Wicket：一种构建和测试动态 Web 页面的简化框架

级别： 中级

Kumarsun Nadar, 高级资深软件工程师, IBM

2008 年 12 月 04 日

Wicket 提供了一种面向对象的方式来开发基于 Web 的动态 UI 应用程序。由于 Wicket 是纯 Java™ 和 HTML 代码，所以您可以充分利用自己的 Java 知识来基于 Wicket 编写应用程序，这会极大减少您的开发时间。本文概述了 Wicket 并介绍了如何利用 Wicket 以一种无干扰的简化方式来快速构建基于 Web 的应用程序。

概览

Wicket 是最近才启用的 Java Web 开发框架。它是一种开源、轻量、基于组件的框架，这让 Wicket 迅速从开发 Web 应用程序的常用方法中脱颖而出。Wicket 力图通过支持基于纯 HTML 的模板来清晰地界定 HTML 页面设计人员和 Java 开发人员之间的角色界线，此模板可使用任何的 WYSIWYG HTML 设计工具构建，并且经稍许修改就可以具备动态特征。

与其他框架类似，Wicket 也构建在 Sun Microsystems 的 servlet API 之上。不过，与基于 Model-View-Controller (MVC) 模型（比如 Struts）的其他框架不同，Wicket 可以让您从处理请求/响应对象的任务中解脱出来，而这些任务是诸如 servlet 这类技术所固有的。去掉这些任务后，Wicket 让您能将精力更多地集中于应用程序的业务逻辑。

作为一个 Wicket 开发人员，应该考虑构建有状态的可重用组件，而不是构建用来处理请求/响应对象的控制器并且同时还要担心多线程问题。与构建控制器或动作类相反，您创建的是一个页面，在这个页面上放置组件，然后定义每个组件如何响应用户输入。

HelloWorld 示例

要真正展示使用 Wicket 开发基于 Web 的应用程序的简便性，不妨先来开发一个简单的 “Hello World” 示例。在 Wicket 开发一个动态页面通常只会涉及创建如下两个工件：

- HTML 模板
- Java 页面类

注意：必须确保实际的 HTML 文件和页面类名称是相同的（例如，HelloWorld.html 和 HelloWorld.java）而且二者均处在 CLASSPATH 上的相同位置。而且最好要将二者置于相同的目录内。

Wicket 的 Java 页面类

Java 页面类是一个 Wicket 约定，不能与 JavaServer Pages (JSP) 相混淆。在这种上下文环境中，Java 页面类只是处理 Web 页面上动态内容的简单 Java 代码。在本例中，就是对应于 HelloWorld.html 的 HelloWorld.java。

HTML 模板（HelloWorld.html）

清单 1 中所示的是 HelloWorld 示例的模板文件。

清单 **1. HelloWorld.html**

```
<html>
        <head><script type="text/javascript" ></script></head>
        <body bgcolor="#FFCC00">
                <H1 align="center">
```

```
                        <span wicket:id="message">Hello World Using Wicket!</span>
                </H1>
        </body>
</html>
```

要制作动态的 Web 页面，需要确定页面的动态部分并告知 Wicket 使用组件来呈现这些部分。在 清单 1 中，我想要获得动态的消息，于是我使用了 span 元素来标记该组件，使用 wicket:id 属性来标示该组件。

Java 页面类（HelloWorld.java）

清单 2 所示的是 HelloWorld.java 示例的页面类。

清单 **2. HelloWorld.java**

```
package myPackage;

import wicket.markup.html.WebPage;
import wicket.markup.html.basic.Label;

public class HelloWorld extends WebPage
{
        public HelloWorld()
           {
                        add(new Label("message", "Hello World using Wicket!!"));
           }
}
```

页面类中 label 组件的 ID（"message"）必须要与模板文件内此元素的 Wicket ID（wicket:id="message"）相匹配。Wicket 的 Java 页面类包含 Web 页的所有动态行为。而 HTML 模板和页面类之间是一对一的关系。

最后，需要创建一个 Application 对象，当应用程序由 Web 容器加载时，该对象是起始点，而且它还是进行应用程序初始化设置和配置的地方。比如，可以通过覆盖 getHomePage() 方法并返回对应于应用程序主页的页面类来定义应用程序的主页，如清单 3 所示。

清单 **3. HelloWorldApplication.java**

```
package myPackage;

import wicket.protocol.http.WebApplication;

public class HelloWorldApplication extends WebApplication {

           protected void init() {
        }

           public Class getHomePage() {
                        return HelloWorld.class;
        }
}
```

此外，还可以修改或覆盖默认应用程序设置，具体方法是覆盖 `init()` 方法并随后调用 `getXXXSettings()` 来检索可更改的 `Settings` 对象的一个接口。由这些方法返回的接口如表 1 所示，这些接口可用来配置应用程序的框架设置：

表 1 给出了设置的示例列表，这些设置可应用于 `Application` 类的应用程序级别。

表 **1.** 应用程序级设置

方法	使用目的
getApplicationSettings	应用程序的应用程序级别设置
getDebugSettings	应用程序与调试相关的设置
getExceptionSettings	应用程序的异常处理设置
getMarkupSettings	应用程序与标记（Markup）相关的设置
getPageSettings	应用程序与页面相关的设置
getRequestCycleSettings	应用程序与请求周期相关的设置
getSecuritySettings	应用程序与安全性相关的设置
getSessionSettings	应用程序与会话相关的设置

表 2 给出了一些示例，展示了如何将应用程序级别的设置应用于 `Application` 类。

表 **2.** 应用程序级别设置的例子

示例	用途
getApplicationSettings().setPageExpiredErrorPage(<Page class>)	定义在页面因会话超时而终止时应该显示的通用页面
getMarkupSettings().setDefaultMarkupEncoding("UTF-8")	设置 markup 格式以便呈现
getSecuritySettings().setAuthorizationStrategy(<IAuthorizationStrategy Instance>)	设置可用于应用程序的授权策略

web.xml 配置文件

最后，若要加载应用程序并让其可用，需要定义这个 Wicket servlet 类，在 web.xml 配置文件内用参数的形式为其传递应用程序类名，如清单 4 所示。

清单 **4. web.xml** 配置文件

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app
      PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
      "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
        <display-name>Wicket Hello World Example</display-name>
        <servlet>
                <servlet-name>HelloWorldApplication</servlet-name>
                <servlet-class>
                        wicket.protocol.http.WicketServlet
                </servlet-class>
                <init-param>
                        <param-name>applicationClassName</param-name>
                        <param-value>myPackage.HelloWorldApplication</param-value>
                </init-param>
        </servlet>
        <servlet-mapping>
                <servlet-name>HelloWorldApplication</servlet-name>
```

```
                <url-pattern>/hello/*</url-pattern>
        </servlet-mapping>
</web-app>
```

此应用程序现在可以包装成一个 War/Ear 文件并被部署到任何基于 Java Platform, Enterprise Edition（Java EE）的 servlet 容器，比如 Tomcat 或 WebSphere®，并可通过 URL http://<serverName:port>/warfileName/hello/ 调用，只需用具体的值代替其中的 servername 和 warfilename 即可。在本例中，我调用的是 http://localhost:8090/sample/hello，如图 1 所示。

图 **1.** 示例 **HelloWorld Wicket** 应用程序

回页首

Wicket 生命周期

如能对 Wicket 生命周期有深入的了解将非常有利于更有效地使用 Wicket。这个生命周期包含如下一些步骤:

- 应用程序加载
- 请求处理
- 呈现

应用程序加载

基于 Wicket 的应用程序可通过定义 web.xml 文件内的 Wicket servlet 加载，该文件可载入到任何基于 Java EE 的应用服务器，如清单 5 所示。

清单 **5. web.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app
      PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
      "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
        <display-name>Sample Wicket Application</display-name>
        <servlet>
                <servlet-name>SampleWicketApplication</servlet-name>
```

```
                <servlet-class> wicket.protocol.http.WicketServlet </servlet-class>
                <init-param>
                        <param-name>applicationClassName</param-name>
                        <param-value>wicket.sample.SampleApplication</param-value>
                </init-param>
        </servlet>
        <servlet-mapping>
                <servlet-name>SampleApplication</servlet-name>
                <url-pattern>/sample/*</url-pattern>
        </servlet-mapping>
        <welcome-file-list>
                <welcome-file>index.html</welcome-file>
        </welcome-file-list>

</web-app>
```

所定义的 servlet 类必须一直是 `wicket.protocol.http.WicketServlet`，而 `applicationClassName` 参数值的类型则必须是 `WebApplication`。在本例中，`SampleApplication` 扩展 `WebApplication`。无论何时，只要客户机使用 URL `/sample/*` 调用清单 5 中的应用程序，该服务器就要加载 `WicketServlet`，而这反过来又能创建应用程序类的一个实例，即 `SampleApplication`。

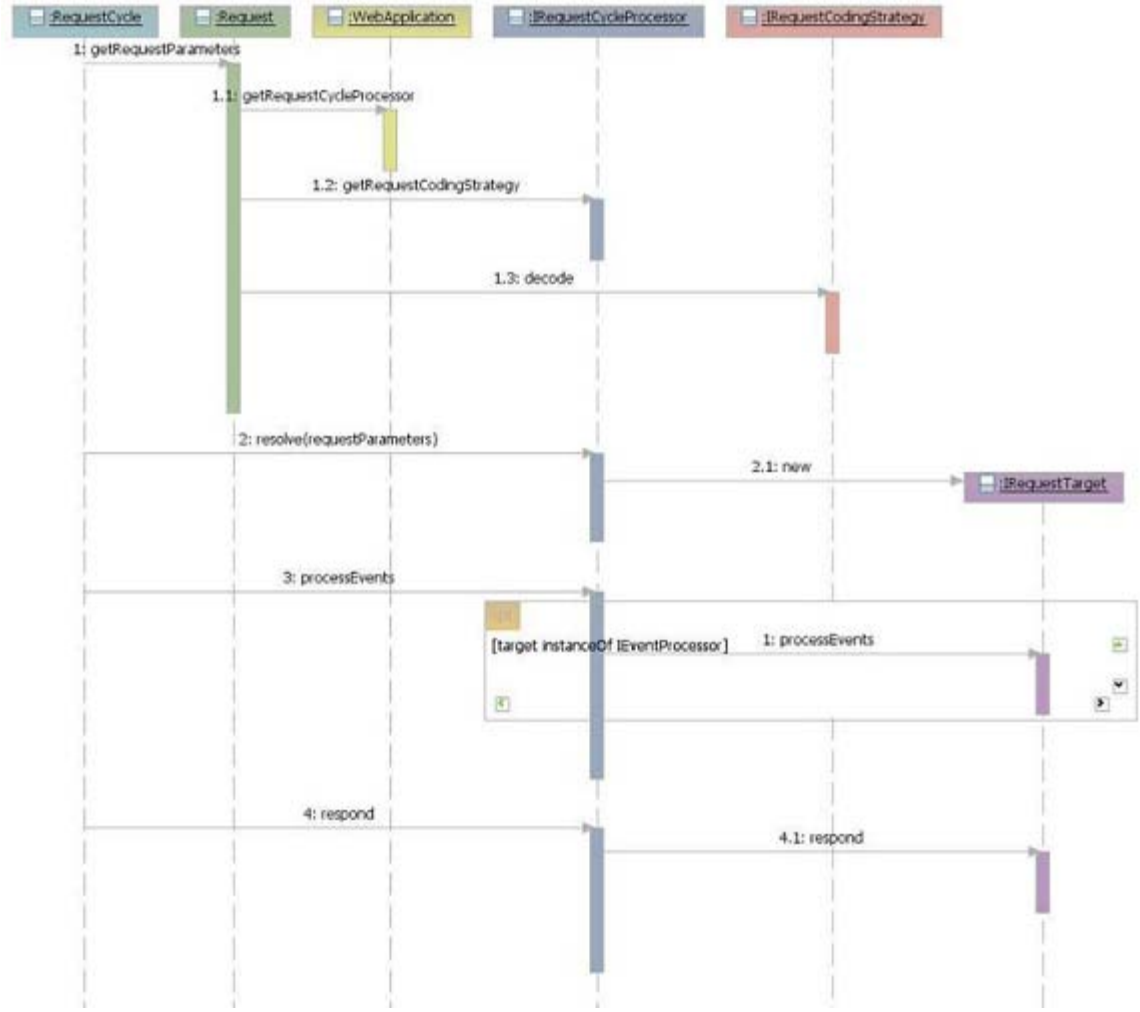
所定义的 `applicationClassName` 参数必须是扩展 `WebApplication` 的类的完全限定名。如果应用程序类未能找到、不能扩展 `WebApplication` 或不能被实例化，就会抛出 `WicketRuntimeException` 类型的运行时异常。

请求处理

应用程序类由 Wicket servlet 加载后，如果没有会话存在，它就会使用会话库创建针对 servlet 请求的一个会话。应用程序之后使用这个会话对象创建一个 `RequestCycle` 对象并委派对 `RequestCycle` 的调用去处理此请求。

之后，请求周期调用 `onBeginRequest` 方法以便让子类进行请求处理之前相应的预处理操作。请求周期会经历若干状态并会根据当前状态向请求周期处理器发出不同的指令。发出所有指令后，此请求周期即达到了其最后的状态，表明请求处理完毕，如图 2 所示（要看放大的图片，请单击 此处）。

图 **2. Wicket** 的请求处理顺序流程



请求周期处理器负责处理请求周期内的指令。该处理器会被 `RequestCycle` 所用以便以预定义的格式调用其方法：

- 调用 `IRequestTarget resolve(RequestCycle, RequestParameters)` 以获得请求的目标。例如，请求针对的可能是可被书签标记的一个页面，也可能是之前呈现的页面上的某个组件。这是请求周期处理器的主要功能之一。`RequestParameters` 对象包含所有可从 `servlet` 请求参数转换过来的可选参数并可充当这些参数的强类型的变体（参见 `IRequestCodingStrategy`）。
- `void processEvents(RequestCycle)` 的作用是处理类似组件调用这类事件，比如 `onClick()` 和 `onSubmit()` 事件处理程序。
- 调用 `void respond(RequestCycle)` 是为了创建响应，即生成 `Web` 页面或是进行重定向。
- 只要在事件处理过程或响应阶段发生未被捕获的异常，`void respond(RuntimeException , RequestCycle)` 都会被调用以便生成合适的异常响应。

呈现

页面通过呈现与它相关的标记（页面的 `HTML` 文件）来呈现页面自身。正如 `MarkupContainer`（页面的超类）会为这个相关的标记遍历标记流一样，它会按 `ID` 搜索与此 标记内的标签（`tag`）相关联的组件。由于 `MarkupContainer`（在本例中，是一个页面）已经通过 `onBeginRequest()` 构造并初始化，每个标签的子标签都应该在容器内可用。组件检索完毕后，会调用它的 `render()` 方法。

`Component.render()` 遵循如下步骤来呈现一个组件：

1. 决定组件的可见性。如果组件不可见，`RequestCycle` 的 `Response` 就会被更改为 `NullResponse.getInstance()`，它是丢弃输出的一种响应实现。
2. 调用 `Component.onRender()` 的目的是让此组件的呈现实现开始呈现此组件。
3. 所有组件模型均被分离以减少组件层次结构的大小，并防止它被跨集群复制。

`renderComponent` 方法获得此标记流中的下一个 `Tag` 的可更改版本并调用 `onComponentTag(Tag)` 以便子类能够修改此标签。子类更改了标签之后，会被 `renderComponentTag(Tag)` 写出到此响应，而此标记流则前进到下一个标签。

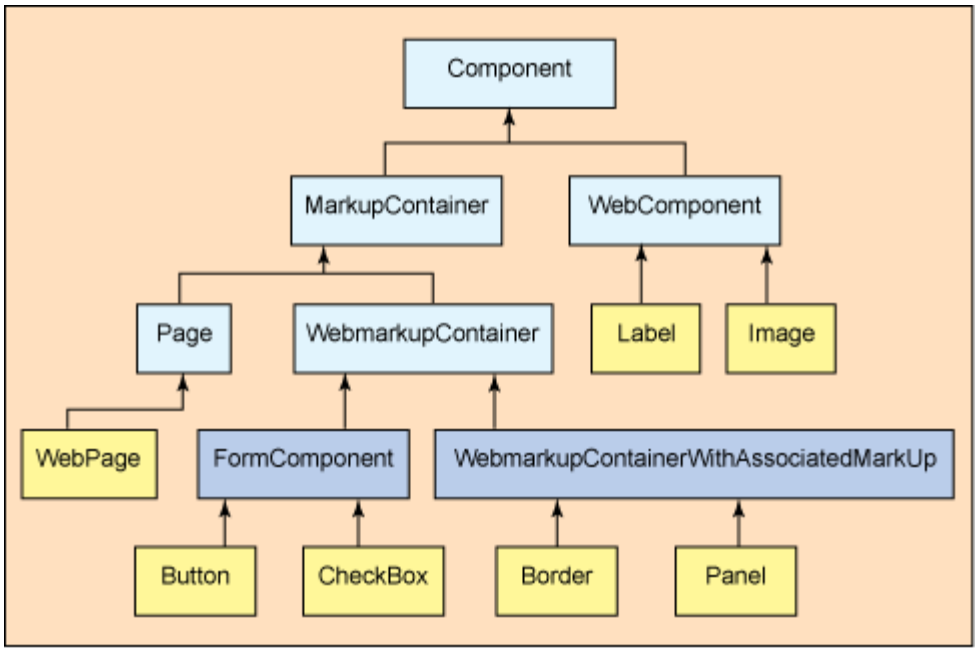
接下来，调用 `onComponentTagBody()`，传递已经作为打开标签写出的 `MarkupStream` 和 `ComponentTag`。这就让组件可以为了生成此组件标签的主体而进行所有必要的操作。子类能够在 `onComponentTagBody()` 内调用的一个操作是 `Component.replaceComponentTagBody()`，它用一个任意字符串替代此组件主体内的那个标记。最后，此框架为此组件的打开标签写出适当的结束标签。

创建定制组件

Wicket 的主要特性之一是它提供了开发定制组件的便利性。Wicket 提供了一个非常灵活的组件/编程模型，利用此模型可以轻而易举地开发定制组件。定制组件可以是 HTML 字段的格式，也可以是可在页面内使用的面板的格式。在基于 Web 的应用程序中，可重用组件的常见格式是页眉、页脚、导航栏等。

首先，我们来看看 Wicket 自身的组件层次结构，该结构很容易用来构建一个新的定制组件，也可以对它进行扩展以便添加新特性。

图 **3.** 组件的层次结构



组件

`Component` 位于层次结构的最顶端，充当所有组件的抽象基类。它提供了多种特性，比如

- Identity：必须是非空 ID，并且此 ID 在容器内惟一并能通过 `getID()` 检索到
- Model：保存要被作为 HTML 响应而呈现的数据
- Attributes：可被添加到任何组件以处理与此组件关联的标记标签

WebComponent

`WebComponent` 充当诸如标签和图像这类简单的 HTML 组件的基类。

MarkupContainer

`MarkupContainer` 保存所有子组件并且自身没有标记。子组件可通过调用 `add()` 和 `replace()` 方法而被添加或替换，并且可以使用 OGNL 注释进行检索。例如，调用 `get("a.b")` 将会返回 ID 为 `"b"` 且父组件 ID 为 `"a"` 的组件。

WebMarkupContainer

它充当 HTML 标记和组件的容器。它非常类似于基类 `MarkupContainer`，只不过标记类型被定义成 HTML。

WebMarkupContainerWithAssociatedMarkup

它扩展 WebMarkupContainer 并提供附加特性来处理页眉标签。

Panel

Panel 是一个可重用组件，用来保存标记和其他组件。通过扩展此类可以创建可重用的定制组件。

Border

Border 组件具备自身的关联标记并可被用来定义其关联标记文件的一部分，以呈现其边界。

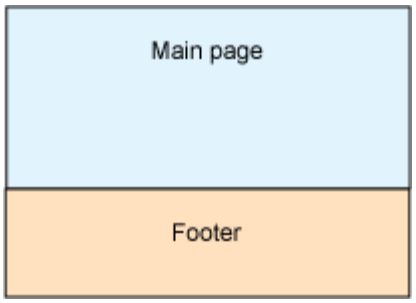
Page

Page 充当所有页面的抽象基类并可包含 Component 的任意树。所有生命周期事件，比如 onBeginRequest、onEndRequest() 和 onModelChanged() 都可被 Page 的子类覆盖，这些子类可处理这些事件。

简单了解了 Wicket 的组件层次结构后，现在可以来看看如何通过扩展 Wicket 的 Panel 组件构建您自己的定制组件。

借助图 4，可大致了解一下如何使用 Wicket 编写示例定制组件（如页脚）的代码。

图 **4. Footer** 组件的布局



一个具有直观表示的定制 Wicket 组件通常包含如下工件：

- 一个 HTML 模板（Footer.html），如清单 6 所示
- 可选的 JavaScript、样式表或图像
- 相应的 Java Component 类（Footer.java），这个类扩展标准 Wicket 发布版自带的几个组件基类中的一个，如清单 7 所示。

清单 **6. Footer.html**

```
<wicket:panel>
        <hr>
        Copyright <span wicket:id="year">2008</span>. IBM Inc. All rights reserved.
</wicket:panel>
```

清单 **7. Footer.java**

```
import java.util.GregorianCalendar;
import wicket.markup.html.basic.Label;
import wicket.markup.html.panel.Panel;

public final class Footer extends Panel {

  public Footer(String id) {
```

```
    super(id);
    add(new Label("year", "" + new GregorianCalendar().get(GregorianCalendar.YEAR)));
  }

}
```

注意：清单 6 和 清单 7 中的工件、HTML 工件以及该组件（一个 Java 类）的服务器端的表示通常位于同一个包内。

嵌入/使用组件

要使用组件 — 在定义完之后 — 只需在所需的 Page 类（即 <Page>.java）文件（例如 Home.java，如清单 8 所示）内对之进行实例化，然后通过将此定制组件的 ID 嵌入到相应的模板（<Template>.html）文件（比如 Home.html，如清单 9 所示）内对它进行调用。

清单 **8. Home.java**

```
import wicket.markup.html.WebPage;
public class Home extends WebPage {

  public Home() {
    add(new Footer("footer"));
  }

}
```

清单 **9. Home.html**

```
<html>
<head></head>
<body>
        Body of Home page.
        <span wicket:id="footer">Footer Info</span>
</body>
</html>
```

Wicket 验证

Wicket 支持客户端和服务器端的表单验证。验证是通过内置的验证器实现的。Wicket 自带很多验证器，比如 NumberValidator、StringValidator、PatternValidator、DateValidator 和 RequiredValidator。

此外，还可以编写一个定制的验证器来执行那些未内置到 Wicket 内的验证。对于服务器端验证而言，在表单提交后，Wicket 就会遍历置于表单内的所有的表单组件并对组件输入执行所有的关联验证器。在处理过程中，只要验证器抛出任何错误消息，Wicket 都会收集这些错误消息。之后，FeedbackPanel 组件显示所有收集到的错误消息。

内置验证器

Wicket 提供了处理验证的一种更为简便的方式。通过 Wicket，当字段组件在页面文件内创建的时候，就可以在字段组件上设置一个验证器，比如在页面内创建一个要求用户进行输入的强制字段，如清单 10 所示。

清单 **10.** 将 **TextField** 标记为强制输入字段

```
TextField firstNameComp = new TextField("firstName");
        firstNameComp.setRequired(true);
```

在 Wicket 内，FeedbackPanel 组件呈现页面内所有与表单相关的错误。实际上，FeedbackPanel 显示与 Page 内包含的组件相关联的所有类型的反馈消息。当然，也可以只过滤那些需要显示的消息类型（信息、错误、调试、警告等）。此组件可被添加到页面，如清单 11 所示。

清单 **11.** 在页面类内添加 **FeedbackPanel**

```
add(new FeedbackPanel("feedBack"));
```

对 "feedback" 组件的引用也需要添加到 HTML 标记以显示可能存在的验证错误，如清单 12 所示。

清单 **12.** 在模板文件内嵌入 **FeedbackPanel**

```
<span wicket:id="feedback"></span>
```

定制验证器

如果表单级验证不能通过使用内置验证器得到处理，就可以创建一个定制验证器，方法是建立 AbstractFormValidator 类的子类并使用一个构造函数，把要验证的表单组件作为该函数的参数。定制验证器需要在表单实例化的时候添加到此表单。

举个例子，假如需要确保一个表单内两个给定字段都不为空。由于此目的不能通过使用内置验证器实现，所以需要编写一个定制的验证器，如清单 13 所示。

这个定制验证器通过扩展 AbstractFormValidator 类创建，如清单 13 所示。

清单 **13. EitherInputValidator.java**

```
import java.util.Collections;
import wicket.markup.html.form.Form;
import wicket.markup.html.form.FormComponent;
import wicket.markup.html.form.validation.AbstractFormValidator;
import wicket.util.lang.Objects;

public class EitherInputValidator extends AbstractFormValidator {

        /** form components to be validated. */
        private final FormComponent[] components;

        public EitherInputValidator(FormComponent f1, FormComponent f2) {
                if (f1 == null) {
                                throw new IllegalArgumentException(
                                        "FormComponent1 cannot be null");
```

```
                }
                if (f2 == null) {
                                throw new IllegalArgumentException(
                                                "FormComponent2 cannot be null");
                }
                components = new FormComponent[] { f1, f2 };
        }

        public FormComponent[] getDependentFormComponents() {
                return components;
        }

        public void validate(Form form) {
                // we have a choice to validate the type converted values or the raw
                // input values, we validate the raw input
                final FormComponent f1 = components[0];
                final FormComponent f2 = components[1];
                String f1Value = Objects.stringValue(f1.getInput(), true);
                String f2Value = Objects.stringValue(f2.getInput(), true);
                if ("".equals(f1Value) || "".equals(f2Value)) {
                                final String key = resourceKey(components);
                                f2.error(Collections.singletonList(key), messageModel());
                }
        }
}
```

注意：必须调用 `FormComponent.getInput` 获得需要验证的值，而不是获得相关的模型。这是因为模型是在验证后才更新的。

要使用定制验证器，必须将其添加到表单，并为它传递需要被验证的那些字段元素，如清单 14 所示。

清单 **14.** 在页面类内使用定制验证器

```
        Form myForm = new Form("form");
        FormComponent f1 = new TextField("firstName");
        myForm.add(f1);
        FormComponent f2 = new TextField("lastName");
        myForm.add(f2);
        myForm.add(new EitherInputValidator (f1, f2));
```

最后，如果验证失败，需要将所显示的消息添加到属性文件，如清单 15 所示。

清单 **15.** 属性文件内的验证消息

```
EitherInputValidator = Please enter data for either '${input0}' from ${label0}
or '${input1}' from ${label1}.
```

联合使用 Ajax 和 Wicket

通过 Ajax（Asynchronous Javascript And XML），不仅可以构建富 UI，还能构建高性能的应用程序，因为启用了 Ajax 的应用程序只更新页面，并不导致整个页面的刷新。其结果是反馈和用户体验十分类似于桌面应用程序。而这要归功于浏览器的 XMLHttpRequest 实现，它能让客户机与服务器进行异步通信并能基于从服务器收到的响应（使用 HTML DOM API）动态处理页面内容。

Wicket 解决了用户必须要发送和处理服务器和浏览器之间的数据的问题。与向客户机发送数据相反，Wicket 在服务器端呈现组件并发送所呈现的那个标记。仅此可能还不够，开发 Ajax 行为要更为简单和迅速。例如，如果想要使用 Ajax 更新一个显示用户信息的面板，所需做的只是告知 Wicket 此面板应该更新。而根本无需使用 JavaScript 在客户机上解析由 Wicket 返回的响应，亦无需在客户机上处理 DOM。

在 Wicket 内，基于 Ajax 的请求被建模为一种行为。Wicket 具有 IBehaviour 接口的几个实现，比如 AbstractDefaultAjaxBehaviour 和 AjaxFormComponentUpdatingBehaviour，它们进行内部处理并调用 AjaxRequestTarget 内传递的 respond() 方法。此目标对象获取为响应 Ajax 请求而需要发送回浏览器的实际内容。AjaxRequestTarget 只呈现那些需要添加给它的组件，而内置的 JavaScript 工件则通过初始化 HTML outerHTML 属性来重新呈现所添加的组件。

让我们以 Ajax AutoCompleteName 为例，基于用户输入，应用程序将会预测用户选择的几种可能性。

应用程序由一个具备文本字段元素的页面（如清单 16 所示）、内存中的一个预定义的名称（也可以是动态的）列表以及用来在用户开始输入文本时显示可能值列表的一个 Ajax 行为组成。

清单 **16. HTML** 页面模板（**AutoCompleteName.html**）

```
<html>
  <head>
        <wicket:head>
                <style>
                  div.wicket-aa {
                        font-family: Verdana,"Lucida Grande","Lucida Sans
Unicode",Tahoma;
                        font-size: 12px;
                        background-color: white;
                        border-width: 1px;
                        border-color: #cccccc;
                        border-style: solid;
                        padding: 2px;
                        margin: 1px 0 0 0;
                        text-align:left;
                  }
                  div.wicket-aa ul {
                    list-style:none; padding: 2px; margin:0;
                  }
                  div.wicket-aa ul li.selected {
                    background-color: #FFFF00; padding: 2px; margin:0;
                  }
                </style>
        </wicket:head>
  </head>

  <body bgcolor="#FFCC00">
        <br>
        <br>
        <form wicket:id="form">
```

```
                <b>Name :</b> <input type="text" wicket:id="name" size="60" />
        </form>
  </body>
</html>
```

图 **5. Wicket Ajax** 示例（加载中）

在 Wicket 中，这个自动完成行为是在类 `AutoCompleteBehaviour` 内建模的。需要扩展此类并为方法 `Iterator getChoices(String input);` 提供实现，以便基于输入返回这个可能的用户列表（清单 17）。

清单 **17.** 页面类（**AutoCompleteName.java**）

```
package myPackage;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Iterator;
import java.util.List;

import wicket.extensions.ajax.markup.html.autocomplete.AutoCompleteTextField;
import wicket.markup.html.WebPage;
import wicket.markup.html.form.Form;
import wicket.model.Model;

public class AjaxWorld extends WebPage {
        private List names = Arrays.asList(new String[] { "Kumarsun", "Ramkishore",
          "Kenneth", "Kingston", "Raju", "Rakesh", "Vijay", "Venkat", "Sachin" });

        public AjaxWorld() {

      Form form = new Form("form");
      AutoCompleteTextField txtName = new AutoCompleteTextField("name", new Model()){
```

```
         protected Iterator getChoices(String input) {
                                 List probables = new ArrayList();
                                 Iterator iter = names.iterator();
                                 while (iter.hasNext()) {
                                         String name = (String) iter.next();
                                         if (name.startsWith(input)) {
                                                         probables.add(name);
                                         }
                                 }
                                 return probables.iterator();
                         }
        };
        form.add(txtName);
        add(form);
   }
}
```

图 **6. Wicket Ajax** 示例（**Ajax** 响应）

I18N 支持

Wicket 通过读取来自特定于本地语言环境（locale）属性文件的消息提供对 I18N 的支持。而这则是通过在页面类内使用 `StringResourceModel` 类或通过在 HTML 标记文件内使用 `<wicket:message>` 标签实现的。`StringResourceModel` 也可用来格式化要显示的本地化了的消息。

I18N 使用 <wicket:message> 标签

页面内需要显示的标签和其他信息可使用 `<wicket:message>` 标签本地化，而不是硬编码这个标签，如清单 18 所示。

清单 **18.** 模板文件（**MyPage.html**）

```
<html>
<head>
<title></title>
</head>
```

```
<body>
        <form wicket:id="myForm">
                <wicket:message key="first-name">First Name</wicket:message>
        </form>
</body>
</html>
```

只要 Wicket 遇到 `<wicket:message>` 标签，它就会基于在 HTTP 请求对象上设置的本地语言环境从特定于本地语言环境的属性读取键值。

I18N 使用 StringResourceModel 类

如果不在模板页内使用 `<wicket:message>` 标签，也可以直接在页面内使用 `StringResourceModel` 来基于输入键检索本地化了的消息，如清单 19 所示。`StringResourceModel` 类的优势是可以在呈现之前灵活地格式化消息。

清单 **19.** 页面类（**MyPage.java**）

```
public class MyPage extends WebPage {
        public MyPage() {
                Form form = new MyForm("myForm");
                String firstNameLabel = new StringResourceModel("first-name").getString();
                form.add(new Label("firstName", new Model(firstNameLabel)));
                add(form);
        }
}
```

资源包搜索顺序

通常，在 Java i18n 系统中，消息由 locale 查找。locale 会自动从 User Agent 字段内的 HTTP 请求头获得，并且在 Wicket `WebRequest` 对象内设置。不过，它也可以用 `getSession().setLocale(Locale.US)` 显式地设置。

如果您需要为某个特定的组件覆盖这个 locale，可以只覆盖该组件上的 `getLocale()`，之后，它将只能由该组件及其子组件使用。如果客户机没有提供想要的 `Locale`，就会使用 Java 代码默认的 `Locale`。

Wicket 搜索所有资源文件中名称与组件层次结构内的组件相同的文件，最后搜索的是应用程序。消息一旦找到，Wicket 就会中止搜索过程。所以，若 Wicket 想要查找一个 `MyPanel` 内使用的消息，其中 `MyPanel` 包含在 `MyPage` 之下的 `MyForm` 内，而应用程序的名称为 `MyApplication`，Wicket 将会这样查找:

- MyPanel_locale.properties，……，然后是 MyPanel.properties
- MyForm_locale.properties，……，然后是 MyForm.properties
- MyPage_locale.properties，……，然后是 MyPage.properties
- MyApplication_locale.properties，……，然后是 MyApplication.properties (..)

实际上，它还更进了两步。Wicket 还将为了 `MyPanel`、`MyForm`、`MyPage` 和 `MyApplication` 的基类而查看属性文件。若 `MyPanel` 直接继承自 `Panel`、`MyForm` 直接继承自 `Form`、`MyPage` 直接继承自 `Page`、`MyApplication` 直接继承自 `Application`，Wicket 将会查看:

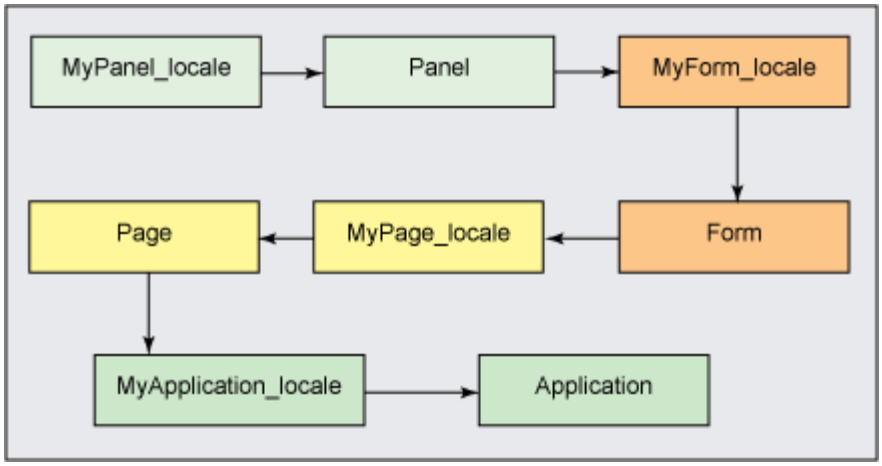
- MyPanel_locale.properties，……，然后是 MyPanel.properties
- Panel_locale.properties，……，然后是 Panel.properties
- MyForm_locale.properties，……，然后是 MyForm.properties
- Form_locale.properties，……，然后是 Form.properties
- MyPage_locale.properties，……，然后是 MyPage.properties

- Page_locale.properties，……，然后是 Page.properties
- MyApplication_locale.properties，……，然后是 MyApplication.properties (..)
- Application_locale.properties，……，最后是 Application.properties (..)

如果所有前面的步骤都失败，Wicket 将会默认使用 `labelId` 作为 `Label`。

注意：如果 `MyForm` 作为 `MyPage` 内的内部类建模，那么 wicket 将会查找名为 MyPage$MyForm.properties 的资源文件。所以，一种最佳的做法是为站点范围内的消息使用 MyApplication.properties 并在任何其他的属性文件内覆盖它们。

图 **7.** 资源包搜索顺序



对 `Wicket` 页面进行单元测试

Wicket 通过使用内置的模仿对象框架提供对容器外单元测试的支持。该框架会确保框架以及应用程序与之交互的环境对象能按配置执行操作，即使是运行于 Java EE servlet 容器之外也应如此。这有助于提高效率，这是因为无需重启此容器，让您能集中精力对感兴趣的组件进行单元测试。

模仿对象用来单独测试代码逻辑的一部分。它们提供了一些方法来让测试控制这个虚构类的所有业务方法的行为。

Wicket 对单元测试的支持基于的是对 JUnit 框架的扩展。它的 `wicket.util.tester.WicketTester` 类提供了大量帮助方法，可帮助模仿各种用户动作（比如单击链接或表单提交）以及行为（比如页面呈现或声明错误消息的存在）。

清单 20 给出了一个示例生产页面（MyPage.java），它具有一些组件和用户动作，比如页面提交和链接单击。

清单 **20.** 页面类（**MyPage.java**）

```
import wicket.markup.html.WebPage;
import wicket.markup.html.basic.Label;
import wicket.markup.html.form.Button;
import wicket.markup.html.form.Form;
import wicket.markup.html.form.TextField;
import wicket.markup.html.link.Link;

public class MyPage extends WebPage {
        public MyPage() {
                MyForm form = new Form("myForm");
                form.add(new Label("firstNameLabel", "First Name"));
                form.add(new Label("lastNameLabel", "Last Name"));

                form.add(new TextField("firstName"));
                form.add(new TextField("lastName"));
```

```
                form.add(new Button("Submit"));
                form.add(new Link("nextPage") {
                                public void onClick() {
                                        setResponsePage(new NextPage("Hello!"));
                                }
                });
        }
}
```

测试页面呈现器的测试用例

需要做的最基本的测试是要确保每个页面都能正确呈现。该测试的成功执行（如清单 21 所示）能确保模板和页面层次结构是相匹配的。

清单 **21.** 页面呈现器测试（**MyPageRenderTest.java**）

```
import wicket.util.tester.WicketTester;
import junit.framework.TestCase;

public class MyPageRenderTest extends TestCase {

        private WicketTester tester;

        public void setUp() {
                tester = new WicketTester();
        }

        public void testMyPageBasicRender() {
                WicketTester tester = new WicketTester();
                tester.startPage(MyPage.class);
                tester.assertRenderedPage(MyPage.class);
        }
}
```

测试页面组件的测试用例

`WicketTester` 类具备内置的方法来验证给定的页面具有所有必需的组件。使用它的 `assertComponent()` 方法并为其传递组件路径和组件类型，如清单 22 所示。所给定的路径需要与它所嵌入的页面相关。

清单 **22.** 页面组件测试（**MyPageComponentsTest.java**）

```
import junit.framework.TestCase;
import wicket.markup.html.form.TextField;
import wicket.util.tester.WicketTester;

public class MyPageComponentsTest extends TestCase {
```

```
        private WicketTester tester;

        public void setUp() {
                tester = new WicketTester();
        }

        public void testMyPageComponents() {
                WicketTester tester = new WicketTester();
                tester.startPage(MyPage.class);

                // assert rendered field components
                tester.assertComponent("myForm:firstName", TextField.class);
                tester.assertComponent("myForm:lastName", TextField.class);

                // assert rendered label components
                tester.assertLabel("myForm:firstNameLabel", "First Name");
                tester.assertLabel("myForm:lastNameLabel", "Last Name");

        }
}
```

测试 OnClick 用户动作的测试用例

测试诸如单击链接这样的用户动作可以通过使用 WicketTester 的 `clickLink()` 方法实现，只需为之传递链接组件 ID 路径，然后再验证所呈现的组件即可，如清单 23 所示。

清单 **23.** 页面 **OnClick** 测试

```
public void testOnClickAction() {
                tester.startPage(MyPage.class);

                // click link and render
                tester.clickLink("nextPage");

                tester.assertRenderedPage(NextPage.class);
                tester.assertLabel("nextPageMessage", "Hello!");
        }

public void testNextPageRender() {
                // provide page instance source for WicketTester
                tester.startPage(new TestPageSource() {
                        public Page getTestPage() {
                                return new NextPage("Hello!");
                        }
                });

                tester.assertRenderedPage(YourPage.class);
                tester.assertLabel("nextPageMessage", "Hello!");

        }
```

测试表单提交的用户动作的测试用例

Wicket 内的表单提交可通过使用 Wicket 的 `wicket.util.tester.FormTester` 类进行测试，该类具有一些 API，可用来设置表单内的字段组件的输入值，并最终提交此表单。假设提交表单时要显示的页面是包含 “Welcome to Wicket” 消息的 Welcome.java，对表单提交的测试将类似清单 24。

清单 **24.** 页面 **OnSubmit** 动作测试

```
public void testFormSubmit ()
   {
                // Create the FormTester object
                FormTester ft = tester.newFormTester("myForm");

                // Set the input values on the field elements
                ft.setValue("firstName", "Kumar");
                ft.setValue("lastName", "Nadar");

                // Submit the form once the form is completed
                ft.submit("Submit");

                // Check the rendered page on form submission
                tester.asserRenderedPage(Welcome.class);
                // verify the message on the rendered page
                tester.assertInfoMessage(new String[]{"Welcome to Wicket"});

        }
```

结束语

像 Wicket 这样的以 Plain Old Java Object (POJO) 为中心的框架可被用来以一种无干扰的简便方式快速构建基于 Web 的应用程序。HTML 或其他的标记不会受编程代码的任何干扰和影响，这就让 UI 设计人员很容易辨别和避免框架标记。

Wicket 让开发人员能够以一种类似 Java-Swing 的方式创建页面，以便避免对 XML 配置文件的过度使用。它还提供了一种十分全面的方式来对所开发的页面进行单元测试。

回页首

下载

描述	名字	大小	下载方法
本文的示例 Wicket 代码	wa-aj-wicketsource.zip	4KB	**HTTP**

关于下载方法的信息

参考资料

学习

- 您可以参阅本文在 developerWorks 全球网站上的 英文原文。
- 阅读 Apache Wicket 文档中的 Control where HTML files are loaded from 部分。
- 浏览 Wicket 主页 以及其中的教程来更好地理解 Wicket 的工作原理。
- 访问 Wicket 示例 获得一系列能很好展示其核心功能的例子。

获得产品和技术

- 获得 Apache Wicket 并试用。
- 下载 Qwicket，它是 Wicket 框架的一个快速入门应用程序。

讨论

- 通过参与 developerWorks blogs 加入 developerWorks 社区。

关于作者

Kumarsun Nadar 目前是位于印度孟买的 IBM 印度软件实验室（ISL）WebSphere Business Service Fabric 产品团队的一名高级资深软件工程师。作为该团队的一员，他一直从事基于 Wicket 框架的 Fabric Web Tools 模型的 UI 开发。他获得了 SUN 公司 SCJP、SCWCD 和 SCBCD 认证证书并对基于 Java/J2EE 的各种客户端和服务器端技术，比如 Wicket、EJB、Hibernate、Struts 等有丰富的经验。他在业余时间喜欢观看和参加一些体育项目，比如板球和旅游杯摩托车赛。

对本文的评价

- ○太差！ (1)
- ○需提高 (2)
- ○一般；尚可 (3)
- ○好文章 (4)
- ○真棒！(5)

建议？

反馈意见

回页首

表单 联系我们的编辑团队。