T2ユーザーガイド

T2 - the WEB connector -

T2は、Webにつなげる・つながる部分に特化した部品化指向のWebアプリケーションフレームワークです.
「**the WEB Connector**」というコンセプトを元に、小さく扱いやすい部品として提供します. Webアプリケーションの入り口部分に特化して徹底的にリッチにし、そのほかの部分を削り落とすことで部品として使いやすい・組み込みやすいフレームワークを継続的に提供するのが私達のゴールです.

目次

1.T2の特徴

T2の特徴、扱うモデルについて簡単に記載します.

1.1.T2の特徴

T2はWebフレームワークの一種で、以下のような特徴を持っています.

- Webにつなげる・つながる部分にのみ特化しており、ユーザが制御しやすい
- Ajaxやリッチクライアントのような現代的なWebアプリケーションに必須な技術や製品などとの連携を念頭においている
- 部品化指向なフレームワークなので、ユーザが自分のフレームワークやアプリケーションに組み込んで使いやすい
- ビューテクノロジを選ばないので、JSPでもFlexでもSilverlightでも好きなViewテクノロジが使える.
- 扱うモデルがPOJOで出来ており、特別なインタフェースや抽象クラスを継承する必要がない
- ちょっと足りないくらいが丁度良いというポリシーで作り、足りない場合は拡張してもらう。小さくシンプルな内部構造なのでユーザが拡張しやすいようにする

1.2.T2のPageモデル

T2では**Page**モデルという考え方をもっています.
POJO(シンプルなJavaオブジェクト)とURLの断片を1対1でマッピングして使います.マッピングにはアノテーションを使います.

たとえば、足し算を行うアプリケーションでは、画面側（JSP）で、
<form name="addForm" action="/t2-sample/add" method="post">
　.....

のように指定し、一方でそのマッピングするPageクラスには@Page("add")とアノテーションで指定しておきます.
アノテーションの引数の中にはURLの断片を記述します.
例えば下記のようになります.クラス名は特にPageとする必要はありません.

```
@Page("add")
public class AddPage {
    ......
```

まとめると、ある**URL**からリクエストが送られると、その**URL**で登録しておいた**Page**クラスが呼ばれる、そんなシンプルなつくりになっています. また、たとえばリンクで指定する場合も、

```
<ul>
  <li><a href="/t2-samples/add">足し算</a></li>
   ......
```

のようにマッピングしているURLを介してPageクラスが呼ばれます.

1.3.Pageクラスの各部品の呼び方と役割

Pageクラスの各部品を以下のような名称で呼びます.

```
@Page("hoge") //<---- Pageアノテーション（必須）.引数でPageパスを指定する.
public class Sample { //<---- Pageクラス（末尾にPageとつける必要はない）

   @GET //<----  アクションメソッドアノテーション
   @ActionPath("foo")  //<----  アクションメソッドアノテーション
   public Navigation index() { //<---- アクションメソッド(戻り値は、Navigationインタフェースの実装クラス)
       return ......
   }
}
```

Pageアノテーションとは、T2のPageクラスをあらわすためのアノテーションです.これがついていないクラスはT2が処理するPageクラスとみなされません.Pageアノテーションで指定するURL（Pageパス）は一意である必要があります.

アクションメソッドアノテーションとは、上記サンプルのようにView側から呼ばれるメソッド（アクションメソッド）を指定するためのアノテーションです.
具体的には下記の2点の事をあらわします.

- このメソッドはどのプロトコルや**HTTP**のメソッドで呼ばれるか
- そのほかにどのような条件を満たすときに呼ばれるか

上の例では、こうなります.

- HTTPのGETメソッドのときに、
- URLのパスが、@Pageに記載されている「hoge」＋@ActionPathで記載されている「foo」=/hoge/fooにアクセスしている

ときに呼ばれます.

アクションメソッドは、View側から呼ばれるPageクラスのメソッドです.アクションメソッドの戻り値は、遷移先と遷移する方法を持つ**Navigation**インタフェースの
実装クラスである必要があります.

2.稼動環境

2.1.JDKバージョン

T2では**JDK1.6**以上必須です.

2.2.T2の依存している**JavaEE**仕様

T2は以下の仕様に依存しています.

- **Servlet**仕様（**2.5**以上）
- **JSP(JavaServer Pages)**仕様**(2.0**以上**)**

2.3.T2の依存しているフレームワーク/ライブラリ

T2は以下のフレームワーク/ライブラリが必須です.

- **commons : T2**の共通ライブラリ
- **slf4api :** ロギング用インタフェース**API(**実際のロギングライブラリは選択可能**)**

以下のフレームワーク/ライブラリはオプションですが必要になる場合があります.

- ログ出力
    - logback-core：ロギングのコアライブラリ

    - logback-classic：slf4j-apiの実装ロギングライブラリ
- ファイルアップロードがしたいとき
    - commons-io：ApacheのIO出力ライブラリ
    - commons-fileupload：Apacheのファイルアップロードライブラリ
- DIコンテナが使いたいとき
    - Lucy：T2プロジェクトで作られたシンプルなDIコンテナ
    - Seasar2：Seasarプロジェクトで作られたDIコンテナ(必要なライブラリはサイトを参照ください)
    - Spring：SpringSourceで作られたDIコンテナ(必要なライブラリはサイトを参照ください)
    - Guice：Googleで作られたDIコンテナ(必要なライブラリはサイトを参照ください)

3.1.**T2**機能

T2の機能はJava5のアノテーションをコア技術として採用しています.
アノテーションとは、Javaの標準機能のひとつでクラスやメソッドにつける注釈のようなものです.
T2 はこのアノテーションを読み取って、実行時値を設定したり、あるメソッドを呼び出すか・呼び出さないかを決定したりします.

T2が使うアノテーションには大きく分けて3つあります.

- クラスにつける**クラスアノテーション**
- メソッドにつける**メソッドアノテーション**
- メソッドの引数につける**引数アノテーション**

クラスアノテーションはクラス全体で有効なもので、メソッドアノテーションはそのメソッドだけに影響し、引数アノテーションはメソッドに
渡されるパラメータだけに影響するアノテーションです.
このように、アノテーションをソースコードに書いておくことで、「このクラスはPageクラスなので、/hogeというURLの時に呼び出してほしい」ということや、
「このメソッドは、リクエストパラメータに"foo"という文字列があった時に呼んでほしい」というような事を、T2に伝えることができます。

3.1.1.規約

T2では極力規約は使いません.**規約よりもアノテーションを重要視**しています.
しかしながらミニマムレベルの規約は導入しており、下記のようになります.

- **クラスアノテーション/メソッドアノテーションで省略可能な値がある場合、原則として付与されているクラス・メソッド名が使われます.**
- 引数アノテーションで、代替手段として型によって情報が十分得られる場合、引数アノテーションがなくても必要なインスタンスが設定されます.（たとえば**@UploadFile**と**UploadFile**クラスなど.）

3.2.アノテーション

T2で使うアノテーションについて説明します.

3.2.1.クラスアノテーション

・**@Page**アノテーション
Pageコンポーネントであることを通知するアノテーションです. **T2**では必須です.
属性は下記のとおりです.

属性名	説明
value	アクセスされたときのURLを記載する. 省略した場合、Pageクラス名が設定されたとみなす.

サンプルはこのようになります.下記の例では、**http://localhost:8080/context-root/hello** のようなURLにマッピングされます.

```
@Page("hello")
public class HelloPage {
    .....
```

3.2.2. メソッドアノテーション

・**HTTP**メソッドアノテーション群

HTTPメソッドアノテーションはHTTPメソッドをあらわします.HTTPメソッドアノテーションがついていると、そのHTTPメソッドでリクエストされたときだけ、アクションメソッドが呼ばれます.

以下のようなHTTPメソッドアノテーションがあります.各アノテーションをアクションメソッドにつけておくと、該当のHTTPメソッドのみでそのアクションメソッドが呼ばれます・

HTTPメソッドアノテーション(*1)	説明
@GET	HTTP GETメソッドを表します・
@POST	HTTP POSTメソッドを表します・
@PUT	HTTP PUTメソッドを表します・
@DELETE	HTTP DELETEメソッドを表します・
@HEAD	HTTP HEADメソッドを表します・
@OPTIONS	HTTP OPTIONSメソッドを表します・

*1：HTTPメソッドアノテーションのパッケージ名は
org.t2framework.t2.annotation.compositeです・

デフォルトでは、**@GET/@POST**だけが使えるようになっています.

・**@ActionPath**アノテーション**(org.t2framework.t2.annotation.core.ActionPath)**

ActionPathアノテーションは**URL**指定でメソッドが呼べるか呼べないかを指定するアノテーションです.通常のHTTPメソッドアノテーションだけだと、ServletのようにHTTPメソッドが同じであればどのリクエストでも受け取ってしまうので、このActionPathアノテーションで制限を加えます.

属性は下記のとおりです.

属性名	説明
value	アクセスされたときのURL.省略されると、メソッド名と同一のものが呼ばれる.

サンプルは以下のようになります.下記の例では、**http://localhost:8080/context-root/hello/request** のようなURLでリクエストが来た場合に、requestメソッドが呼ばれます.

```
@Page("hello")
public class HelloPage {
    @ActionPath
```

```
    public Navigation request(HttpServletRequest request) {

    .....
```

・**@ActionParam**アノテーション**(org.t2framework.t2.annotation.core.ActionParam)**

ActionParamアノテーションは主にPOSTで使用され、**submit**時に**name**属性で指定された**value**が同一であれば指定したアクションメソッドが呼ばれるアノテーションです.
@ActionParamとつけることで、画面上に配置されたボタンと、Pageクラスのメソッドが1対1で対応するので直感的にわかりやすく、メソッドもアクションごとに分割することが出来ます.

属性値は下記のとおりです.

属性名	説明
value	主にボタン押下時のname属性.省略すると、メソッド名と同一のボタン等が押されてPOSTされたものとする.

サンプルは以下のようになります.下記の例では、画面側で**submit**したリクエストパラメータで**execute**という値を持つものがあれば、executeメソッドが呼ばれます.

```
@Page("add")
public class AddPage {
    @ActionParam
    public Navigation execute(HttpServletRequest request) {

    .....
```

・**@Default**アノテーション**(org.t2framework.t2.annotation.core.Default)**

Defaultアノテーションは、該当Pageクラス内でどのアクションメソッドも呼ばれなかったときのデフォルトのメソッドを指定します・T2では各Pageクラスに対して、デフォルトメソッドを作ることを推奨しています・

サンプルは以下のようになります.下記の例ではAddPage内でどのアクションメソッドの呼び出し条件ともマッチしなかった場合、@Defaultを指定したindexメソッドが呼ばれます.

```
@Page("add")
public class AddPage {

    @Default
    public Navigation index(WebContext context) {
        return Redirect.to("/jsp/add.jsp");
    }
    .....
```

・**@Ajax**アノテーション**(org.t2framework.t2.annotation.core.Ajax)**

Ajaxアノテーションは**Ajax**リクエストによって呼ばれるアクションメソッドを示すアノテーションです.@Ajaxとついているメソッドは、クライアント上からXmlHttpRequestでリクエストが来た場合にしか呼び出せなくなります.これによって、通常のボタンサブミットなどのリクエストと、Ajaxでのリクエストを明確に分離することが出来ます.**Ajax**リクエストを処理する場合には、このアノテーションが必須です**.**

サンプルは以下のようになります.下記の例では、クライアントサイドから**Ajax**リクエストでかつ**POST**だった場合に、AjaxExtJsPage.execute()が呼ばれます.

```
@Page("/ajaxExtJs")
public class AjaxExtJsPage {

    @Ajax
    @POST
    public Navigation execute(WebContext context) {
        final Request request = context.getRequest();
        final String hoge = request.getParameter("hoge");
        return Json.convert(hoge + " is ExtJS");
    }

    .....
```

@Ajaxの注意事項

- 厳密には**@Ajax**で処理できるのは、リクエストヘッダに「**X-Requested-With**」というキーで、「**xmlhttprequest**」という値が入っている場合に限る
    - 近代的なJavaスクリプトフレームワークではどれもこのリクエストヘッダを採用している
- 現在のところ、下記のJavaスクリプトフレームワークで動作を確認した
    - jQuery 1.2.6
    - Ext JS 2.2.1
    - prototype js 1.6.0.3

・**@AMF**アノテーション

FlexまたはAIRからAMF通信（高速バイナリ通信）で呼ばれるアクションメソッドを示すアノテーション(Version 0.6から)

3.2.3.パラメータアノテーション

・**@Form**アノテーション**(org.t2framework.t2.annotation.core.Index)**

@Formはサブミットされてきたformを**Dto**に変換して扱うためのアノテーションです.

・**@Index**アノテーション**(org.t2framework.t2.annotation.core.Index)**

@Indexアノテーションは、サブミットされたボタンなどでForEach内のindexを取得するためのアノテーションです.

・**@Var**アノテーション**(org.t2framework.t2.annotation.core.Var)**

@Varは、URL断片を取得するためのアノテーションです.

・**@Uplodaアノテーション(org.t2framework.t2.annotation.core.Uploda)**
@Uploadは、FileUploadされたファイルを取得するためのアノテーションです.
commons-io/commons-fileuplodaが必要です.

・**@RequestParamアノテーション**
(org.t2framework.t2.annotation.core.RequestParam)
@RequestParamは、リクエストパラメータを指定して取得するためのアノテーションです.

・**@SessionAttrアノテーション(org.t2framework.t2.annotation.core.SessionAttr)**
@SessionAttrは、HTTPセッション内のパラメータを指定して取得するためのアノテーションです.

・**@RequestHeaderアノテーション**
(org.t2framework.t2.annotation.core.RequestHeader)
@RequestHeaderは、リクエストヘッダ内の値を指定して取得するためのアノテーションです.

3.3.API

3.3.1.Navigationクラス

NavigationクラスはT2のアクションメソッドの次の画面遷移先とその出力内容を決めるためのクラスです.
例えば下記のようになります.下記の例では、シンプルに/WEB-INF/pages/login.jspにフォワードしています.

```
@Default
public Navigation index(final WebContext context) {
    return Forward.to("/WEB-INF/pages/login.jsp");
}
```

もう一つ具体例を見てみましょう.下記の例では、オブジェクトをJSON形式に変換してレスポンスに値を出力します。

```
@Default
public Navigation toJson(WebContext context) {
    Hoge jsonObject = new Hoge();
    return Json.convert(jsonObject);
}
```

このようにNavigationは用途によって使い分けることで様々なレスポンスを返すことが出来ます.また、Navigationインタフェースを実装すれば独自のNavigationも簡単に実装できます.Navigationは下記のようなインタフェースになっています.

```
@Published
public interface Navigation {

    /**
     * Execute navigation processing.
     *
     * @param context
     * @throws Exception
     */
    void execute(WebContext context) throws Exception;

}
```

現在T2であるNavigationの実装クラスは下記のようになります.

Navigationクラス名（※１）	説明	代表的な使い方
Forward	フォワード処理	Forward.to("next.jsp");
Redirect	リダイレクト処理	Redirect.to("next.jsp");
NoOperation	何もしない.サーブレットフィルタのFilterChain.doFilter()の呼び出しもしない.	NoOperation.noOp();
PassThrough	サーブレットフィルタのFilterChain.doFilter()	PassThrough.pass();
Direct	渡されたデータを直接HttpServletResponseに書き込む	Direct.from(file);
Json	渡されたデータをJSON形式にしてHttpServletResponseに書き込む	Json.convert(targetObject);

※１：Navigationのクラスは全て**org.t2framework.t2.navigation**パッケージ

3.4.設定

3.4.1.web.xmlの基本設定

・エンコーディング指定

リクエストのエンコーディング指定はT2のフィルタクラスによって行われます.

コンテキストパラメータに以下の値を設定することでエンコーディングをセットします.

- **t2.encoding：**エンコーディング指定する

設定例：

<context-param>

```
        <param-name>t2.encoding</param-name>
        <param-value>UTF-8</param-value>
    </context-param>
```

・**T2**の本体のフィルタ動作指定

T2の起点となる org.t2framework.t2.filter.T2Filterの設定オプションは下記のようなものがあります.
必須なのは、Pageクラスのあるルートパッケージの登録です.T2ではルートパッケージ以下サブパッケージも見て、Pageクラスの
メタ情報を初期起動時に作るためです.

設定項目	必須/任意	デフォルト値	説明
t2.rootpackage	必須	なし	Pageクラスを読み込むためのルートパッケージ.カンマ区切りで複数指定可能.
t2.container.adapter	任意	LucyContainerAdapter	IoCコンテナとの連携機能.標準ではシンプルIoCコンテナ「Lucy」が使われる.
t2.config	任意	なし	IoCコンテナの任意の設定ファイルを使うときに指定する.
t2.eagerload	任意	false(自動登録しない)	T2のPageクラスの自動登録機能を使う場合はtrueに設定する.

・設定のサンプル

サンプルとして以下のような場合を考えてみましょう.

- ルートパッケージに「examples.employee.page」,「examples.admin.page」
- IoCコンテナにSeasar2を使用する
- Seasar2の設定ファイルとしてapp.diconを指定する
- Pageクラスの自動登録はしない

web.xmlの設定のサンプルは、下記のようになります.

```
<filter>
   <filter-name>t2</filter-name>
   <filter-class> org.t2framework.t2.filter.T2Filter</filter-class>
   <!-- ルートパッケージの指定 -->
    <init-param>
       <param-name>t2.rootpackage</param-name>
       <param-value>examples.employee.page,
examples.admin.page</param-value>
    </init-param>
    <init-param>
       <param-name>t2.container.adapter</param-name>
       <param-
value>org.t2framework.t2.adapter.S2Adapter</param-value>
```

```
    </init-param>
    <!-- 設定ファイルの読み込み -->
    <init-param>
      <param-name>t2.config</param-name>
      <param-value>app.dicon</param-value>
    </init-param>
    <!-- Pageの自動登録 -->
    <init-param>
      <param-name>t2.eagerload</param-name>
      <param-value>false</param-value>
    </init-param>
</filter>
```

3.4.2.web.xmlのファイルアップロード設定

ファイルアップロード機能を使う場合、個別にフィルタを設定する必要があります.
フィルタクラスは、org.t2framework.t2.filter.MultiPartRequestFilterになります.
設定項目としては以下のようなものがあります.

設定項目	必須/任意	デフォルト値	説明
uploadMaxSize	任意	100MB	リクエストの最大サイズ
uploadMaxFileSize	任意	100MB	アップロードファイル1つの最大サイズ
uploadThresholdSize	任意	10MB	ディスク書き出しファイルの閾値
uploadRepositoryPath	任意	なし	テンポラリのディスク書き出しのパス

設定サンプルは下記のようになります.

```
  <filter>
    <filter-name>uploadFilter</filter-name>
    <filter-class>org.t2framework.t2.filter.MultiPartRequestFilter
    </filter-class>
    <init-param>
      <param-name>uploadMaxSize</param-name>
      <param-value>100m</param-value>
    </init-param>
    <init-param>
      <param-name>uploadMaxFileSize</param-name>
      <param-value>100m</param-value>
    </init-param>
    <init-param>
      <param-name>uploadThresholdSize</param-name>
      <param-value>100k</param-value>
    </init-param>
    <init-param>
      <param-name>uploadRepositoryPath</param-name>
      <param-value>C:/temp/</param-value>
    </init-param>
  </filter>
```

3.5.推奨プラグイン

T2のサンプルを動かすために、以下のプラグインを推奨します.勿論あくまで推奨ですので使いやすいプラグインを
使っていただいてもかまいません.T2はどのプラグインにも依存しているわけではありません.

WebLauncherプラグイン　：　Tomcat/Jetty/SDLoaderをサポートしたWebコンテナブートストラップのためのEclipseプラグインです・
Eclipseプラグイン Updateサイトは以下のとおりです.Eclipseのプラグインアップデートからお使いください・使い方についてはまた別途記載します.

http://dev.handwerkszeug.org/eclipse/3.4/werkzeugkasten.update

4.T2サンプル

4.1.Pageクラスの作り方

Pageクラスは、リクエストスコープ（リクエストの度にPageクラスが生成される）を推奨します.

特に小規模案件ではユースケースをPageパスにマッピングし、そのままPageクラスをマッピングするので 1ユースケース:1Pageパス:1Pageクラスとなります.
例えば従業員管理を考えるとEmployeePageのような Pageクラスがいて、その中に従業員一覧・従業員追加などのメソッドが全て定義されているような形になります.

例えば、http://www.sample.com/root/hoge/fooというURLとするとPageクラスとのマッピングは以下のようになります.

http://www.sample.com
+/root(コンテキストルート**)**
+/hoge(Pageパス**)**
+/foo(Actionパス**)**

中大規模案件では、各ViewごとにPageクラスを個別に作成し、 1View:1Pageパス:1Pageクラスとなります.
例えば先ほどの従業員管理を考えると、 従業員一覧画面・従業員編集画面・従業員削除画面といった各画面ごとにPageクラスを作成します.

例えば、http://www.sample.com/root/subroot/hoge/fooというURLで、Pageクラスとのマッピングは以下のようになります.

http://www.sample.com
+/root(コンテキストルート**)**
+/subroot/hoge(subrootを1ユースケースとし、それとあわせて**Page**パスとする**.hoge**は**1View**に対して割り当てられる**)**
+/foo(Actionパス**)**

4.2.引数渡しモデル

Pageクラスは、欲しいインスタンスは全てメソッドの引数によって取得します.
これを引数渡しモデルと呼びます.引数に適切なオブジェクトを記載しておけば、T2がセットしてくれます.ちょうどLucyやSeasar2、SpringなどのDIコンテナがオブジェクトをインジェクトするのと同じです.

引数渡しモデルの代表例はApache Struts です.
StrutsのActionクラスはActionMapping/ActionForm/HttpServletRequest/HttpServletResponseを引数としてexecuteメソッドに渡します.T2ではメソッドで渡す引数（とメソッド呼び出し部）にもう少し柔軟性をもたせていますが、View層から呼び出されるメソッドに対して、必要なオブジェクトはそのメソッドの引数で渡す点は同じです.

例えばメソッドは以下のようになります.

```
public Navigation add(HttpServletRequest request, HttpServletResponse response)
```

この例では、HttpServletRequest、HttpServletResponseを引数で受け取ることが出来ます.

このように、開発者は欲しいクラスのインスタンスを、アノテーションもしくは型宣言で引数に指定することで、メソッド引数で受け取るのが
引数渡しモデルの特徴です.

T2では以下のようなクラスのインスタンスがアノテーションなしで受け取れます.

クラス名	説明
HttpServletRequest	サーブレットリクエスト
HttpServletResponse	サーブレットレスポンス
HttpSession	HTTPセッション
ServletContext	サーブレットコンテキスト
Cookie/Cookie[]	CookieまたはCookieの配列
WebContext	T2のコンテキストオブジェクト。全ての情報が詰まっている
Request	T2のリクエストオブジェクト
Response	T2のレスポンスオブジェクト
UploadFile	T2用のアップロードファイルオブジェクト

4.3.サンプルコード

T2の機能をサンプルコードを使って紹介します・

基本的に欲しい値はメソッドの引数で渡すのがT2のやり方なので、メソッドにつくアノテーションとどのような引数をもらうのか、引数のアノテーションは何を使うかが決まれば簡単に制約を与えることが出来ます.たとえば自動生成でコードを吐き出したりするのは とても簡単に行うことが出来ます.

4.3.1.はじめに（リソースのありかなど）

本ドキュメントのソースコードは全てサンプルプロジェクトとして公開しています.
t2-samples-(バージョン番号).zipという形式で配布していますので、そちらもあわせてご覧ください.

下記サイトからダウンロード可能です.
http://code.google.com/p/t-2/downloads/list

4.3.2.最もベーシックな@GET/@POSTを使ったサンプル（Servlet3.0ライク）

HTTPのGETとPOSTのそれぞれのメソッドで受け取る簡単なサンプルを作ってみましょう.単純なServletに近いようなやり方です.
T2では@GETと@POSTのようにHTTPメソッドアノテーションをつけることで、実現できます.

@GETと@POSTが書いてあるメソッド（下記ソースコードのpostメソッド、getメソッド）に注目してみてください.
アノテーションがそれぞれ@POST、@GETとついているのがわかりますでしょうか？このアノテーションによって、HTTPのPOSTとGETでそれぞれのメソッドが呼ばれる仕組みになっています.

```
@Page("getandpost")
public class GetAndPostPage {
……
    /**
     * {@.ja @POST はHTTPのPOSTメソッドのときに呼び出されます. }
     *
     */
    @POST
    public Navigation post(HttpServletRequest request,
            HttpServletResponse response) {
        request.setAttribute("message", "Do POST.");
        return Forward.to("/jsp/simpleGetAndPost.jsp");
    }

    /**
     * {@.ja @GET はHTTPのGETメソッドのときに呼び出されます. }
     *
     */
    @GET
    public Navigation get(HttpServletRequest request,
            HttpServletResponse response) {
        request.setAttribute("message", "Do GET.");
        return Forward.to("/jsp/simpleGetAndPost.jsp");
    }

}
```

このソースコードの全文はt2-samplesのGetAndPostPage にあります.

4.3.3.@ActionPathを使ったサンプル

T2の最も基本的な方法が下記のような**@ActionPath**と**@ActionParam**を使った方法になります**.@ActionPath**を使えば、**HTTP**のメソッド指定から、さらに**URL**でメソッドを呼ぶかどうかを絞り込めます**.**

下記のサンプル(HelloPage)では@ActionPathでrequestメソッド（ソースコード内参照）が呼ばれる条件を以下のように絞っています.

- リクエストされたURLが、http://yourdomain/t2-samples/hello/requestのとき

```
@Page("hello")
public class HelloPage {
    /**
     * {@.ja @ActionPath は、URL指定でメソッドが呼べるか呼べないかを
     * 指定するアノテーションです.
     * @Page でつけたURL断片＋@ActionPathをつけたメソッド名なURLに
     * リクエストすれば呼ばれます.
     * HelloPageのこのメソッド(requestメソッド)だと、
     *        http://yourdomain/t2-samples/hello/request
     * で呼ばれます. }
     */
    @ActionPath
    public Navigation request(HttpServletRequest request) {
        System.out.println("request.getContextPath() : "
                + request.getContextPath());
```

```
        request
                .setAttribute("greet", helloService.hello()
                        + " from request().");
        return Forward.to("/jsp/hello.jsp");
    }

.....
}
```

このサンプルの全文はt2-samplesのHelloPage にあります.

4.3.4.@ActionParamを使ったサンプル

一方**@ActionParam**はサブミットされたときのリクエストパラメータでメソッドが呼ばれるか呼ばれないかの条件を絞ります**.**

例えば下記のサンプルのaddメソッドは、以下の場合に呼ばれます.

- HTTPのPOSTメソッドが使われた
- リクエストパラメータにaddという名前の属性がある
- 画面側のFORMのactionが/context-root/add

```
@Page("add")
public class AddPage {
    /**
     * {@.ja @ActionParam は画面からサブミットされたFORMの中の入力項目に
     * メソッド名と同じname属性を持つ項目があれば、対象メソッドを呼ぶ
     * アノテーションです.
     * このアノテーションを使えば、押されたボタンを認識して、処理をユーザの
     * アクション単位できれいに切り分けられます.
     *
     * この例では以下のような場合に呼ばれます.
     *  <li>画面側のFORMのactionが/context-root/add</li>
     *    <li>POSTでサブミットされる</li>
     *    <li>サブミットされたフォーム内の入力項目に、addというname属性がある</li>
     * }
     */
    @POST
    @ActionParam
    public Navigation add(WebContext context) {

        //リクエストの値検証、足し算のロジックなどの記述

        return Forward.to("/jsp/add.jsp");
    }
.....
}
```

JSP側の記述は下記のようになります.${t:url}はT2のFunction関数を使って、コンテキストルートを補完しています.

```
<form name="addForm" action="${t:url('/add')}" method="post">
   <input type="text" name="arg1" />
   <br />
   <input type="text" name="arg2" />
   <br />
   <span>${result}</span><br />
   <input type="submit" name="add" value="同一ページ"/>
</form>
```

このサンプルの全文はt2-samplesのAddPage とt2-samplesのadd.jsp を見てみてください.

4.3.5.@Formによるフォームオブジェクトを受け取るサンプル

@Formを使うと、サブミットされたフォームをJavaオブジェクトの形にして、引数に渡すことが出来ます。フォーム内のオブジェクトをまとめて扱いたい場合は、@Formを使えば一つのオブジェクトとして扱えます。

先ほどのAddのサンプルを@Formで書き換えてみましょう。

```
@Page("add")
public class AddPage {
   /**
    * {@.ja @Form は、サブミットされたFORMの値全てをJavaのオブジェクトに
    * マッピングしてもらうためのアノテーションです.
    * 型があわないなどの変換エラーが発生した場合、その値はnullのままに
    * なります.@Formで変換エラーが出たときの情報は{@code ErrorInfo}に
    * よって取得することが可能です. @Form はバージョン0.4からの追加機能です.
    *
    * @since 0.4
    */
   @POST
   @ActionParam
   public Navigation addWithForm(@Form AddForm dto,
WebContext context, ErrorInfo errorInfo) {
      Request request = context.getRequest();
      RequestUtil.save("arg1", request);
      RequestUtil.save("arg2", request);
      if (errorInfo.hasError()) {
         request.setAttribute("message",
MessageConstants.ADD_ERROR_MESSAGE);
         return Forward.to("jsp/add.jsp");
      }
      request.setAttribute("result", new Integer(dto.getArg1().intValue()
            + dto.getArg2().intValue()));
      return Forward.to("/jsp/add.jsp");
   }

}
```

AddFormクラスは単純なPOJOです.

```
/**
 * {@.ja {@code examples.page.AddPage}で使うフォームオブジェクトです. }
 *
 * @author shot
 */
public class AddForm implements Serializable {

    private static final long serialVersionUID = 1L;

    protected Integer arg1;

    protected Integer arg2;

    //単純なgetter/setter
}
```

4.3.6.@RequestParamを使ったサンプル

@RequestParamでは、リクエストそのものを引数としてとるのではなく、リクエスト上の特定パラメーターだけを取り出して、メソッドの引数に渡す事が出来ます。メソッド内で限定された特定パラメーターだけを扱わせたい場合などには有効です。もしキーに指定した値がリクエストに存在しない場合、デフォルトでは空文字が引数として渡されます。また、引数の型としてはString/String[]しか受け付けていません。

```
@Page("requestparam")
public class RequestParamPage {
    /**
     * {@.ja {@code @RequestParam}は引数アノテーションです.
     * 引数の項目につけます.引数の項目はStringまたはString[]である
     * 必要があります.
     * キーは必ず指定する必要があり、値がない場合は空文字列がデフォルトで
     * インジェクトされます.
     * オプション属性で、empty=falseとすると、空文字が渡された場合例外が
     * 発生するようになります. }
     */
    @POST
    @ActionParam
    public Navigation message(@RequestParam("left") String left,
            @RequestParam("right") String right, HttpServletRequest
request) {
        request.setAttribute("message", left + " : " + right);
        return Forward.to("/jsp/requestParam.jsp");
    }
}
```

4.3.7.@SessionAttrを使ったサンプル

@SessionAttrでは、セッション上の特定パラメーターだけを取り出して、メソッドの引数に渡す事が出来ます。メソッド内で限定された特定パラメーターだけを扱わせたい場合などには有効です。もしキーに指定した値がセッションに存在しない場合、デフォルトではnullが引数として渡されます。

サンプルコードは下記のとおりです。

```
/**
 * {@.ja このサンプルでは、0.4から加わった新機能の{@code @SessionAttr}
の
 * 使い方をご紹介します.
 * このサンプルは以下のURLで呼ばれます.
 *
 * http://yourdomain/context-root/sessionattr }
 *
 * @since 0.4
 * @author shot
 */
@Page("sessionattr")
public class SessionAttrPage {

    /**
     * {@.ja {@code @SessionAttr}はSession内に入っている値を
     * 引数として渡す引数アノテーションです.
     *
     * @SessionAttr("hoge")のように、キー名は必須で指定します.
     * キー名にない値はデフォルトではnullがインジェクトされます.
     * null値を許容したく無い場合は、nullable属性をfalseにすると例外が発生
し、
     * このメソッドは呼ばれません.}
     */
    @POST
    @ActionParam
    public Navigation message(@SessionAttr("hoge") String hoge,
            @SessionAttr("no_such_attr") String nosuchAttr,
            HttpServletRequest request) {
        request.setAttribute("message1", "hogeというキーの値" + " : " +
hoge);
        request.setAttribute("message2", "no_such_attrというキーの値" + "
: "
                + nosuchAttr);
        return Forward.to("/jsp/sessionAttr.jsp");
    }

}
```

4.3.8.ファイルダウンロードのサンプル

ファイルのダウンロードのサンプルは下記のようにFileまたはInputStreamをDirectナビゲーションに渡してやります.
下記のサンプルのsimpleDownloadByGetメソッドのように、ダウンロードさせるファイル情報をセットしたあと、Directクラスに Fileオブジェクトを渡してやるだけです。

```
@Page("download")
public class DownloadPage {

    /**
```

```
    * {@.en  }
    *
    * <br />
    *
    * {@.ja ファイルダウンロードはアクションメソッド内でダウンロードファイルに
    * 関する情報を組み立てて、{@code Direct}にFileまたはInputStreamなどを
    * 渡します. }
    */
   @GET
   @ActionParam
   public Navigation simpleDownloadByGet(HttpServletRequest request,
           HttpServletResponse response) {
       String filename = request.getParameter("filename");
       if (StringUtil.isEmpty(filename)) {
           request.setAttribute("result", "filenameは必須です.");
           return list();
       } else if (!filename.endsWith(".csv")) {
           filename = filename + ".csv";
       }
       final File file = downloadHelper.getFile(filename);
       response.setContentType("application/octet-stream");
       response.setHeader("content-disposition", "attachment; filename=\""
               + file.getName() + "\"");
       return Direct.from(file);
   }

}
```

4.3.9.ファイルアップロードのサンプル

ファイルアップロードではUploadFileを使います.T2ではアップロードファイルは全てUploadFileの形式で引数で取得します.
アップロードファイルが一つの場合はUploadFileを型にとって、複数のアップロードファイルの場合は、UploadFile[]としてください.

```
@Page("upload")
public class UploadPage {

   /**
    * {@.en  }
    *
    * <br />
    *
    * {@.ja ファイルがアップロードされたときに、{@code UploadFile}としてアップロードされます.
    * シンプルに一つのアップロードされたファイルを扱う場合にはこれで十分です. }
    *
    * @param file
```

```
     * @param request
     * @return
     */
    @POST
    @ActionParam
    public Navigation upload(UploadFile file, HttpServletRequest
request) {
        System.out.println("file:" + file.getName());
        System.out.println("size:" + file.getSize());
        System.out.println("contentType:" + file.getContentType());
        request.setAttribute("fileUploadResult", file.getName()
                + " is uploaded.");
        return Forward.to("/jsp/upload.jsp");
    }

}
```

なお、アップロードファイルを受け取るためには専用のFilterクラスをweb.xmlに登録しておく必要があります.
設定詳細については**3.4.2.web.xml**のファイルアップロードの設定をご覧ください.

4.3.10.@VarによるURLの一部分を取得するサンプル

RESTのようなURLに意味をもたせているアプリケーションでは、URLの一部を抜き取りたい場合があります。
このような場合に@VarアノテーションでURLの一部を引数として受け取る事が出来ます。

TODO：サンプルの追記

4.3.11.ForEachのインデックスを取得する方法

実プロジェクトではテーブル上の繰り返し項目の各行にボタンを配置して、どのボタンが押されたかを知りたい場合があります.
T2ではその場合、ForEachのインデックス値を@Indexという引数アノテーションで取得して、該当メソッドにインデックス値として渡します.
@Indexを使うには幾つかのルールがあります.

- 引数の型としてはintまたはIntegerである必要があります.
- @ActionParamでaaa[{index}]という形式でvalue値を指定します.indexのところはT2が簡易式言語として解釈して、繰り返し項目のインデックスを当てはめます.また、画面側からはそのようなname属性になるように、name="aaa[{$index}]"のようにしておく必要があります.
- @Indexのvalueはデフォルトで"index"という文字列になっているので、それ以外の場合は@Index("your_index")のように指定する必要があります.

サンプルは下記のとおりになります.@ActionParamでのvalue値の指定の仕方と、@Index引数アノテーションの付け方に注目してみてください.

```
/**
 * {@.en  }
 *
```

```
 * <br />
 *
 * {@.ja 繰り返し項目のインデックス値を取得するためのアノテーションである
 * {@code Index}の簡単なサンプルです. }
 */
@Page("foreach")
public class ForeachPage {

   @POST
   @ActionParam("hoge[{index}]")
   public Navigation hoge(@Index int id, WebContext context) {
      final Integer i = Integer.valueOf(id);
      context.getRequest().setAttribute("message",
            "button : " + String.valueOf(i.intValue() + 1) + "
submitted.");
      return Forward.to("/jsp/foreach.jsp");
   }

}
```

JSPは下記のようになります.サブミットボタンの部分のname属性の記述方法について注目してください.

```
<%@ page contentType="text/html;charset=UTF-8"
pageEncoding="UTF-8" errorPage="/error/debug.jsp"%>
<%@taglib prefix="t" uri="http://www.t2framework.org/web/t2/
functions"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"
/>
<title>繰り返し項目</title>
</head>
<body>
<form method="post" action="${t:url('/foreach')}">
   <c:forEach var="e" items="${hogeList}" varStatus="s">
      <input type="submit" name="hoge[${s.index}]"
value="${e}"/>
      <br />
   </c:forEach>
   <span>${message}</span>
</form>
</body>
</html>
```

4.3.12. もう少し実用的なサンプル

イントラネット系のサンプルである従業員管理のシステムのログイン部分を見てみましょう.
ログイン部分の仕様としては下記のようになります.

- ログイン情報はPOSTでloginというリクエストパラメータと共に送られてくる.

- ログイン情報がOKな場合、遷移先のURLを返す.
- ログイン情報がNGな場合、ログインのViewに戻し、適切なエラーメッセージを表示する.

loginメソッドを注目してみてください.

```
@RequestScope
@Page("login")
public class LoginPage {

   private static Logger logger = Logger.getLogger(LoginPage.class);

   protected LoginService loginService;

   @Default
   public Navigation index(final WebContext context) {
      context.getSession().removeAttribute(Constants.AUTH_KEY);
      return Forward.to("/WEB-INF/pages/login.jsp");
   }

   @POST
   @ActionParam
   public Navigation login(WebContext context) {
      final String user = context.getRequest().getParameter("user");
      final String pass = context.getRequest().getParameter("pass");
      final Map<String, String> map = CollectionsUtil.newHashMap();
      final String contextPath = context.getRequest().getContextPath();
      String next;
      if (StringUtil.isEmpty(user)) {
         map.put("errorMessage", Messages.nls("login_userid_empty"));
         next = "/login";
      } else if (StringUtil.isEmpty(pass)) {
         map.put("errorMessage",
Messages.nls("login_password_empty"));
         next = "/login";
      } else {
         final boolean logined = loginService.login(user, pass, 0);
         if (!logined) {
            logger.debug("Login error");
            map.put("errorMessage", Messages.nls("login_auth_failed"));
            next = "/login";
         } else {
            context.getSession().setAttribute(Constants.AUTH_KEY,
                  System.currentTimeMillis());
            next = "/employee/list";
         }
      }
      map.put("url", contextPath + next);
      return Json.convert(map);
   }

   @Binding(bindingType = BindingType.MUST)
   public void setLoginService(LoginService loginService) {
      this.loginService = loginService;
   }

}
```

画面側はjQueryを使ってAjaxでログイン処理をしています.

```
<script>
$(function() {
   $("#login").click(
      function() {
         var userid = $("#user").val();
         var password = $("#pass").val();
         (途中略)
         //jQueryのAjaxでログイン
         $.ajax(
            {
               "url"    : "${t:url('/login')}",
               "type"   : "post",
               "data"   : {
                  "user" : $("#user").val(),
                  "pass" : $("#pass").val(),
                  "login": "login"
               },
               "success" : function(response) {
                  var o = eval("(" + response + ")");
                  if(!o.errorMessage) {
                     //ログイン処理が成功していれば、次画面遷移
                     location.href = o.url;
                  } else {
                     $("#errorMessage").text(o.errorMessage)
                        .css("fontWeight", "bolder")
                        .css("color", "red");
                  }
               },
               "error"   : function(xmlHttpReq, status, e) {

$("#error").text($("#login_system_error").text());
               }
            }
         );
         return false;
      }
   );
});
</script>
```

サンプルは**T2-employee**として公開しています.

5.ライセンス

5.1.T2のライセンス

T2のライセンスとして、**ASL2.0**を採用します.
その他使用しているライブラリのライセンスは別途記載します.

6.開発体制とリリース

（下記ドキュメントはユーザガイドから切り離す予定です.）

6.1.バージョン管理

開発はsvnのbranchをリリース版とする、「release branch」の方法を採用しています.
ある機能追加をしてリリースする場合には、branchを作成し、そちらで行います. バグ修正は、リリース版を修正した後にtrunkにマージを行います.

6.2.バージョニング

6.2.1.プロダクトのバージョンについて

プロダクトのバージョンについては以下のような形式をとります.

t2-xx.yy.zz(Q)

例えば、t2-1.0.1-GAのようになります. xxは、メジャーバージョンです.数値で表現されます.
アーキテクチャの変更や稼動条件の変更、アプリケーションの作り方を大幅に変えてしまう変更はこのリリースになります.
yyは、マイナバージョンです.数値で表現されます.公開インタフェースの変更を伴う新しい機能の追加、ライブラリの新規追加・削除などはこのリリースになります.
zzは、バグフィックスバージョンです.数値で表現されます.バグ修正、または公開インタフェースの変更を伴わない機能追加、ライブラリのバージョンアップなどはこのリリースになります.
Q*は、識別番号です.アルファベットと数値で表現され、以下のような順番に変化していきます.

- SNAPSHOT：開発途中のリリースです.どのような変更が入っても不思議ではありません.
- Alpha：基本機能は出来上がっている状態です.ただしinterfaceの変更などの変更は以前として入る可能性があります.
- Beta：interfaceはほぼ固定され、機能の基本部分は確定されています.ただし最適化・リファクタリングなどが発生する可能性があります.
- RC：interface及び機能も固定されます.何らかのバグが発生した場合、RC番号をインクリメントします.
- GA：正式リリース版です.
- SP：正式リリース版に深刻なバグがあった場合にリリースされます.これは最新版にしか適用されません.

これらのルールは1.0以降に適用されます. 1.0以前は識別番号はSNAPSHOTとGAの簡易系統とします. バージョン番号も以下のとおりとします.

t2-xx.yy.zz.aa(SNAPSHOT|GA)

xx/yy/zzの意味は上記と同じです.aaはリビジョンアップのための数字で、どのように使ってもかまいません.

6.2.リリース

6.2.1.シングルリリースプラン

t2では安定版/開発版とわけることはせず、安定版のみのリリースします.

6.2.2.リリース間隔

リリース間隔は、
1.0以前

マイナーバージョンアップで1-2ヶ月に1度を目安とする.
バグフィックスバージョンアップは適宜行う.

1.0以後

マイナーバージョンアップで4ヶ月に1度を目安とする.
バグフィックスバージョンアップは適宜行う.

6.2.3.互換性について

互換性は以下の部分で守られます.

- @Publishedとついたインタフェース・クラスは、公開インタフェース/公開クラスとして互換性を保つ.
- 公開インタフェース/公開クラスはメソッドの追加する分には問題ないが、それ以外での変更の場合はマイナバージョンリリース以上になる.
- 公開インタフェース/公開クラス上の使われなくなったメソッドは@Deprecatedとするのみとする.

6.3.リリースプラン

現在は以下のようなリリースプランになっています.背景色が赤がメジャーバージョン、黄色がマイナーバージョンです.
予定は場合により変更される可能性があります.

バージョン	コードネーム	概要	リリース状況
0.2	Menu	1.0へ向けての機能の洗い出し.これ以上大きく機能は増やさない.	リリース完了
0.3	Diet	一旦開発ブランチへ全て保管し、機能を最小限まで削る.アーキテクチャの基本を確定する.インタセプタ処理の差込ポイントの考慮.	リリース完了
0.4	Core	HTTP/HTTPSを主としたアプリケーションの機能.エラー処理・テスト支援機能	リリース完了
0.4.0	-	フォームデータを丸ごと受け取る、@Formの実装	最新版
0.5	Rest	REST/Ajax機能の強化.Ajaxリクエストの判別.	開発中
0.5.0	-	パッケージ構成の変更	

0.5.1	-	@Ajax機能 POX(Plain Old XML)サポート	
0.5.2	-	HTPTヘッダ制御機能	
0.6	Star	Flex/AIRとの接続機能	未開発
0.6.0	-	@AMFによるBlazeDS経由でのリクエスト受付	
0.6.1	-	AMFリクエストを受け取る部分の拡張ポイントの作成	
0.7	Silver	Silverlightとの連携.SOAP通信予定	
0.8		未定	
0.9		未定	

尚、リリースプランは下記のGoogleCalendar?でいつでも確認できます.
http://www.google.com/calendar/embed?src=8f2qarhl1446e64qbj295p5b1o%40group.calendar.google.com&ctz=Asia/Tokyo

バグ修正はこのリリースプランに則らず順次行います.

7.ポリシー

7.1.T2がやること (Things T2 does.)

-Webにつながること -Connect to the WEB-

T2はWebにつながる・つなげる部分に特化したWebフレームワークです. Webフレームワークのコア部分であるクライアントから来たHTTPリクエストを処理する接続部分にこだわって作成されています.
Viewは原則選びませんが、それ以外のことはなるべくやらないように意図的に設計されています. T2はそれ単体で使われることだけでなく、使って下さるユーザさんが自由にその上に独自のフレームワークを 重ねやすいようにしています.

-ステートレスを基本方針とする

T2 はステートレスなフレームワークです. ここでいうステートレスとはなるべくフレームワーク側で状態を持たないという意味です. なぜステートレスにしたかというと、ステートレスなフレームワークはステートフルなフレームワークのミニマルセットだからです. ステートレスフレームワークの上にステートフルなものを構築するのは比較的容易ですが、 その逆のステートフルなフレームワークをステートレスに見せるのは難易度が高いです. よってより自由度の高いステートレスを基本とし、ユーザが必要であればステートフルに仕上げることも出来るようにしました.

-コードを小さくシンプルに保つ

保守しやすく、何かあってもユーザが修正しやすいコードを維持します.

-フレームワークのフレームワークの位置を目指す

単体でよりもむしろラップして使われることを意識したつくりにします. T2フレームワークはそれ単体で使うときよりも何かとあわせて使うときに その威力を発揮できるようなつくりを目指します.

-極力美しい**URL**（**REST**ライク）を実現しやすくする.ただし強制はしない

URL の美しさ・見易さ・直感性はWebアプリケーションにとって非常に大事だと考えています. そのため、できるだけRESTライクなURLを使えるように機能を洗練させます. ブックマーカブルな綺麗なURLを使ったシステム開発がT2の目指すべき目標のひとつです. ただし、そのような綺麗なURLは強制ではなくあくまで実現方法の一つとして捉えています.

-保守を考えた足跡を残すような開発を可能にすること

シ ステム開発を考えた場合に、保守のことを無視することは出来ません. 保守性、これがT2が重要視するキーワードの一つです. 保守を担当する開発者がT2で開発された実装成果物を引き継いだときに、 なるべく違和感なく、わかりやすいようなコードを開発できるようなフレームワークにしていきます.
具体的に目指すものとしては以下のようになります.

- 目に見えないものより見えるものを優先する.例えば規約よりもアノテーションを使う
- 侵略的でない方法で制約をあたえ、出来るだけ枠組みの中で開発できる
- フレームワーク自体の互換性を保ち、長いスパンで同一フレームワークを動かすことが出来る

-プラグインによる拡張機能の枠組みを提供する

Webフレームワークは、そのユーザごと・業務ごと・開発するチームごとによって必要な機能は千差万別です. そのため、フレームワークとしてある程度の拡張性を維持するためにT2にプラグインを差し込める枠組みを準備します.

-マルチビューフレームワーク

T2 では使うビューテクノロジを自由に選ぶことが出来ます. 例えば、JSPでの開発でも使えますし、FlexやAIR、SilverlightなどのRIA(Rich Internet Application)開発でも 利用できます.このようにビューテクノロジの種別を問わないで開発可能なのがT2です.

7.2.T2がやらないこと (Things T2 does not do.)

-バリデーション

バ リデーションはユーザの方が適切に選択して、Pageクラスのメソッド内に記述する必要があります. バリデーションはフレームワーク側で勝手にやるべきことではなく、ユーザが最も手になじむバリデーションの 仕組みを使って適切に行うべきと私達は考えます（ただし今後の予定として、バリデーションのユーティリティ程度は 準備するかもしれません）.
例えばOvalフレームワークやCommons Validator、または自作のバリデーションユーティリティなどが 良い選択ではないでしょうか.

-フレームワークによる自動コンバージョン

これは全ての場合ではないですが、原則としてT2ではフレームワーク内部による 自動コンバージョンは行いません.ただしそれを徹底してしまうとあまりに使い勝手が悪くなるので、 適切なバランスを絶えず見つけるようにしていきます.

- リッチコンポーネントの開発

実際にどのようなViewが使われるのかをT2ではあまり意識していません. Viewのコンポーネントは各プロジェクトによってその特性は異なります. そのため、私達T2プロジェクトではなるべくViewコンポーネント自体はもたないようにしています.