 講師・コーチに尋ねるのは最終手段！
5. …となるのが理想
    - 授業の進め方などの質問は遠慮無く聞いてください

■共同体になるためにお互いを知ろう

- 皆さんの自己紹介
    - 名前（可能であれば所属も）
    - どんな仕事をしているか（あるいは大学で学んだこと）
    - この授業を履修した動機

1.2 第 2 章コラボレイティブ開発の道具達

1.2.1 モダンなソフトウエア開発とは

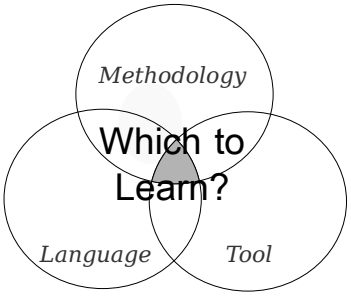

図1　The Framework-Language-Tool framework.

■ソフトウエア開発のための方法・言語・道具

■授業で取り上げる範囲

1. 取り上げること

- 良い道具には設計思想そのものに方法論が組み込まれている
- 世界中の技術者の知恵が結晶した成果としての OSS のツール

2. 取り扱わないこと
   - 方法論そのものについてはアジャイル開発特論で学ぶ
   - プログラミングの初歩については教えない

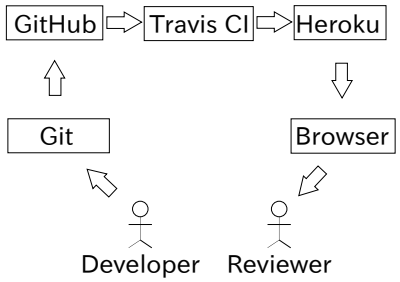

図2　The modern tools for Scrum developments.

■Scrum するための道具
■モダンな開発環境の全体像

1. 仮想化技術（Virtualization）
   - Windows や Mac で Linux 上での Web アプリケーション開発を学ぶことができる
   - Heroku や Travis CI 等のクラウドでの実行や検査環境として用いられている
2. ソーシャルコーディング（Social Coding）
   - Linux のソースコードの VCS として用いられている Git を学ぶ
   - Git は GitHub と連携することで OSS 型のチーム開発ができる

■enPiT 仮想化環境

1. 仮想環境にインストール済みの道具
   - エディタ（Emacs/Vim）
   - Ruby の実行環境
   - GitHub，Heroku，Travis CI と連携するための各種コマンド（github-connect.sh，hub，heroku，travis）
   - PostgreSQL のクライアント・サーバーと DB
   - 各種設定ファイル（.bash_profile，.gemrc，.gitconfig）
   - その他
2. 仮想化環境の構築用リポジトリ（参考）
   - ychubachi/vagrant_enpit

1.2.2　仮想環境の準備から起動

■enPiT 仮想化環境のアップデート

1. 作業内容
   - enPiT 仮想化環境（vagrant の box）を更新しておく
2. コマンド

```
cd ~/enpit
vagrant destroy
vagrant box update
```

■Port Forward の設定 (1)

1. 説明
   - Guest OS で実行するサーバに，Host OS から Web ブラウザでアクセスできるようにしておく
   - 任意のエディタで Vagrantfile の「config.vm.network」を変更
   - 任意のエディタで Vagrantfile を変更

■Port Forward の設定 (2)

1. 変更前

```
# config.vm.network "forwarded_port",
  guest: 80, host: 8080
```

2. 変更後

```
config.vm.network "forwarded_port", guest:
  3000, host: 3000
config.vm.network "forwarded_port", guest:
  4567, host: 4567
```

■enPiT 仮想化環境にログイン

1. 作業内容
   - 前の操作に引き続き，仮想化環境に SSH 接続する
2. コマンド

```
vagrant up
vagrant ssh
```

1.2.3　クラウド環境のアカウント・設定

■GitHub/Heroku のアカウントを作成

1. GitHub
   - Join GitHub ·GitHub
2. Heroku
   - Heroku - Sign up
3. Travis CI
   - Travis CI
     - Travis CI は，GitHub のアカウントでログインできる

■github-connect スクリプト

1. URL
   - github-connect.sh
2. git conifg を代行
   - GitHub にログインし,名前と email を読み込んで git に設定
3. SSH の鍵生成と登録
   - SSH 鍵を作成し，公開鍵を GitHub に登録してくれる

■github-connect.sh の実行

- スクリプトを起動し，設定を行う
- GitHub のログイン名とパスワードを聞かれるので，入力する
- rsa key pair のパスフレーズは入力しなくて構わない

1. コマンド

```
github-connect.sh
```

■Git と GitHub の設定確認

1. Git の設定確認

```
git config --list
```

2. GitHub の設定確認
   - ブラウザで GitHub の SSH Key ページを開く

1.2.4 演習: GitHub ユーザ名の提出

■演習: GitHub ユーザ名の提出

- 次の URL から授業で利用する GitHub ユーザ名と URL を登録してください.
  - コラボレイティブ開発特論-GitHub ユーザ名と URL

2 Part 2: GitHub 入門（別資料）

2.1 第 3 章 GitHub 入門

2.1.1 GitHub 入門の解説と演習

■GitHub 入門について

1. GitHub 入門 Git と GitHub にとことん精通しよう
2. 演習資料 ychubachi/github_practice: Git/GitHub 入門

3 Part 3: Sinatra/Heroku

3.1 第 4 章 Sinatra で Web アプリを作ろう

3.1.1 Sinatra アプリケーションの作成

■Sinatra を使った簡単な Web アプリケーション

1. Sinatra とは？
   - Web アプリケーションを作成する DSL
   - Rails に比べ簡単で，学習曲線が緩やか
   - 素早く Web アプリを作って Heroku で公開してみよう
2. 参考文献
   - Sinatra
   - Sinatra: README

■Sinatra アプリ用リポジトリを作成する

- Sinatra アプリを作成するため，新しいリポジトリを作る
  - Web ブラウザで GitHub を開き，作成できたことを確認

1. コマンド

```
mkdir ~/sinatra_enpit
cd ~/shinatra_enpit
git init
git create
```

■Sinatra アプリを作成する (1)

- エディタを起動し，次のスライドにある「hello.rb」のコードを入力

1. コマンド

```
emacs hello.rb
git add hello.rb
git commit -m 'Create hello.rb'
```

■Sinatra アプリを作成する (2)

- Sinatra アプリ本体のコード（たった 4 行！）

1. コード: **hello.rb**

```ruby
require 'sinatra'

get '/' do
  "Hello World!"
end
```

■Sinatra アプリを起動する

1. 起動の方法
   - hello.rb を ruby で動かせば，サーバが立ち上がる
     - vagrant の port forward を利用するため，「-o」オプションを指定する
2. コマンド

```
ruby hello.rb -o 0.0.0.0
```

■Sinatra アプリの動作確認

1. 動作確認の方法
   - **Host OS** の Web ブラウザで，`http://localhost:4567` にアクセスする.
     -「Hello World!」が表示されれば成功

■参考文献

- ruby - Unable to access Sinatra app on host machine with Vagrant forwarded ports - Stack Overflow

3.1.2 Heroku で Sinatra を動かす

■Sinatra アプリのディプロイ

- Sinatra アプリを Heroku で動作させてみよう
- Web アプリは世界中からアクセスできるようになる
- Web アプリを Heroku（などのアプリケーションサーバ）に設置することを配備（Deploy）と言う

■コマンドラインで Heroku にログインする

- enPiT 環境には `heroku` コマンドをインストールしてある
- `heroku` コマンドを用いて，Heroku にログインできる
  - Heroku の ID と PW を入力する
- 以後の作業は Heroku コマンドを利用する

1. コマンド

```
heroku login
```

■heroku に SSH の公開鍵を設定する

- Heroku も git のリモートリポジトリである
- これを公開鍵でアクセスできるようにする

1. コマンド

```
heroku keys:add
```

2. 確認

```
heroku keys
```

■Sinatra アプリを Heroku で動かせるようにする

- Sinatra アプリを Heroku で動作させるには，（少ないものの）追加の設定が必要

1. 内容
   - 次スライドを見ながら，エディタを用いて，新たに次の 2 つのファイルを作成する

ファイル名	内容
config.ru	Web アプリサーバ（Rack）の設定
Gemfile	アプリで利用するライブラリ（Gem）

■追加するコード

1. コード: **config.ru**

```
require './hello'
run Sinatra::Application
```

2. コード: **Gemfile**

```
source 'https://rubygems.org'
gem 'sinatra'
```

■関連する Gem のインストール

- Gemfile の中身に基づき，必要な Gem（ライブラリ）をダウンロードする
  - Gemfile.lock というファイルができる
  - このファイルも commit の対象に含める

1. コマンド

```
bundle install
```

■アプリを GitHub に push する

- Heroku で動かす前に，commit が必要
  - 後に Heroku のリポジトリに対して push する
- ここでは，まず，GitHub にコードを push しておく
  - この場合の push 先は origin master

1. コマンド

```
git add .
git commit -m 'Add configuration files for
 ↪ Heroku'
git push -u origin master
```

■Heroku にアプリを作る

1. コマンド

```
heroku create
git remote -v # 確認用
```

   - 1 行目: Heroku が自動生成した URL が表示されるので，メモする
   - 2 行目: heroku という名前の remote が追加されたことが分かる
   - Web ブラウザで Heroku の管理画面を開き，アプリができていることを確認する

■Heroku にアプリを配備する

- Heroku にアプリを配備するには，Heroku を宛先としてリモートリポジトリに push する

1. コマンド

```
git push heroku master
```

   - Web ブラウザでアプリの URL（heroku create の際にメモしたもの）を開き，動作を確認する

3.1.3 演習課題

■演習課題 4-1

1. Sinatra アプリの作成
   - Sinatra アプリを作成して，Heroku で動作させなさい
   - Sinatra の DSL について調べ，機能を追加しなさい
   - コミットのログは詳細に記述し，どんな作業を行ったかが他の人にも分かるようにしなさい
   - 完成したコードは GitHub にも push しなさい

■演習課題 4-2 (1)

1. Sinatra アプリの共同開発
   - グループメンバーで Sinatra アプリを開発しなさい
   - 代表者が GitHub のリポジトリを作成し他のメンバーを Collaborators に追加する
     - 他のメンバーは代表者のリポジトリを clone する
   - どんな機能をもたせるかをチームで相談しなさい
     - メンバーのスキルに合わせて，できるだけ簡単なもの（DB は使わない）

■演習課題 4-2 (2)

1. Sinatra アプリの共同開発（続き）
   - 慣れてきたら GitHub Flow をチームで回すことを目指す
     - ブランチを作成し，Pull Request を送る
     - 他のメンバー（一人以上）からレビューを受けたら各自でマージ
   - GitHub の URL と Heroku の URL を提出
     - http://goo.gl/forms/p1SXNT2grM

4 Part 3: Ruby on Rails/Heroku

4.1 第 5 章 Ruby on Rails アプリの開発

4.1.1 Ruby on Rails アプリの生成と実行

■RoR を使った Web アプリケーション

1. Ruby on Rails（RoR）とは？
   - Web アプリケーションを作成するためのフレームワーク
2. 参考文献
   - Ruby on Rails

■rails_enpit アプリを作成する

- rails は予め，仮想化環境にインストールしてある
- rails new コマンドを用いて，RoR アプリの雛形を作成する

– コマンドは次スライド

■rails_enpit を作成するコマンド

```
rails new ~/rails_enpit --database=postgresql
cd ~/rails_enpit
git init
git create
git add .
git commit -m 'Generate a new rails app'
git push -u origin master
```

■Gemfile に JS 用 Gem の設定

- Gemfile に Rails 内部で動作する JavaScript の実行環境を設定する
  - 当該箇所のコメントを外す

1. 変更前

```
# gem 'therubyracer',  platforms: :ruby
```

2. 変更後

```
gem 'therubyracer',  platforms: :ruby
```

■Bundle install の実行

- Gemfile を読み込み，必要な gem をインストールする
  - `rails new` をした際にも，`bundle install` は実行されている
  - `therubyracer` と，それが依存している gem でまだインストールしていないものをインストール

1. コマンド

```
git commit -a -m 'Run bundle install'
```

■Gemfile 設定変更のコミット

- ここまでの内容をコミットしておこう

1. コマンド

```
git add .
git commit -m 'Edit Gemfile to enable the
 ↪ rubyracer gem'
git push -u origin master
```

■データベースの作成

- rails_enpit アプリの動作に必要な DB を作成する
- Database は Heroku で標準の PostgreSQL を使用する
  - RoR 標準の sqlite は使わない
- enPiT 仮想環境には PostgreSQL インストール済み

■PostgreSQL に DB を作成

1. 開発で利用する DB

rails_enpit_development	開発作業中に利用
rails_enpit_test	テスト用に利用
(rails_enpit_production)	（本番環境用）

- 本番環境用 DB は **Heroku** でのみ用いる

2. コマンド

```
createdb rails_enpit_development
createdb rails_enpit_test
```

■PostgereSQL クライアントのコマンド

1. クライアントの起動
   - `psql` コマンドでクライアントが起動
2. psql クライアンで利用できるコマンド

Backslash コマンド	説明
l	DB の一覧
c	DB に接続
d	リレーションの一覧
q	終了

■Rails server の起動

- この段階で，アプリケーションを起動できるようになっている
- Host OS の Web ブラウザで，`http://localhost:3000` にアクセスして確認
- 端末にもログが表示される
- 確認したら，端末で Ctrl-C を押してサーバを停止する

1. コマンド

```
bin/rails server -b 0.0.0.0
```

4.1.2 Controller/View の作成

■Hello World を表示する Controller

- HTTP のリクエストを処理し，View に引き渡す
  - MVC 構造でいう Controller である
- `rails generate controller` コマンドで作成する

1. コマンド

```
bin/rails generate controller welcome
```

■生成された Controller コードの確認

- git diff コマンドでどのようなコードができたか確認

```
git diff
```

- Controller のコードを作成した作業をコミット

```
git add .
git commit -m 'Generate the welcome
 ↪ controller'
```

■Hello World を表示する View

- HTML 等で結果をレンダリングして表示する
  - erb で作成するのが一般的で，内部で Ruby コードを動作させることができる
- `app/views/welcome/index.html.erb` を（手動で）作成する
  - コードは次スライド

■Hello World を表示する View のコード

1. **index.html.erb**

```
<h2>Hello World</h2>
<p>
  The time is now: <%= Time.now %>
</p>
```

■生成された View コードの確認

- git diff コマンドで変更内容を確認

```
git diff
```

- View のコードを作成した作業をコミット

```
git add .
git commit -m 'Add the welcome view'
```

■root となる route の設定

- Route とは？
  - HTTP のリクエスト（URL）とコントローラを紐付ける設定
- ここでは root へのリクエスト（GET /）を welcome コントローラの index メソッドに紐付ける

1. config/routes.rb の当該箇所をアンコメント

```
root 'welcome#index'
```

   - rake routes コマンドで確認できる

■routes.rb の設定変更の確認

- git diff コマンドで変更内容を確認

```
git diff
```

- routes.rb を変更した作業をコミット

```
git add .
git commit -m 'Edit routes.rb for the root
↪ controller'
```

■Controller と View の動作確認

- 再度，rails server でアプリを起動し，動作を確認しよう
- Web ブラウザで http://localhost:3000/ を開く

1. コマンド

```
bin/rails server -b 0.0.0.0
```

   - git log コマンドで一連の作業を確認してみると良い

■GitHub への Push

- ここまでの作業で，controller と view を 1 つ備える RoR アプリができた
- 作業が一区切りしたので，GitHub への push もしておく

1. コマンド

```
git push
```

4.1.3 Heroku にディプロイする

■RoR を Heroku で動かす

- 作成しと RoR アプリを Heroku で動作させよう

1. Getting Started
   - Getting Started with Rails 4.x on Heroku

■Heroku 用設定を Gemfile に追加

- Gemfile に rails_12factor を追加する
- Ruby のバージョンも指定しておく
- Gemfile を変更したら必ず bundle install すること

1. Gemfile に追加する内容

```
gem 'rails_12factor', group: :production
ruby '2.2.2'
```

■デプロイ前に Git にコミット

- Heroku にコードを送るには，git を用いる
- 従って，最新版を commit しておく必要がある
- ここでは，commit 後，まずは GitHub にも push しておく

1. コマンド

```
git commit -a -m 'Set up for Heroku'
git push
```

   - 2 行目: push する先は origin（=GitHub）である

■Heroku アプリの作成とディプロイ

- heroku コマンドを利用してアプリを作成する

1. コマンド

```
heroku create
git push heroku master
```

   - 1 行目: heroku create で表示された URL を開く
   - 2 行目: git push は heroku の master を指定．ディプロイすると，Heroku からのログが流れてくる

4.1.4 演習課題

■演習課題 6

1. RoR アプリの作成
   - ここまでの説明に従い，Heroku で動作する RoR アプリ（rails_enpit）を完成させなさい

4.2 第 6 章 DB を使うアプリの開発と継続的統合

4.2.1 DB と Scaffold の作成

■Scaffold

1. Scaffold とは
   - scaffold - Google 検索
2. RoR では，MVC の雛形のこと
   - CRUD 処理が全て自動で実装される

■Scaffold の生成方法

1. コマンド

```
git checkout -b books
```

```
bin/rails generate scaffold book
↪ title:string author:string
```

- 多くのコードが自動生成されるので，branch を切っておくと良い
  - 動作が確認できたら branch をマージ
  - うまく行かなかったら branch ごと削除すれば良い

■route の確認

1. route
   - Scaffold の生成で変更されたルーティングの設定を確認
2. コマンド

```
bin/rake routes
```

■DB の Migrate

1. migrate とは
   - Database のスキーマ定義の更新
   - Scaffold を追加したり，属性を追加したりした際に行う
2. コマンド

```
bin/rake db:migrate
```

■参考：Migrate の取り消しの方法

- DB の migration を取り消したいときは次のコマンドで取り消せる

```
bin/rake db:rollback
```

- 再度，migrate すれば再実行される

```
bin/rake db:migrate
```

■参考：Scaffold 作成の取り消しの方法

1. コマンド

```
git add .
git commit -m 'Cancel'
git checkout master
git branch -D books
```

   - 1 ～ 2 行目：自動生成された Scaffold のコードを branch に一旦コミット
   - 3 行目：master ブランチに移動（）
   - 4 行目：branch を削除（`-D` オプション使用）

■動作確認

1. 動作確認の方法
   - Web ブラウザで `http://localhost:3000/books` を開く
   - CRUD 処理が完成していることを確かめる
2. コマンド

```
bin/rails server
```

■完成したコードをマージ

1. ブランチをマージ
   - 動作確認できたので，books branch をマージする
   - 不要になったブランチは，`git branch -d` で削除する
2. コマンド

```
git add .
git commit -m 'Generate books scaffold'
git checkout master
git merge books
git branch -d books
```

■Heroku にディプロイ

1. ディプロイ
   - ここまでのアプリをディプロイする
   - heroku にある db も migrate する
   - Web ブラウザで動作確認する
2. コマンド

```
git push heroku master
heroku run rake db:migrate
```

4.2.2 RoR アプリのテスト

■テストについて

1. ガイド
   - A Guide to Testing Rails Applications —Ruby on Rails Guides

■テストの実行

1. テストコード
   - Scaffold はテストコードも作成してくれる
   - テスト用の DB（`rails_enpit_test` ）が更新される
2. コマンド

```
bin/rake test
```

4.2.3 Travis CI との連携

■Travis CI のアカウント作成

1. アカウントの作り方
   - 次のページにアクセスし，画面右上の「Sign in with GitHub」のボタンを押す
     - Travis CI - Free Hosted Continuous Integration Platform for the Open Source Community
   - GitHub の認証ページが出るので，画面下部にある緑のボタンを押す
   - Travis CI から確認のメールが来るので，確認する
2. Ruby アプリ Travis CI: Building a Ruby Project

■Travis の初期化

1. 内容
   - Travis の CI ツール
     - travis-ci/travis.rb
   - Travis にログインして初期化を行う
   - `init` すると `.travis.yml` ができる
2. コマンド

```
gem install travis      # Travis CLI のアップデート
travis login --auto     # GitHub のログイン情報で自動ログイン
travis init             # 質問には全て Enter を押す
```

■Heroku との連携

1. Heroku との連携
   - Travis CI から Heroku への接続を設定する
     - Travis CI: Heroku Deployment
2. コマンド

```
travis setup heroku
```

■Travis で動かす Ruby のバージョン設定

1. 設定ファイルの変更
   - まず、Ruby のバージョンを指定する
   - 変更の際は YAML のインデントに注意する
2. .travis.yml を書き換える

```
language: ruby
rvm:
- 2.2.2
```

■Travis 用 DB 設定ファイル

1. Travis でのテスト DB
   - テスト DB 用の設定ファイルを追加する
2. `config/database.yml.travis`

```
test:
  adapter: postgresql
  database: travis_ci_test
  username: postgres
```

■Travis 上の DB 設定

1. 設定ファイルの変更（追加）
   - PostgreSQL のバージョン
   - DB の作成
   - Travis CI: Using PostgreSQl on Travis CI
2. .travis.yml（抜粋）

```
addons:
  postgresql: "9.3"
before_script:
  - psql -c 'create database travis_ci_test;' -U postgres
  - cp config/database.yml.travis config/database.yml
  - rake db:migrate RAILS_ENV=test # いらない？
```

■GitHub と Travis CI 連携

1. 説明
   - ここまでの設定で，GitHub に push されたコードは，Travis CI でテストされ，テストが通ったコミットが Heroku に送られるようになった
   - Web ブラウザで Travis CI を開いて確認する
2. コマンド

```
git add .
git commit -m 'Configure Travis CI'
git push
```

■Travis 経由での Heroku への deploy

1. Travis のログを閲覧
   - Web ブラウザで Travis CI の画面を開く
   - ログを読む
2. Heroku への Deploy
   - テストが通れば，自動で Heroku に配備される
   - 配備できたら Web ブラウザでアプリのページを開いて確認する

4.2.4 演習課題

■演習課題 7-1

1. `rails_enpit` の拡張
   - View を変更
     - welcome コントローラの view から，books コントローラの view へのリンクを追加する etc
   - Scaffold の追加
     - 任意の Scaffold を追加してみなさい
     - DB の migration を行い，動作確認しなさい
   - Heroku への配備
     - Travis 経由で Heroku へ deploy できるようにする

5 Part 4: Web API

5.1 第 7 章楽天 API を利用したアプリケーション

5.1.1 楽天 API

■楽天 API とは？

1. ご利用ガイド
   - 楽天ウェブサービス: ご利用ガイド
2. 楽天 API SDK
   - rakuten-ws/rws-ruby-sdk

■楽天 API サンプルアプリ

- ychubachi/rakuten_enpit_example
  - `git clone` する
  - `bundle install` する
- Heroku でアプリを作りアプリ URL を取得
  - `heroku create` する

■アプリ ID の発行

- 新規アプリを登録する
  - 楽天ウェブサービス: 新規アプリ登録
- アプリ名（任意），アプリの URL，認証コードを入力

- アプリ ID，アフィリエイト ID 等を控えておく

■環境変数の設定

- アプリ ID（APPID）とアフィリエイト ID（AFID）を環境変数に登録
- ~/.bash_profile に次の行を追加（自分の ID 等に書き換えること）
- exit して，再度 vagrant ssh

```
export APPID=102266705971259xxxx
export
 ↪  AFID=11b23d92.8f6b6ff4.11b23d93.???????
```

■ローカルでの動作確認

- ローカルで動作確認する

```
ruby hello.rb -o 0.0.0.0
```

5.1.2 Heroku で動作させる

■Heroku の環境変数

1. 環境変数の作成
   - 次のコマンドで，Heroku 内部にも環境変数を作る
2. コマンド

```
heroku config:set
 ↪  APPID=102266705971259xxxx
heroku config:set
 ↪  AFID=11b23d92.8f6b6ff4.11b23d93.???????
```

   - Configuration and Config Vars | Heroku Dev Center

■Heroku での動作確認

- Heroku に直接 Push してみる

1. コマンド

```
git push heroku master
```

   - web ブラウザで動作確認

5.1.3 Travis CI 連携

■楽天 API サンプルアプリの Travis CI 連携

- 楽天 API サンプルアプリを修正して Travis CI と連携させよう
- この作業を行うために，ychubachi が所有する rakuten_enpit_example を フォークする

■フォーク（fork）とは？

- 他人の GitHub のリポジトリを複製して自分で更新できるようにすること
  - 他人の GitHub には push できない
- フォークをすると，他人が作成したソースコードを修正できる
  - フォーク（複製）した自分のリポジトリに push 可能
- フォーク元のリポジトリに対して Pull request を送ることができる
  - フォーク元の持ち主がマージするともとともとの master に反映される
- まさに **OSS 流開発**のスタイル！

■楽天 API サンプルのフォーク

1. コマンド

```
git fork
git remote -vv
```

   - 1 行目: コマンドラインでフォークを作成
     - GitHub の Web で作成することもできる
   - 2 行目: remote に自分の GitHub ユーザ名がついた宛先が追加されている
     - Web でフォークを作成した場合と挙動が異なるので注意

■.travis.yml の再生成

- 下の $GITHUB_NAME は自分の名前に置換して実行

```
rm .travis.yml
travis init -r
 ↪  $GITHUB_NAME/rakuten_enpit_example
travis setup heroku
emacs .travis.yml # 任意のエディタで
```

- 1 ～ 2 行目: travis.yml の削除と新規作成
- 3 行目: Heroku 用の追加設定
- 4 行目: 利用する Ruby のバージョンを指定

■Travis CI の環境変数

- Travis CI にも環境変数を設定
- 自分の APPID，AFID に書き換えること

1. コマンド

```
travis env set APPID 102266705971259xxxx
travis env set AFID
 ↪  11b23d92.8f6b6ff4.11b23d93.???????
```

■GitHub に push して Travis CI を走らせる

- 変更をコミットして 自分のリポジトリに push

1. コマンド

```
git add .
git commit -m 'Update .travis.yml'
git push -u $GITHUB_NAME master
```

   - しばらくすると Travis CI が動き出すので，確認する

5.1.4 演習課題

■演習課題 8-1

1. ローカルでサンプルを動かす
   - 自分の APPID を作成する
   - 仮想化環境と Heroku の環境変数を設定
   - ローカルで動かしてみよう
   - Heroku に直接 Push して動かしてみよう

■演習課題 8-2

1. Travis 経由で動かす
   - サンプルを Travis 経由で動作させてみよう
     - Fork して，自分のリポジトリに push できるようにする
     - `.travis.yml` の設定を変更する
       * やり方は各自で考えてみよう
     - Travis CI に環境変数を設定する

6 Part 5: Mini Project

6.1 第 8 章ミニプロジェクト

6.1.1 演習課題

■ミニプロジェクト演習

- 楽天 API を利用した Web アプリの **コラボレイティブ開発**
  - 概ね 1 時間ごとに最新版を「デモ」する超短距離スプリント
- アプリそのものの完成度は問わない
  -「GitHub フロー」をうまく回せるかどうか
- 注意事項
  - 難しい知識を無理して使わない
  - 自分がチームに貢献できることを自分で発見しよう

■課題の提出先

- グループの代表者はアプリの URL 等を次のフォームから提出してください
  - http://goo.gl/forms/xdeirTA169
- その他
  - README.md に使い方，GitHub/Heroku の URL などを書く
  - LICENCE は必ず設定する
  - コミットメッセージやブランチ名は分かりやすく

6.1.2 おわりに

■Thank you

- and enjoy **Scrum!**

7 補足資料

7.1 補足資料

7.1.1 Vagrant 関連

■Vagrant の補足

1. 仮想環境とのファイル共有
   - Guest OS 内に `/vagrant` という共有フォルダがある
   - このフォルダは Host OS からアクセスできる
   - 場所は Vagrantfile があるフォルダ
   - 共有したいファイル（画像など）をここに置く

7.1.2 Git 関連

■Git の補足

1. 元いた branch に素早く戻る方法

```
git checkout other_branch # master で
# 編集作業と commit
git checkout - # master に戻る
```

2. Git brame
   - だれがどの作業をしたかわかる（誰がバグを仕込んだのかも）
     - Using git blame to trace changes in a file ·GitHub Help

■バイナリのコンフリクト (1)

- git merge でバイナリファイルがコンフリクトした場合、ファイルは git merge 実行前のままとなります[*1]。
- 以下を実行してコンフリクトが発生したとします。

```
git checkout master
git merge branch_aaa
```

■バイナリのコンフリクト (2)

- そのままにしたいとき (=master を採用）は

```
git checkout --ours <binaryfile> # 明示的な実行
↪ は不要
git add <binaryfile>
git commit
```

- branch_aaa のファイルを採用したいときは

```
git checkout --theirs <binaryfile>
git add <binaryfile>
git commit
```

■Hub コマンドについて

- enPiT 環境には Hub コマンドが仕込んである
  - github/hub
- 通常の Git の機能に加えて，GitHub 用のコマンドが利用できる
  - コマンド名は「git」のまま（エイリアス設定済み）
- 確認方法

```
git version
alias git
```

7.1.3 GitHub 関連

■GitHub の補足 (1)

1. Issue
   - 課題管理（ITS: Issue Tracking System）
   - コミットのメッセージで close できる
     - Closing issues via commit messages ·GitHub Help
2. Wiki
   - GitHub のリポジトリに Wiki を作る
     - About GitHub Wikis ·GitHub Help

■GitHub の補足 (2)

1. GitHub Pages
   - 特殊なブランチを作成すると，Web ページが構築できる
     - GitHub Pages

[*1] git merge でバイナリファイルがコンフリクトした場合·Issue #6

7.1.4 Heroku 関連

■Heroku の補足

1. Heroku のアプリの URL 確認

```
heroku apps:info
```

2. Heroku のログをリアルタイムで見る

```
heroku logs --tail
```

3. `rails generate` などが動かない

```
spring stop
```

7.1.5 Travis CI 関連

■Travis CI の補足

1. Status Image
   - README.md を編集し，Travis のテスト状況を表示する Status Image を追加する
   - Travis CI: Status Images
2. Deploy 後、自動で heroku の db:migrate
   - 次の URL の「Running-commands」の箇所を参照
     - Heroku Deployment - Travis CI

■Sinatra でテストを実行可能に

- `Gemfile` に `rake` を追加する

```
gem 'rake'
```

- `Rakefile` を作成する

```
task :default => :test

require 'rake/testtask'

Rake::TestTask.new do |t|
  t.pattern = "./*_test.rb"
end
```