課題B

- ウェブサイトを確認のこと
  https://www.se.cs.titech.ac.jp/lecture/exp4/first.html

タスク管理アプリケーション?

- **Web アプリとしても多く実現されている.**
  - check*pad
    http://www.checkpad.jp/
  - Remember the milk
    http://www.rememberthemilk.com/
  - Trello
    https://trello.com/

開発ステップ

1. **学習フェーズ**
   - Play! でサーバを起動してみよう.
   - Play! の動作法を学ぼう.
   - コントローラ, ビュー, モデルを記述してみよう.
2. **実践フェーズ**
   - 画面遷移の設計
   - モデルの設計
   - 実装
3. **JavaScript による拡張フェーズ（しばらく後で）**

開発キットのダウンロード

- **開発キットを以下からダウンロードする.**
  - https://www.se.cs.titech.ac.jp/lecture/exp4/B02-exp4todo.zip
- **HOME 直下へ展開する.**
  - 以下の例では，~/exp4todo に展開している.
    - `$ cd`
    - `$ unzip B02-exp4todo.zip`
    - `$ cd exp4todo`
  - `~/exp4todo` ディレクトリ中にファイルが展開されていることを確認する.

Play!

- **オープンソースで開発されているJava製のWebアプリケーションフレームワーク.**
  http://www.playframework.com/
  - 最新バージョンは 2 系列（2.6.6）
- **利用するバージョン: 1.5.0**
  - CSC では /usr/local/share/java/play-1.5.0 以下に配置されている.（以下, このパスを $PLAY_PATH とする. ）
  - 自宅等に環境を用意する場合, play-1.5.0.zip をダウンロードして適当な場所に展開すること.
    - https://www.playframework.com/download#older-versions
    - その際, 後述する PLAY_PATH を適切に設定しなおすこと.
  - ドキュメントの日本語訳は以下のサイトで確認できる.
    - https://www.playframework.com/documentation/ja/1.2.x/home
    - 一部英語が残っているので注意すること.
  - Javadoc
    - https://www.playframework.com/documentation/1.4.x/api/

環境設定

- ~/.bash_profile 等に以下を記述しておく.
  - export PLAY_PATH=/usr/local/share/java/play-1.5.0
    - 下記の PATH の定義より上に記述すること.
  - export PATH=$PLAY_PATH:$PATH
    - すでに expert PATH= という記述がある場合，上記の赤字部分を追加すること.
    - このようにすることにより，play コマンドにパスが通る.
    - 以下を実行し，コマンドを発見できることを確認すること.

```
$ which play
/usr/local/share/java/play-1.5.0/play
```

Play! サーバの起動

- **サーバの起動・停止は以下のコマンドで行う.**
  - **`$ cd ~/exp4todo`　　`# (実験キットディレクトリへ移動)`**
  - **`$ play run`**
    - サーバが立ち上がる．ログがコンソールに出力される.
    - http://localhost:9001/ でアクセス可能である.
      - Play! の標準ポートは 9000 だが，環境の都合でポートを変更している.
    - Ctrl+C を押下することによりサーバを停止できる.
  - **`$ play start`**
    - サーバがバックグラウンドで立ち上がる．コマンドの実行は直ちに終了する.
  - **`$ play stop`**
    - バックグラウンドで立ち上がったサーバを終了させる.
- **サーバ起動の前にコンパイルを自力で行う必要はない.**
  - *.java ファイルのコンパイルは Play! がバックグラウンドで自動的に行う.
  - サーバ起動後も, ファイルを編集しブラウザをリロードすれば裏で自動的にコンパイルが行われる.
- **その他の play コマンドを知りたい場合：**
  → **`$ play help`** の結果を読んでみよう.

開発キットのディレクトリ構造

(Play! アプリのディレクトリ構造)

- **app/: アプリケーションの中心（主な編集対象）**
  - models/: モデル（*.java ファイル）
    - データ及びその処理を担う.
  - views/: ビュー（*.html ファイル）
    - モデルデータを加工しユーザに表示する処理を担う.
  - controllers/: コントローラ（*.java ファイル）
    - アプリケーションの制御を担う.
- **conf/: 設定ファイル**
  - application.conf: アプリケーションの設定を行う.
  - routes: URLとコントローラの対応付けを記述する.
- **logs/: ログファイル**
  - system.out: play start 後，サーバの標準出力が追記されていく.
    - $ tail -f system.out で垂れ流しにすることができる.
- **public/: 静的ファイル置き場**
  - (routes の記述により) http://localhost:9001/public/... で参照できる.

Play! 版アンケートフォーム

- **play サーバ起動後，以下にアクセスすることにより閲覧できる.**
  - http://localhost:9001/sample.questionnaire/
- **以下の機能を持つ.**
  - アンケートデータの送信機能
    - 登録されたアンケートをデータベースに保存する.
  - アンケートデータの閲覧機能
    - 全体，男性のみ，女性のみの区分で閲覧できる.
- **対応するプログラムは以下に存在する.**
  - app/controllers/sample/Questionnaire.java
  - app/models/sample/SampleEntry.java
  - app/views/sample/Questionnaire/*.html

conf/routes

- **どのURLに対してどのアクションメソッドを実行するかの対応関係を記述する設定ファイルである.**
  - リクエストのURLに対して，上から順に行をチェックする.
  - 最初にマッチした行のアクションメソッドを起動する.

```
……
GET /favicon.ico            ...

GET /public/                staticDir:public

GET /                       Application.index
*   /{action}               Application.{action}

*   /{controller}/          {controller}.index
*   /{controller}/{action} {controller}.{action}
……
```

http://localhost:9001/ へのアクセス
→ controller.Application.index を起動.

http://localhost:9001/foo/bar へのアクセス
→ controller.Foo.bar を起動.

詳細: https://www.playframework.com/documentation/ja/1.2.x/routes

コントローラ

- **app/controllers 以下に配置されたクラス群**
  - play.mvc.Controller を継承して作成する.
- **アクションメソッドをコントローラに実装**
  - アクションメソッドとは，コントローラに実装された public static void なメソッドであり，HTTPアクセスの際に Play! によって実行される.
  - routes での指定先となる.
  - 主な流れ：必要な処理を行った後，render メソッドを呼び出し，レスポンスのHTMLを構築する.
    - 主にアクションメソッド名に対応したテンプレートからHTMLを構築する.
    - 例えば，controller.sample.Questionnaire.index からは views/sample/Questionnaire/index.html のテンプレートを利用する.

```
public class Questionnaire extends Controller {
  public static void index() {
    ...
    // sample/Questionnaire/index.html に記述されたビューを描画
    render();
  }
...
```

詳細: https://www.playframework.com/documentation/ja/1.2.x/controllers

コントローラその他

- **コントローラからアクセスできるオブジェクト群（抜粋）**
  - params: リクエスト中の（フォームやURLの）パラメーター式
    - params.get("key") で該当キーの文字列を獲得できる.
  - session: セッションデータ
    - session.put("key", "value") で格納したデータが session.get("key") で獲得できる.
  - request: HTTPリクエスト
  - response: HTTPレスポンス
- **リダイレクト**
  - ある URL から別の URL へジャンプする処理.
  - アクションメソッド中で他のアクションメソッドを呼び出すことにより, Play!が自動的に行う.

```
public class Questionnaire extends Controller {

    public static void postGender() {
      ...
      // sample.questionnaire/nameform にリダイレクトする
      nameForm();
    }
...
```

コントローラに関する注意

- **アクションメソッドはオーバーロードできません.**
  - アクションメソッドの引数は，実際には HTTP のパラメータとして扱われるため.
  - 引数の種類の異なるアクションメソッドを用意するのではなく，内部で分岐すること.

- **アクションメソッドからアクションメソッドの「呼び出し」はできません.**
  - リダイレクトとして扱われるため.
  - コンパイラに手を入れて挙動を変えていることに注意.

```
public class Questionnaire extends Controller {

    public static void postGender() {
      ...
      // sample.questionnaire/nameform にリダイレクトする
      nameForm();
    }
...
```

ビュー

- HTML に特殊構文が埋め込まれたテンプレートを用いる.

```
#{extends 'sample/Questionnaire/main.html'/}
<h2>確認</h2>
<p>あなたの入力は以下の通りです. </p>
<dl>
  <dt>性別</dt>
  <dd>${session.gender}</dd>
  <dt>名前</dt>
  <dd>${session.name}</dd>
  <dt>感想</dt>
  <dd>${session.comment.nl2br()}</dd>
</dl>

<form action="@{submit()}" method="post">
  <input type="submit" />
</form>
```

```
<!DOCTYPE ...>
<html ...>
<body>
<h1>アンケート</h1>
<h2>確認</h2>
<p>あなたの入力は以下の通りです. </p>
<dl>
  <dt>性別</dt>
  <dd>男性</dd>
  <dt>名前</dt>
  <dd>飼育員</dd>
  <dt>感想</dt>
  <dd>背中がむれちまって妙にかゆい。</dd>
</dl>
<form action="submit" method="post">
  <input type="submit" />
</form>
</body>
</html>
```

詳細: https://www.playframework.com/documentation/ja/1.2.x/templates

ビューの構文（抜粋）

- **テンプレート**
  - `${...}` 内側を実行し，結果を展開する.
  - `@{...}` 該当のアクションに対応するURLを生成する.
    - 例: `<a href="@{action()}">リンク</a>`
    - `@@{...}` で展開すると絶対URL になる.
  - `*{...}*` コメント．内容は無視される.
  - `#{...}` テンプレートデコレータ，タグ
    - `#{if 条件} ... #{/if} #{else} ... #{/else}` 条件分岐
    - `#{list items:リスト変数, as:'変数名'} ... #{/list}` ループ
    - `#{extends 親HTML /}` テンプレートの包含関係を示す.
      - 親に `#{doLayout /}` を記述しておくと，子の生成結果が埋め込まれる.
    - `#{set key:'value' /}` のあとに `#{get 'key' /}`
      テンプレート間のデータをやりとりできる．setで渡した値をgetで参照できる.
  - `%{...}%` スクリプトを直接記述できる.
- **{} 内部には Groovy の式が記述可能である.**
- **展開結果は自動的にHTMLエスケープ（< → < など）される.**
  - HTMLをそのまま出力したい：`${value.raw()}`
  - 改行を
 に変換したい：`${value.nl2br()}`

詳細: https://www.playframework.com/documentation/ja/1.2.x/javaextensions

モデル

- **作成方法**
  - models パッケージ下に作成する.
  - java.db.jpa.Model クラスを継承する.
  - javax.persistence.Entity アノテーションを付与する.
    - public の前の「@Entity」に相当.
    - データベース格納対象のデータであることを明示するもの.
  - フィールドにデータを格納する.
    - final にしてはならない.
    - 自動で対応するテーブルが作成される.
    - 基本的には，数値（int/long）や文字列（String）を用いる.
    - 他のモデルへの参照も格納できる.
- **Model クラス**
  - ユニークな ID が格納される属性 id を持つ.

```
@Entity
public class SampleEntry extends Model {
  public int gender;
  public String name;
  public String comment;

  public SampleEntry(int gender,
    String name, String comment) {
    this.gender = gender;
    this.name = name;
    this.comment = comment;
  }
}
```

詳細: https://www.playframework.com/documentation/ja/1.2.x/jpa

モデルオブジェクトの取得・保存

- **保存**

```
SampleEntry entry = new SampleEntry(gender, name, comment);
entry.save();
```

  - 背後でデータベースへの保存（INSERT文，UPDATE文）が実行される．

- **取得**

```
// 全エントリを取得
List<SampleEntry> entries = SampleEntry.all().fetch();
// 条件付きで取得
List<SampleEntry> males = SampleEntry.find("gender = ?1", 1).fetch();
// ひとつだけ取得．id は long 型であるため，1 ではなく 1L を指定する．
SampleEntry entry = SampleEntry.find("id = ?1", 1L).first();
```

  - 背後で行の取得（SELECT 文）が実行される．
  - 引数部分には，SQL の select 式が記述できる．
    - 直後に指定する値をどこに埋め込むかを ?1 (1番目), ?2 (2番目), ... で指定

モデルオブジェクトへのアクセス

- **複数の条件を使用する場合は，条件を AND 演算子で結合する.**
  - List<SampleEntry> entries =
    SampleEntry.find("gender = ?1 AND name = ?2", g, n).fetch();
- **全データの個数を取得する場合には，count メソッドを用いる.**
  - int size = SampleEntry.count(); // 全行数
  - int size = SampleEntry.count("gender = ?1", 1); // 男性の行数
- **ソートして取得したい場合には，ORDER BY 句を指定する.**
  - List<SampleEntry> es = Task.find("...条件... ORDER BY id ASC"); // idで降順
  - List<SampleEntry> es = Task.find("...条件... ORDER BY id DESC"); // id で昇順
- **個数を制限して取得したい場合は，from 等のメソッドを利用する.**
  - List<SampleEntry> es = Task.find("...", ...).from(2).fetch(5); // 3番目から5個取得
- **データを削除する場合には，delete メソッドを用いる.**
  - SampleEntry e = SampleEntry.find("...").first(); e.delete(); // 取得済み行を削除
  - int deleted = SampleEntry.delete("gender = ?", ...); // 条件を満たす全行を削除
- **データを更新する場合は，変更後に save メソッドを実行する.**
  - SampleEntry e = SampleEntry.find("id = ?1", 1L);
  - e.name = "hayashi"; e.save(); // id が1番のデータの名前を hayashi に変更

データベース上でのデータの確認

- **データベース上でのデータの確認方法**
  - http://localhost:9001/@db にアクセスする.
  - JDBC URL に「jdbc:h2:mem:play」を指定し，「接続」
  - 左側のテーブル名（モデルのクラス名に対応している）をクリックし，SQL文がペースとされたのを確認後，「実行」
- **データベースはメモリ上に構築されている.**
  - 実験キットでは，データはメモリ上に保存されるよう設定されている.
  - サーバを停止させると，データベースの内容は初期化される.
  - 開発初期の試行錯誤には都合が良いが，後段では面倒が生じる.
- **データのハードディスクへの保存**
  - conf/application.conf 内の「db=mem」という行を「db=fs」に変更すると，ハードディスクに保存される.
    - 保存先は $PLAY_HOME/db/h2/play.* になる.
      - 初期化したいときは，これらを削除してサーバを再起動すればよい.
  - データ確認時のJDBC URLは「 jdbc:h2:~/exp4todo/db/h2/play」とする.

Play! チートシート集

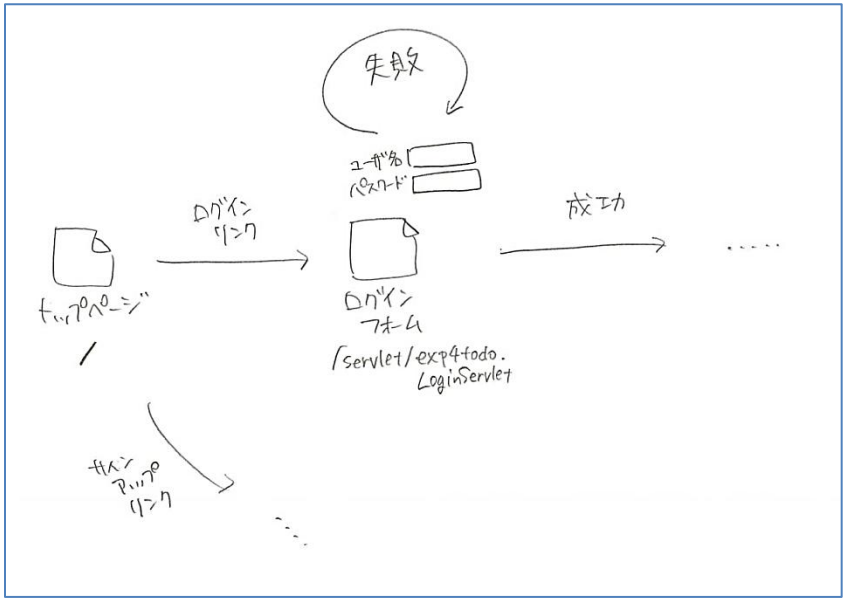
- **コマンドライン**
  - https://www.playframework.com/documentation/ja/1.2.x/cheatsheet/commandLine
- **モデル**
  - https://www.playframework.com/documentation/ja/1.2.x/cheatsheet/model
- **コントローラ**
  - https://www.playframework.com/documentation/ja/1.2.x/cheatsheet/controllers
- **ビュー**
  - https://www.playframework.com/documentation/ja/1.2.x/cheatsheet/templates

開発ステップ

1. **学習フェーズ**
   - Play! でサーバを起動してみよう.
   - Play! の動作法を学ぼう.
   - コントローラ, ビュー, モデルを記述してみよう.
2. **実践フェーズ**
   - 画面遷移の設計
   - モデルの設計
   - 実装
3. **JavaScript による拡張フェーズ（しばらく後で）**

画面遷移設計

- **どんなページが必要?**
  - どんなリンクが必要?
  - どんなフォームが必要?

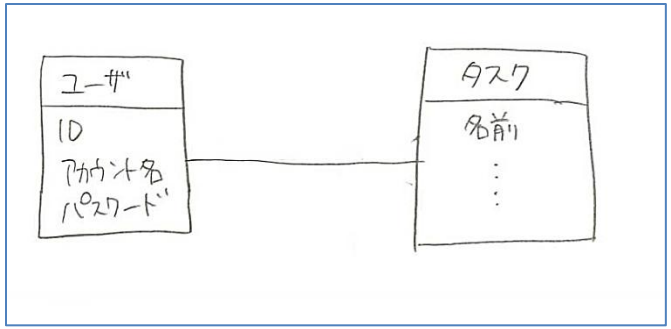


- **図形式で書き下してみよう.**
  - 節：ページ
  - 辺：ページ遷移
    - リンクをクリック
    - フォームを submit
  - エラーが起きたときはどうする?

- **プログラムを書く前に HTML として組んでみてもよい.**

データベース設計

- **どんなデータをデータベースに格納する?**
  - 基本的には数値や文字列に落とし込む必要がある.
    （それ以外のデータ型も扱える場合がある. ）
- **どのように配置する?**
  - 関連付いたひとつのまとまりとして扱えるものは同じオブジェクトに所属させるとよい.
- **データ間の連携**
  - フィールドに他のオブジェクトを持たせて関連付ける.
  - 1対多関係に注目するとよい.
    - 例: 「ひとりのユーザーは複数のタスクを持つ」→ タスクがユーザを持つ.
- **図形式で書き下してみよう.**
  - ER図など



機能を追加していこう

- **例えば……（この順序通りである必要はない．）**
  1. ユーザの新規登録（サインアップ）
  2. ユーザのログイン（サインイン）
  3. タスクの追加
  4. タスク一覧の表示
  5. タスクの完了処理
  6. タスク名の編集
  7. タスク数が多くなった場合の一覧表示の対処

Eclipse の利用方法

- **Play による Eclipse プロジェクトファイルを生成する.**
  - **`$ play eclipsify`** コマンドを実行する.
    - .classpath, .project 等の設定ファイルが生成される.
  - 上記プロジェクトを Eclipse 環境にインポートする.
    - File > import > Existing Project
    - ~/exp4todo ディレクトリを指定する.

- **ビューの HTML ファイルの扱い**
  - エディタを変更する.
    - Eclipse > 環境設定 > General > Editors > File Associations で「.html」を選択,「HTML (Play)」を選択し「default」をクリックする.
  - 文字コードの問題に注意すること.
    - Eclipse の HTML Editor は頭が悪く, UTF-8 で記述されたファイルの文字コードを正しく識別してくれない.
    - 各 .html テンプレートファイルの先頭に `*{<meta charset="utf-8">}*` というコメントを挿入すると正しく認識される.

レポートについて

- **画面例を載せましょう**
  - キャプチャ画像など（具体的なシナリオに基づいて）
  - すべての画面を撮る必要はない（典型的/重要な部分のみ）
- **画面遷移の設計がわかるように．例えば:**
  - 画面遷移図を書く.
  - どのように遷移するかを文章で記述する.
- **実装方針について，なぜそのようにしたのかの記述を考察として含めること**
  - 資料やサンプル等で「ヒント」として示している実装法についても同様.
- **ソースコードを添付**
  - 課題A同様（ant pack）