

参考サイト

公式サイト

[公式サイト](#) [\(Google Code\)](#) ダウンロードはコチラから
[ひがやすをblog](#) 開発者ひがやすをさんのブログ

- [環境構築とプロジェクト作成](#)
- [Slim3でGWT](#)
- [とりあえずSlim3アプリケーションを作ろう](#)

Controller作成

- 基本的に、リクエストを受け取るアクション1つに対して1つのControllerを割り当てる
 - Controllerは、`slim3.rootPackage`の直下に`controller`という名前のパッケージを作成し、その下に`XxxxController`という名前で作成する。build.xmlにある`gen-controller`ターゲットを利用すれば、パッケージ作成からクラス作成まで面倒見てくれる(後述)
 - `controller`以下のパッケージ階層と`XxxxController`の`Xxxx`の名前が、アクションのリクエストURLに対応する [参考\(URL Mapping\)](#)。

`slim3.rootPackage`に`jp.fujiyan.gae.example.slim3`を指定している場合

リクエストのURL	必要なController
/	<code>jp.fujiyan.gae.example.slim3.controller.IndexController</code>
/show	<code>jp.fujiyan.gae.example.slim3.controller.ShowController</code>
/sub/	<code>jp.fujiyan.gae.example.slim3.controller.sub.IndexController</code>
/sub/show	<code>jp.fujiyan.gae.example.slim3.controller.sub.ShowController</code>

- `gen-controller`ターゲットでControllerを自動作成する。
 1. build.xmlを開く(ダブルクリック等)。
 2. Outlineビューから`gen-controller`ターゲットを選択し右クリック [Run As] [Ant Build]を選択する
 3. [Ant Input Request]ダイアログの[Input a controller path.]に、作成したいアクションのリクエストURLを入力すると、リクエストURLに対応したパッケージとControllerが作成される。
 4. また、Controllerの遷移先となるJSPも作成される。作成直後はwarディレクトリ直下に作成されるが、外部にJSPが公開されるのも具合が悪いので、WEB-INFの下(WEB-INF/jsp等)に移動したほうがベター。その際は、Controller内のJSPのパスも移動先に修正する。そもそもGAE/JではJSPの直接参照が出来ないらしいので、war直下にあってもイイらしいです([炸裂!情熱の右フック!!](#))。

Service作成

- Slim3では、ユースケース1つに対して1つのServiceを割り当てる、としている。「ユースケース」が何ぞや、というものもあるけど、難しい話はとりあえずすっ飛ばして、個人的には「Slim3に依存するフロー制御(画面遷移の実装等)はController」「ロジックのフロー制御はService」という棲み分けにしておく。
 - Serviceは、`slim3.rootPackage`の直下に`service`という名前のパッケージを作成し、その下に`XxxxService`という名前で作成する。やっぱり自動作成可能(後述)。
- `gen-service`ターゲットでServiceを自動作成する。
 1. build.xmlを開く(ダブルクリック等)。
 2. Outlineビューから`gen-service`ターゲットを選択し右クリック [Run As] [Ant Build]を選択する
 3. [Ant Input Request]ダイアログの[Input a service name.]に、サブパッケージ名+Service名を入力すると、パッケージとServiceが作成される。

Model作成

- DatastoreのEntityをタイプセーフにしたもの。
 - Modelは、`slim3.rootPackage`の直下に`model`という名前のパッケージを作成し、その下任意の名前で作成する。例によって自動作成可能(後述)。
- `gen-model`ターゲットでModelを自動作成する。
 1. build.xmlを開く(ダブルクリック等)。

Outlineビューからgen-modelターゲットを選択し右クリック [Run As] [Ant Build]を選択する

- 2.
3. [Ant Input Request]ダイアログの[Input a model name.]に、サブパッケージ名+Model名を入力すると、パッケージとModelが作成される。

プロパティの定義

- インスタンス変数とgetter/setterを定義する。特にAnnotationは不要

```
private String name;

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}
```

- 指定可能な型の一覧は、[ここ](#)

リクエストのパラメータをControllerで取得する

- Slim3のControllerには、リクエストのパラメータを取得するための簡易メソッドが用意されている。
 - asString()
 - asShort()
 - asInteger()
 - asLong()
 - asFloat()
 - asDouble()
 - asBoolean()
 - asDate()
 - asKey()

Controllerでの処理結果をJSPで出力

- 基本的には、各スコープ(Application/Session/Request)のattributeに値を設定し、Slim3のJSP Functions(特にh)で出力する。ループや条件分岐などの制御にはJSTLを利用する。
- Slim3のControllerには、attributeに設定するための簡易メソッドが用意されている。
 - requestScope()
 - sessionScope()
 - applicationScope()
- Serviceでデータを準備し、ControllerでServiceが準備したデータを上記メソッドでattributeに設定し、JSP FunctionsやJSTLを使って出力するのが定石か
- Slim3のJSP Functionsのhは、Keyをパラメータとして用いる際に便利。引数にKeyのインスタンスを指定すれば、勝手にBase64にエンコードしてくれる。

```
bookListの要素がEntity(プロパティkeyを持つ)であるとき、下記のコードで、アンカーをクリックした際に、そのEntityのKeyをリクエストに渡す。
<c:forEach var="book" items="${bookList}">
<a href="/showBook?bookKey=${f:h(book.key)}">${f:h(book.name)}</a>
<hr/>
</c:forEach>
```

Controller側ではasKey()を使って、Keyに変換した状態で取得できる。
Key bookKey = asKey("bookKey");

認証

- GAE/Jの標準に準拠[Google App Engine](#)

前方一致検索

- String型プロパティの前方一致検索が可能
- Modelクラス(@Modelアノテーションを適用したクラス)を定義すると、ModelMetaクラスが自動的に定義される
 - Bookクラスを定義すると、BookMetaクラスが自動的に定義される。
- Bookクラスのnameプロパティを前方一致検索するには、下記のように記述する。

```
String value = "検索文字列";

BookMeta meta = new BookMeta();
StringAttributeMeta<Book> attrMeta = meta.name;
List<Book> list = Datastore.query(Book.class).filter(attrMeta.startsWith(value)).asList();
```

ModelRefのLazy Load

- ModelRefのsetKey()で参照先のKeyを設定すれば、getModel()でLoadされる。

```
ModelRef<Foo> fooRef = new ModelRef<Foo>(Foo.class);
```

とある場合、

```
fooRef.setKey(fooKey);
```

とすれば、

```
Foo foo = fooRef.getModel();
```

でfooKeyが指すFooをLoadする

ModelRefで問い合わせ

- ModelRefAttributeMetaを使えば、ModelRefで問い合わせが可能
- 例えば、部署(1)対従業員(多)の場合、

```
// 部署
@Model(schemaVersion = 1)
public class Dept implements Serializable {
}

// 従業員
@Model(schemaVersion = 1)
public class Employee implements Serializable {
    private ModelRef<Dept> deptRef = new ModelRef<Dept>(Dept.class);

    public ModelRef<Dept> getDeptRef() {
        return deptRef;
    }
}
```

とある場合に、「ある部署に所属する従業員全員」という問い合わせは下記のコードで可能。

```
Key deptKey = [ある部署のKey];
```

```
EmployeeMeta employeeMeta = new EmployeeMeta();
ModelRefAttributeMeta<Employee, ModelRef<Dept>, Dept> refMeta = employeeMeta.deptRef;
List<Employee> list = Datastore.query(Employee.class).filter(refMeta.equal(deptKey));
```

- これを使えば、親子関連を持たせたい場合に、entityGroupを使わなくても良い。

ページング

- DatastoreのCursorは、前にしか進むことができないため(多分)、「前ページ」とかの移動ができない、ましてや「nページ目」とかどうなの?
- あと、Slim3のInMemoryFilterとModelQueryのlimit()/offset()を合わせて実行した場合、
 - 先にlimit()/offset()が実行される
 - その結果に対してInMemoryFilterが実行される

という順番なので(多分)、limit(1000)としても、1000件帰ってこない場合がある。例えば、総数2000件あって、InMemoryFilterにマッチするEntityが1001件目以降にしか存在しない場合、limit(1000)としても0件となる(っぽい)。

- ということで、InMemoryFilterの結果に対して、任意のページに遷移可能なページ制御を考える。

- と、以前はここにソースがあったけど、どうも正常動作しないので取り下げ...
 - とりあえずの結論を[文字列の部分一致検索とページング](#)にて

InverseModelListRef(思いっきり想像なので、正確性は保証しません...)

- InverseModelListRefは永続化の対象外(@Attribute(persistent = false))
- InverseModelListRef#getModelList()によって、相手側Entityのクエリを実行する、プロキシ的な役割
- InverseModelListRef#getModelList()を呼び度にクエリが呼びだされ、新しいListのインスタンスが返されるので、それに相手側Entityを追加しても、データベースは変更されない

Slim3とFlex(BlazeDS)

- Slim3とBlazeDSの組み合わせでは、/messagebroker/amfへのリクエストが大量発生して、GAEの上限に引っかかってしまうとの報告あり

参考:[ワタクシゴト](#)

- 開発者のひがやすおサンも、BlazeDSの利用はあまり薦めておらず、HTTPServiceでやった方がラク、とおっしゃってます

参考:[ひがやすを blog](#)

- HTTPServiceのレスポンス形式については、下記2つの選択肢があるかと
 - XML
 - E4Xを使えばFlex側の処理はラク。でもXMLフォーマットはやや冗長ですかねえ
 - JSON
 - ということで、こっちをサーバー側から返すようにしてみよう
 - 利用ライブラリは[JSONIC](#)を使うことに [Flex](#)