Karatachi Project (仮)
たけうち（ちめら）
chimera@karatachi.org

Wicket as Meta-Framework

メタフレームワークとしてのWicket

自己紹介

- 流しの似非プログラマ兼NEET
  - アセンブラからアスペクト指向までが座右の銘
  - 言語オタクというよりはソフトウェア工学オタク
- 一応まだたぶん学生
  - システムのバグでforループから抜け出せません
  - アスペクト指向の研究室のはず
- ベンチャーもしてたりするかもしれない
  - 一緒に働いてくれる人募集中（主にWicketとかC#とか）
  - 仕事はたまに欲しいかも。。。

自分にとってのWicket

- 上から下まで非常に拡張性の高いフレームワーク
  - 高レベルなところではPanelによるコンポーネント化
  - 低レベルでは各種ListenerやBehaviorによる振る舞いの追加
  - もっと低レベルなところにも探せばいろいろ…

とりあえず拡張したいと思ったところにはだいたい拡張ポイントがある

- HTML要素の素直かつエレガントなオブジェクト表現
  - ほどよい粒度でのコンポーネント化
  - タグの階層構造をオブジェクトのコンポジションで表しているのが素敵ポイント

Talk Outline

Component Resolver

- HTMLタグからのコンポーネントインスタンスの自動生成について
- Teedaっぽいフィールドとプロパティの関連づけの設計と実装

Wicket with DI Container

- SeasarとWicketのつなぎこみをしたときに見つけたおもしろい拡張ポイントの紹介
- むしろ作ったS2Wicketもどきの宣伝

London Wicket Users' Group から Tips & Etc...

- Component Visitorによるコンポーネントの振る舞いの一括変更
- Borderの使い道の模索

Component Resolver

コンポーネントの自動解決とその応用

比較のために…
MarkupContainer::add(Component child)

- Wicketでページを作るとき基本となるメソッド
  - HTMLのタグとコンポーネントを関連づける
  - なんか、追加するコンポーネントがたくさんあるとだるい、保守しにくい

```
<h2>名前入力フォーム</h2>
<form wicket:id="form">
  <p><input wicket:id=“name” type=“text” /><p>
  <p><input type=“submit” /></p>
</form>
```

```
private class NameForm extends Form {
  private String name;

  public InputForm(String id) {
    super(id);
    add(new TextField(“name”,
      new PropertyModel(this, “name”)));
  }
}
```

IConponentResolver

```
public interface IComponentResolver extends IClusterable {
  public boolean resolve(
    MarkupContainer container,
    MarkupStream markupStream,
    ComponentTag tag);
}
```

- コンポーネント自動解決のためのインターフェース
  - HTMLのパース中にMarkupContainerに登録されていないwicket:idが出現したとき呼び出される
- MarkupContainerに実装するだけで"勝手に"呼び出される
  - 登録とか何もない。implementsするだけ。ビックリ

IComponentResolver 使い方

```
public interface IComponentResolver extends IClusterable {
  public boolean resolve(
    MarkupContainer container,
    MarkupStream markupStream,
    ComponentTag tag);
}
```

コンポーネントを解決できなかったコンテナ

パース＆変換中のストリーム

解決できなかったタグの情報

タグの情報が渡されるのでcontainerに対してコンポーネントを追加するだけ。

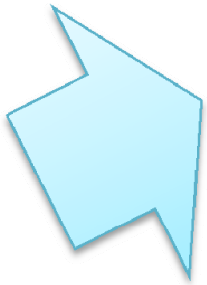
```
public class HomePage extends WebPage implements IComponentResolver {
  public boolean resolve(MarkupContainer container,
                         MarkupStream markupStream, ComponentTag tag) {
    if (tag.isAutoComponentTag())
      return false;

    Label label = new Label(tag.getId(),
      "Component Not Found “ + "[id:" + tag.getId() + "]");
    return container.autoAdd(label, markupStream);
  }
}
```

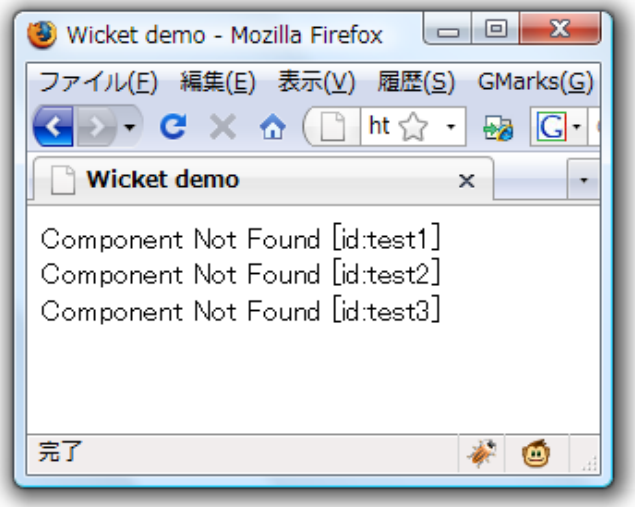
タグのIDを取得してラベルを生成

ラベルをコンテナに追加

IComponentResolver コードと出力

```
<?xml version="1.0" encoding="UTF-8" ?>
<html xmlns:wicket>
<head>
  <title>Wicket demo</title>
</head>
<body>
  <div wicket:id="test1" />
  <div wicket:id="test2" />
  <div wicket:id="test3" />
</body>
</html>
```






```
public class HomePage extends WebPage implements IComponentResolver {
  public boolean resolve(MarkupContainer container,
        MarkupStream markupStream, ComponentTag tag) {
    if (tag.isAutoComponentTag())
      return false;

    Label label = new Label(tag.getId(),
      "Component Not Found “ + "[id:" + tag.getId() + "]");
    return container.autoAdd(label, markupStream);
  }
}
```

IComponentResolver 豆知識

- ComponentTagは、勝手に追加したタグの属性も取得できる
  - つまりはresolverの制御コマンドをタグに埋め込める
  - .propertiesファイルを使うとあんなことやそんなことまで
- タグ内のテキストを取得するのは少し面倒
  - MarkupStreamの動作について知る必要がある

CompoundPropertyModel

- コンポーネントと指定したオブジェクトのプロパティをその名前で結びつけるModel

```
<h2>名前入力フォーム</h2>
<form wicket:id="form">
  <p><input wicket:id="name" type="text" /><p>
  <p><input type="submit" /></p>
</form>
```

名前で関連づける

```
private class NameForm extends Form {
  private String name;

  public InputForm(String id) {
    super(id);
    setModel(new CompoundPropertyModel<NameForm>(this));
  }
}
```

FakeTeeda
（似非Teeda powered by Wicket）

- 以上を踏まえるときわめて怪しいことが可能
  - Formクラスのインスタンス自身をCompoundPropertyModelに入れて、コンポーネントが自身のIDでFormのフィールドを参照できるようにFormクラスを継承。
  - <input>タグや<select>タグのタイプによって自動生成するコンポーネントを変えてしまうようなResolverを実装。
  - 必須項目を属性としてタグに追加。ついでにValidation情報も追加できるようにしちゃったり。Reslover特盛り化。
  - ListViewの子要素のコンポーネント追加処理も自動化しちゃったりなんかして。

FakeTeeda
書いたコード

```
<h2>登録フォーム</h2>
<form wicket:id="form">
<dl>
  <dt>お名前</dt><dd><input wicket:id="name" type="text" size="30" wicket:required="true" /></dd>
  <dt>参加形態</dt><dd><select wicket:id="entry" wicket:type="enum:org.karatachi.demo.EntryType" /></dd>
  <dt>懇親会</dt><dd><select wicket:id="party" wicket:type="enum:org.karatachi.demo.PartyType" /></dd>
  <dt>一言</dt><dd><input wicket:id="message" type="text" size="80" /></dd>
  <dd><input type="submit" value="登録" /></dd>
</dl>
</form>
```

```
private class InputForm extends SelfResolveForm implements Cloneable {
    String name;
    EntryType entry = EntryType.一般参加;
    PartyType party = PartyType.参加しない;
    String message;

    public InputForm(String id) {
        super(id);
    }

    @Override
    protected void onSubmit() {
        list.add((InputForm) this.clone());
    }
}
```

FakeTeeda
出力結果

ちなみに正規表現での
Validationなんかも
タグに記述可能です可能です

ん？Wicket？なにそれ？

Architecture of Wicket with DI Container

WicketにおけるDIコンテナのサポートとその利用

Problem
WicketでDIを利用する上での問題

- Wicketに登録するオブジェクトはすべてSerializableでなくてはいけない
  - WicketはPageクラスのシリアライズ、デシリアライズを繰り返してセッションを保つ
  - DIで利用するオブジェクトはSerializableにすることが難しいことが多い
- Wicketでもコンポーネント管理をおこなっている
  - DI Containerの管理とバッティング
  - Hot Deployを利用しようとするとClassLoaderが2系統作られDI ContainerのHot Deployが利用できない

Objective
解決に向けての

- WicketとDI Containerの間にProxyをはさむ
  - 他方のオブジェクトにアクセスする際には、必ずコンポーネントマネージャーを一度通す

イメージ

```
private Foo foo;

@Override
public void onPageAttached() {
  foo = SingletonS2Container.getComponent(Foo.class);
}

@Override
protected void onDetach() {
  foo = null;
}
```

ページが表示されるときマネージャから取得

表示し終わったら一度nullにもどす

Solutions

- wicket-ioc
  - クラスのフィールドをProxyとするためのライブラリ
  - wicket-springやwicket-guiceなどが利用
- `IComponentInstantiationListener`
  - WebApplication初期化時に登録すると、全Component継承クラスでコンストラクタの一番はじめに呼び出される
  - この時点でDIを利用できるようにすると、継承クラスでのコンストラクタでDIされたコンポーネントを利用できる
- `ReloadableWicketFilter`
  - WicketでHot Deployを実現しているFilter
  - DI Container側でこいつのClassLoaderを利用するようにしてやればDIContainer側もHot Deployされる

Etc...
結局使わなかったけれども

- `IInitializer & IDestroyer`
  - WebApplicationの初期化時・破棄時に呼ばれるインターフェース
  - 拡張ライブラリを作ったときにjarファイル内のwicket.propertiesにinitializerという名前で実装クラスのパスを書いておくと自動的に起動される

Future Work
似非S2Wicket

- Hot Deployの改良
  - Hot Deploy時にSeasar側で全コンポーネントがリロードされてしまうため、PermGenスペースが開発時にすぐ無くなる。
  - 現在あれこれ呼び出し順序などの組み合わせを入れ替えつつ調整中
- 公開にむけて
  - ドキュメントの準備
  - 先のComponentResolverを使用した作りためたコンポーネント群をどうするか