

# Spring Web Flow 2.0 入门

本教程分析了 Spring Web Flow 2.0 的技术要点，并且通过创建一个示例应用程序，展示了 Spring Web Flow 2.0 的基本知识。

## 开始之前

## 关于本教程

本教程通过一个简化的购物车应用，介绍了如何使用 Spring Web Flow 2.0 来构建 Web 应用程序。本教程以讲解实例为主，为了读者更好地理解 Spring Web Flow，也有部分理论的解释。

## 先决条件

本教程要求读者具备 Java Web 应用的基本知识、熟悉 Spring Framework 的应用。

## 系统要求

运行本教程中的示例，需要下列工具：

- JDK 1.6.0+
- Spring Framework 2.5.4+ 及其依赖项
- Spring Web Flow 2.0.2
- Tomcat 6.0.0+（为支持 EL，Tomcat 须 6.0 及以上版本）
- eclipse 3.2.2+

## Spring Web Flow 2.0 新特性

Spring Web Flow 是 Spring 的一个子项目，其最主要的目的是解决跨越多个请求的、用户与服务器之间的、有状态交互问题。最新版本为 2.0，相比于 1.x 版的 Spring Web Flow，有以下几个值得注意的新特性。

- 与 Spring MVC 深度整合

Spring Web Flow 1.x 是个自成体系的框架，可与 Spring Web MVC、Struts、JSF 等 Web 框架整合。最新的 Spring Web Flow 2.0 则明确声明是基于 Spring Web MVC 的一个扩展。

- 提供了处理 Ajax 事件的能力

Ajax 事件的处理与 WebFlow 事件的处理相一致，在处理完成后，flow 即可刷新客户端相关界面代码。

- 与 JSF 整合

通过将 JSF 层层包装，最终可在 Spring Framework 和 Spring Web Flow 中使用 JSF 的各种组件。

- 与 Spring Security （原 Acegi Security ）整合

只需将某个 flow 声明为 “ secured ”，即可利用 Spring Security 来确定当前用户是否有权限运行 flow 、激发事件等等。

- 更简洁的配置

官方的数据说同一个 flow ， 2.0 版的配置比 1.x 版的配置少 50% 的 XML 代码。

- 重用更方便

Spring Web Flow 2.0 提供了 flow 的继承，重用即有的 flow 代码更加容易。

## 本教程的说明

本教程主要讨论 Web Flow 模块的使用，对其他的特性没有涉及。

## 购物车用例

要了解 Spring Web Flow 是什么东西，最好的办法莫过于查看示例，图 1 展示了一个简化的购物车的流程。

图 1 购物车示例

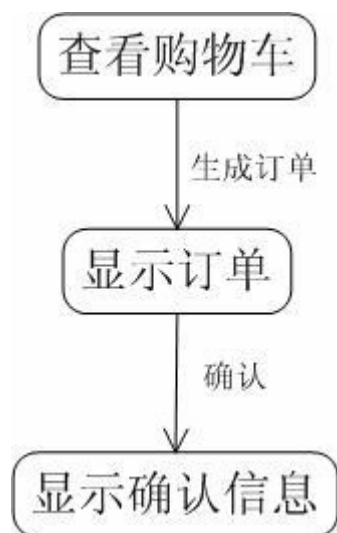


图 1 所示流程用 Spring Web Flow 2.0 的配置文件表示如下：

## 清单 1 用 Spring Web Flow 语义表达购物车流程

.....

```
<flow>
<view-state id="viewCart">
<transition on="submit" to="viewOrder"/>
</view-state>
<view-state id="viewOrder">
<transition on="confirm" to="viewConfirmed"/>
</view-state>
<view-state id="viewConfirmed">
<transition on="returnToIndex" to="returnToIndex"/>
</view-state>
<end-state id="returnToIndex"/>
</flow>
```

清单 1 省略了许多技术细节，展示的只是一个业务的流程，主要是为了让大家对 Spring Web Flow 的语义有个初始的印象。从清单 1 中，应注意到一个很重要的特征—— Spring Web Flow 语义与 Servlet API 无关。更确切地讲，Spring Web Flow 语义关注的是业务的流程，并未与 Sun 公司的 Web 规范紧密结合，这种描述是更高层次的抽象，差不多是在建模的角度来描述业务流程。

不过，Spring Web Flow 也并非只有抽象，现在还没有哪一种工具或语言可以将一个模型直接转换成相应的应用程序。Spring Web Flow 更像是抽象建模和技术细节的混血儿，相比于湮没在繁多的控制器和视图中的 Web MVC 应用来讲，Spring Web Flow 提供了如清单 1 所描述的更高层次的抽象，但同时它也整合了像 Unified EL 这样的工具来控制技术上的细节。

## Spring Web Flow 的基本元素

Flow 可看作是客户端与服务器的一次对话（ conversation ）。Flow 的完成要由分多个步骤来实现，在 Spring Web Flow 的语义中，步骤指的就是 state 。 Spring Web Flow 提供了五种 state ，分别是 ActionState 、 View State 、 Subflow State 、 Decision State 、 End State ，这些 state 可用于定义 flow 执行过程中的各个步骤。除了 End State 外，其他 state 都可以转换到别的 state ，一般通过在 state 中定义 transition 来实现到其他 state 的转换，转换的发生一般由事件（ event ）来触发。

## 什么情况下可以使用 Spring Web Flow?

前面讲了，Spring Web Flow 提供了描述业务流程的抽象能力，但对一种 Web 开发技术而言，仅有这些是不够的。同时，Spring Web Flow 是不是能够取代其他 Web MVC 技术？或者在任何情况下都应优先使用 Spring Web Flow ？要回答这些问题，先来看一下 Spring Web Flow 所着力解决的技术问题。

## Web 应用程序的三种范围

Java Servlet 规范为 Web 应用程序中用到的各种对象规定了三种范围（scope），分别是 request 范围、session 范围和 application 范围。

- request 范围中的对象是跟客户的请求绑定在一起的，每次请求结束都会销毁对象，而新的请求过来时又会重新创建对象。request 范围适合存放数据量较大的临时数据。
- session 范围中的对象是跟会话（session）绑定在一起的，每次会话结束会销毁这些对象，而新的会话中又会重新创建。HTTP 协议本身是无状态的，服务器和客户端要实现会话的管理，只能借助于一些辅助的手段，如在协议的数据包中加一些隐藏的记号，等等。session 范围适合存放本次会话需要保留的数据。
- application 范围的对象是跟应用程序本身绑定在一起，从 Servlet API 的角度来讲，就是存放在 ServletContext 中的对象，它们随着 Servlet 的启动而创建，Servlet 关闭时才会销毁。application 范围适合存放那些与应用程序全局相关的数据。

现实开发中最令人头痛的莫过于 session 范围，Java Servlet 规范指明可在 web.xml 中按如下方式配置 session 的有效时间为 100 分钟：

#### 清单 2 web.xml 中 session 的配置

```
<session-config>
<session-timeout>100</session-timeout>
</session-config>
```

然而，现实中的 session 范围更像是“鸡肋”，把大量数据放入 session 会导致严重的效率问题，在分布式的环境中处理 session 范围更是一不小心就会出错，但抛弃 session 又会给开发带来许多不便。request 范围虽说能存放大量的数据，但有效范围有限。摆在开发者面前的很多用例都要求一种比 request 范围要长，但又比 session 范围要短的这么一种有效范围。

#### Spring Web Flow 的解决方案

针对 Java Servlet 规范中的这个缺陷，Spring Web Flow 2.0 中提供了以下两种范围：

- flow 范围。此范围内的对象在 flow 开始时创建，flow 结束时销毁，在 flow 定义文件中可通过“flowScope”变量名来访问。
- conversation 范围。此范围内的对象与 flow 范围对象基本相似，唯一不同在于 conversation 范围内的对象所在的 flow 如果调用了其他 subflow，那么在 subflow 中也可访问该对象。

由于 flow 是由开发人员自己定义的，可根据业务的需求自由改变，flow 范围和 conversation 范围的使用也就突破了 Java Servlet 规范中 session 范围和 request 范围的局限，真正做到了自由定制。

## 并非所有情形都适用 Spring Web Flow

可以看出，Spring Web Flow 所着力解决的问题即是客户端与服务器的对话（conversation）问题，这个范围比 request 要长，而比 session 要短。为实现 conversation 范围（即 flow 范围），需要付出效率上的代价，因此，并非所有 Web 应用都适合使用 Spring Web Flow。Seth Ladd 等人所著 *Expert Spring MVC and Web Flow* 一书，对何时使用 Spring Web Flow，列出了如下表格。

**表 1 何时使用 Spring Web Flow**

解决方案	何时使用
Spring MVC Controller	某个单独的、只需较少业务逻辑就可创建的页面，同时该页面不是 flow 的一部分
Spring MVC SimpleFormController	某个只涉及表单提交的页面，如一个搜索框
Spring MVC AbstractWizardFormController	由一系列导航页面组成的业务过程
Spring Web Flow	任何比较复杂的、有状态的、需要在多个页面之间跳转的业务过程

## Spring Web Flow 的其他特点

Web Flow 作为一个单独的概念被提出来，也可算是 Spring Web Flow 的一大亮点。目前大多数 Web MVC 框架都把重点放在各种 controller 和形形色色的 view 技术上面，对 Web 应用流程本身的关注是不够的，Web Flow 的提出就提供了一层抽象，设计者就可以从 Web Flow 抽象层面来进行设计、开发。当然，Web Flow 不能理解为只是 Web 页面间的跳转流程，定义 Spring Web Flow 的语义并非只限于页面之间的跳转，而可以是 Web 应用中的各种行为。由此，用例的模型建构好以后，就可直接从该模型转换到相应的 Web Flow，开发人员的设计变得更加直观、有效。

另外，在 Spring Web Flow 中重用 Web Flow 是比较容易的。在定义 flow、state 时可通过继承某个已有的 flow 或 state，来避免重复定义。同时，一个 flow 可以调用其它 flow，就跟一般程序语言中在某个函数内部调用其它函数一样方便。

## 配置 Spring Web MVC

Spring Web Flow 2.0 就是 Spring Web MVC 的一个扩展，如果粗略一些来讲，所谓 flow 就相当于 Spring Web MVC 中一种特殊的 controller，这种 controller 可通过 XML 文件加以配置，因此在使用 Spring Web Flow 2.0 前须先对 Spring Web MVC 进行配置，步骤如下：

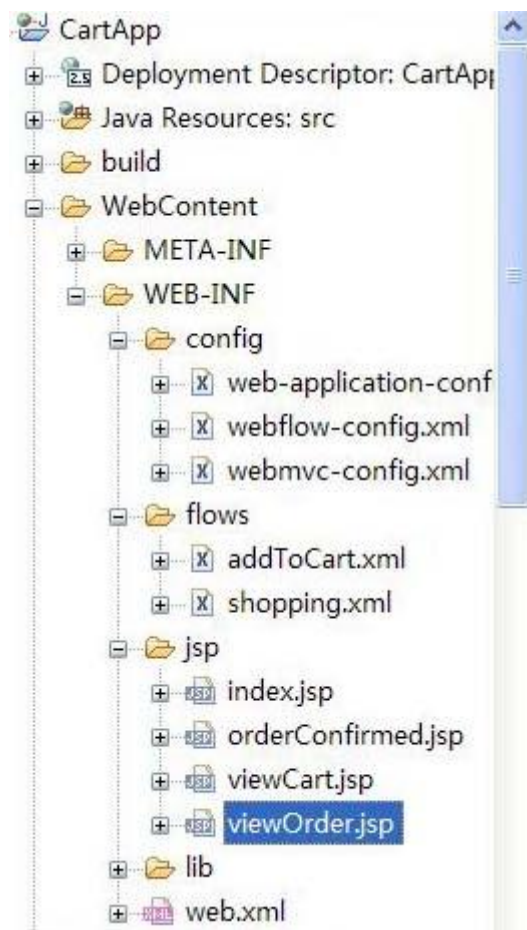
- 创建 Web 应用的目录结构
- 在 /WEB-INF/lib 下导入相关类库

- 在 Web 应用部署描述符文件 web.xml 中声明 DispatcherServlet 并指定配置文件
- 添加 DispatcherServlet 映射
- 创建 web-application-config.xml 文件
- 创建 webmvc-config.xml 文件
- 创建 index.jsp

## 创建 Web 应用的目录结构

本示例应用将采用 eclipse Dynamic Web Project 向导默认生成的目录结构, 在 WEB-INF 目录下添加 config 和 flows 子目录, 其中 config 子目录用来存放各种配置文件, flows 子目录下存放 Spring Web Flow 的定义文件。最后目录如图 2 所示:

图 2 目录结构



在 /WEB-INF/lib 下导入相关类库

只需将以下几个 jar 包导入 /WEB-INF/lib 目录下就可以了:

- commons-logging.jar
- jstl.jar

- standard.jar
- spring-webmvc.jar
- spring.jar

### 声明 DispatcherServlet 并指定配置文件

为使用 Spring Web MVC ，须在 web.xml 中声明 DispatcherServlet ，见清单 3:

#### 清单 3 声明 DispatcherServlet 和指定配置文件

```
<servlet>
  <servlet-name>CartServlet</servlet-name>
  <servlet-class>
    org.springframework.web.servlet.DispatcherServlet
  </servlet-class>
  <init-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>
      /WEB-INF/config/web-application-config.xml
    </param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>
```

### 添加 DispatcherServlet 映射

要让 DispatcherServlet 处理所有以 /spring/ 开头的请求，见清单 4:

#### 清单 4 web.xml 中的 DispatcherServlet 映射

```
<servlet-mapping>
  <servlet-name>CartServlet</servlet-name>
  <url-pattern>/spring/*</url-pattern>
</servlet-mapping>
```

### 创建 web-application-config.xml

开发基于 Spring Web Flow 的应用往往会有大量的配置，这些配置全放在一个文件中是不合适的。本示例参考 Spring Web Flow 2.0 自带示例，将不同功能的配置文件分开。其中 web-application-config.xml 用于配置与 Web 应用全局相关的内容，Spring Web MVC 的相关配置放在 webmvc-config.xml 中，教程后面要添加的 Spring Web Flow 的配置则放在 webflow-config.xml 中。在 web-application-config.xml 中用 import 元素导入其他的配置文件。web-application-config.xml 的内容见清单 5:

#### 清单 5 web-application-config.xml

```

<?xml version="1.0" encoding="utf-8"?>
<beans
  xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:context="http://www.springframework.org/schema/context"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-2.5.xsd">
  <!-- 搜索 samples.webflow 包里的 @Component 注解，并将其部署到容器中 -->
  <context:component-scan base-package="samples.webflow" />
  <!-- 启用基于注解的配置 -->
  <context:annotation-config />
  <import resource="webmvc-config.xml"/>
</beans>

```

加入注解功能是出于最后运行 Web Flow 示例的需要，在这里只要知道注解功能已被启用就可以了。

### 创建 webmvc-config.xml

webmvc-config.xml 主要用于配置 Spring Web MVC 。所要做的就是添加一个 view resolver （视图解析器），用于将视图名解析成真实的视图资源。另外，再配置好 URL 请求的 handler （处理器），用于将 URL 请求定向到某个控制器，在本例中，用到的是 UrlFilenameViewController。

#### 清单 6 webmvc-config.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<beans
  xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd">
  <bean id="viewResolver"
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="viewClass"
      value="org.springframework.web.servlet.view.JstlView"/>
    <property name="prefix" value="/WEB-INF/jsp/" />
    <property name="suffix" value=".jsp" />
  </bean>
  <bean id="viewMappings"
class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
    <property name="defaultHandler">
<!-- UrlFilenameViewController 会将 "/index" 这样的请求映射成名为 "index" 的视图 -->

```



```

        <bean
class="org.springframework.web.servlet.mvc.UrlFilenameViewController" />
        </property>
    </bean>
</beans>

```

## 创建 index.jsp

现在的 index.jsp 只是显示一行文字。

### 清单 7 index.jsp

```

<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
        <title>Cart Application</title>
    </head>
    <body>
        <h1>Hello!</h1>
    </body>
</html>

```

## 运行应用程序

将应用程序发布到 Tomcat 容器，再通过 `http://localhost:8080/CartApp/spring/index.jsp` 访问 index.jsp 页面（应用程序所在文件夹名是 CartApp），测试 Spring Web MVC 配置是否正确。如果一切正常，可得到如下页面：

图 3 显示结果



## 配置 Spring Web Flow 2.0 的基础

配置好 Spring Web MVC 的环境后，接下来就可以往里面加入 Spring Web Flow 2.0 的配置。不过，要搞明白 Spring Web Flow 2.0 的配置，必须先要了解相关的理论知识。

## FlowRegistry

FlowRegistry 是存放 flow 的仓库，每个定义 flow 的 XML 文档被解析后，都会被分配一个唯一的 id，并以 FlowDefinition 对象的形式存放在 FlowRegistry 中。FlowRegistry 配置方式可参看清单 8。

### 说明

以下的示例清单中的 XML 配置元素默认使用了 webflow 名字空间，这也是 Spring Web Flow 习惯上的名字空间，参看教程后面 webflow-config.xml 文件，可以更多了解 webflow 名字空间。

## 清单 8 FlowRegistry 的配置

```
<webflow:flow-registry id="flowRegistry">
<webflow:flow-location path="/WEB-INF/flows/shopping.xml"
id="shopping" />
</webflow:flow-registry>
```

每个 flow 都必须要有 id 来标识，如果在配置中省略，那么该 flow 默认的 id 将是该定义文件的文件名去掉后缀所得的字符串。

## FlowExecutor

FlowExecutor 是 Spring Web Flow 的一个核心接口，启动某个 flow，都要通过这个接口来进行。从配置角度来说，只要保证有个 FlowExecutor 就可以了，Spring Web Flow 的默认行为已经足够。默认配置参看清单 9。

## 清单 9 FlowExecutor 的配置

```
<webflow:flow-executor id="flowExecutor" />
```

## 哪个 flow 被执行了？

FlowRegistry 中注册的 flow 可能会有多个，但前面介绍过，每个 flow 都会有 id，没有配置的，也会有个默认值，FlowExecutor 就是通过 id 来找出要执行的 flow。至于这个 id，则是要由用户来指定的。在默认配置情况下，如果客户端发送了如下 URL 请求：

<http://localhost:8080/CartApp/spring/shopping>

则从 Spring Web Flow 的角度来看，这个 URL 就表示客户想要执行一个 id 为 “shopping” 的 flow，于是就会在 FlowRegistry 中查找名为 “shopping” 的 flow，由 FlowExecutor 负责执行。

## Spring Web Flow 如何与 Spring Web MVC 整合在一起？

客户端发送的请求，先会由 servlet 容器（本教程示例中即为 Tomcat）接收，servlet 容器会找到相应的应用程序（本教程中即为 CartApp），再根据 web.xml 的配置找出符合映射条件的 servlet 来处理。Spring Web MVC 中处理请求的 servlet 是 DispatcherServlet，如果请求的路径满足 DispatcherServlet 的映射条件，则 DispatcherServlet 会找出 Spring IoC 容器中所有的 HandlerMapping，根据这些 HandlerMapping 中匹配最好的 handler（一般情况下都是 controller，即控制器）来处理请求。当 Controller 处理完毕，一般都会返回一个 view（视图）的名字，DispatcherServlet 再根据这个 view 的名字找到相应的视图资源返回给客户端。

搞清楚 Spring Web MVC 处理请求的流程后，基本上就可以明白要整合 Spring Web MVC 与 Spring Web Flow 所需要的配置了。为了让客户端的请求变成执行某个 flow 的请求，要解决以下几个问题：

- 需要在某个 HandlerMapping 中配置负责处理 flow 请求的 handler（或 controller）
- 该 handler（或 controller）要负责启动指定的 flow
- flow 执行过程中以及执行完成后所涉及的视图应呈现给客户端

## FlowHandler 和 FlowController

现在，需要一种接收执行 flow 的请求，然后根据请求来启动相应 flow 的 handler（处理器），Spring Web Flow 2.0 提供了两种方案可供选择。第一种方案是自己编写实现了 FlowHandler 接口的类，让这个类来实现这个功能。第二种方案是使用一个现成的叫做 FlowController 的控制器。第一种方案灵活性比较大，在许多场合可能也是唯一的选择，但对每个 flow 都需要编写相应的 FlowHandler。本教程的示例采用第二种方案，对 FlowHandler 的介绍可参看 Spring Web Flow 2.0 自带的文档。FlowController 其实是个适配器，一般来讲，我们只要明白 FlowController 可根据客户端请求的结尾部分，找出相应的 flow 来执行。配置 FlowController 只需指定 FlowExecutor 即可，具体配置见清单 10：

### 清单 10 FlowController 的配置

```
<bean id="flowController"
class="org.springframework.webflow.mvc.servlet.FlowController">
    <property name="flowExecutor" ref="flowExecutor"/>
</bean>
```

另外还需在 HandlerMapping 中指明 /shopping.do 请求由 flowController 来处理，配置见清单 11：

#### 清单 11 在 viewMappings 中添加配置

```
<bean
    id="viewMappings"
    class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
    <property name="mappings">
        <value>
            /shopping.do=flowController
        </value>
    </property>
</bean>
```

需要指出的是，不管设成 /shopping.do 还是设成 /shopping，或者 /shopping.htm，效果都是一样的，flowController 都会去找 id 为 shopping 的 flow 来执行。

#### FlowBuilder Services

清单 8 所示 FlowRegistry 的配置，其中省略了 flow-registry 元素中一项比较重要的属性，flow-builder-services。flow-builder-services 属性的配置指明了在这个 flow-registry “仓库”里的 flow 的一些基本特性，例如，是用 Unified EL 还是 OGNL、model（模型）对象中的数据在显示之前是否需要先作转换，等等。在本示例中，我们需要在 flow-builder-services 属性中指明 Spring Web Flow 中所用到的 view，由 Spring Web MVC 的“View Resolver”来查找，由 Spring Web MVC 的“View Class”来解析，最后呈现给客户。具体配置参看清单 12：

#### 清单 12 flow-builder-services 配置

```
<webflow:flow-builder-services id="flowBuilderServices"
    view-factory-creator="mvcViewFactoryCreator"/>
<bean
    id="mvcViewFactoryCreator"
    class="org.springframework.webflow.mvc.builder.MvcViewFactoryCreator">
    <property name="viewResolvers" ref="viewResolver"/>
</bean>
```

#### Spring Web Flow 2.0 配置小结

所有这些配置的目的无非是两个：一是要让客户端的请求转变成 flow 的执行，二是要让 flow 执行过程中、或执行结束后得到的视图能返还给客户端。如果对这里的讲解还不是很清楚，可先看下一节实际的配置，再回过头来看本章内容，以加深理解。

## 在购物车示例应用中配置 Spring Web Flow

实现示例应用的购物车流程，可按以下步骤操作：

- 在 /WEB-INF/lib 目录下导入相关类库
- 在 webmvc-config.xml 中添加与 Spring Web Flow 集成的配置
- 添加 Spring Web Flow 的配置文件 webflow-config.xml
- 添加 flow 定义文件 shopping.xml
- 添加三个 jsp 页面
- 修改 index.jsp

### 在 /WEB-INF/lib 目录下导入相关类库

将以下几个 jar 包导入 /WEB-INF/lib 目录：

- org.springframework.webflow-2.0.2.RELEASE.jar
- org.springframework.js-2.0.2.RELEASE.jar
- org.springframework.binding-2.0.2.RELEASE.jar
- jboss-el.jar

### 在 webmvc-config.xml 中添加配置

Spring Web MVC 相关的配置前面已经分析过了，完整的配置见清单 13：

#### 清单 13 webmvc-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
  xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd">
  <bean
    id="viewResolver"
    class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="viewClass"
      value="org.springframework.web.servlet.view.JstlView">
    </property>
    <property name="prefix" value="/WEB-INF/jsp/">
    </property>
    <property name="suffix" value=".jsp">
    </property>
  </bean>
  <bean
    id="viewMappings"
    class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
```

```

    <!-- /shopping.do 请求由 flowController 来处理 -->
    <property name="mappings">
        <value> /shopping.do=flowController </value>
    </property>
    <property name="defaultHandler">
    <!-- UrlFilenameViewController 会将 "/index" 这样的请求映射成名为 "index" 的视图 -->
        <bean
class="org.springframework.web.servlet.mvc.UrlFilenameViewController" />
        </property>
    </bean>
    <bean
        id="flowController"
        class="org.springframework.webflow.mvc.servlet.FlowController">
        <property name="flowExecutor" ref="flowExecutor"/>
    </bean>
</beans>

```

### 添加配置文件 webflow-config.xml

在 /WEB-INF/config 目录下添加 webflow-config.xml 文件， schema 名字空间可直接复制清单 14 中的内容。

#### 清单 14 webflow-config.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<beans
    xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:webflow="http://www.springframework.org/schema/webflow-config"
    xsi:schemaLocation=" http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
        http://www.springframework.org/schema/webflow-config
http://www.springframework.org/schema/webflow-config/spring-webflow-config-2.0.xsd">
    <webflow:flow-executor id="flowExecutor"/>
    <!-- 所有 flow 定义文件位置在此配置， flow-builder-services 用于配置 flow 的特性 -->
    <webflow:flow-registry id="flowRegistry"
        flow-builder-services="flowBuilderServices">
        <webflow:flow-location path="/WEB-INF/flows/shopping.xml" id="shopping"/>
    </webflow:flow-registry>
    <!-- Web Flow 中的视图通过 MVC 框架的视图技术来呈现 -->
    <webflow:flow-builder-services id="flowBuilderServices"
        view-factory-creator="mvcViewFactoryCreator"/>
    <!-- 指明 MVC 框架的 view resolver ， 用于通过 view 名查找资源 -->
    <bean id="mvcViewFactoryCreator"
        class="org.springframework.webflow.mvc.builder.MvcViewFactoryCreator">
        <property name="viewResolvers" ref="viewResolver"/>
    </bean>

```

```
</bean>
</beans>
```

webflow-config.xml 创建完成以后，不要忘记在 web-application-config.xml 中添加 import 元素，将 webflow-config.xml 文件导入。

**清单 15 在 web-application-config.xml 中导入 webflow-config.xml。**

```
<import resource="webflow-config.xml"/>
```

**添加 flow 定义文件 shopping.xml**

在 /WEB-INF/flows 目录下创建 shopping.xml 文件。

**清单 16 shopping.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<flow xmlns="http://www.springframework.org/schema/webflow"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.springframework.org/schema/webflow
http://www.springframework.org/schema/webflow/spring-webflow-2.0.xsd">
  <view-state id="viewCart" view="viewCart">
    <transition on="submit" to="viewOrder">
    </transition>
  </view-state>
  <view-state id="viewOrder" view="viewOrder">
    <transition on="confirm" to="orderConfirmed">
    </transition>
  </view-state>
  <view-state id="orderConfirmed" view="orderConfirmed">
    <transition on="returnToIndex" to="returnToIndex">
    </transition>
  </view-state>
  <end-state id="returnToIndex"
            view="externalRedirect:servletRelative:/index.jsp">
  </end-state>
</flow>
```

在 view-state 元素中指定了 view 属性的名字，这个名字也是 Spring Web MVC 中 viewResolver 所查找的 view 的名字。从清单 16 的配置中可以知道，这三个 view-state 元素所对应的视图资源分别应该是：viewCart.jsp、viewOrder.jsp 和 orderConfirmed.jsp。清单 16 中最后的 end-state 指明了当 flow 执行结束后跳转到初始的 index.jsp 页面，在此处的 view 属性的名字需要解释一下。externalRedirect 用在 view 名字中，表示所指向的资源是在 flow 的外部，servletRelative 则表明所指向资源的路径起始部分与

flow 所在 servlet 相同。Spring Web Flow 2.0 还提供了其他几个关键词用于重定向，这里就不多介绍了。

### 添加三个 jsp 页面

在 /WEB-INF/jsp 目录下创建三个 flow 所需的视图资源。以下清单只给出 jsp 页面中 body 元素以内的代码，其余省略。

#### 清单 17 viewCart.jsp

```
<h1>View Cart</h1>
<a href="${flowExecutionUrl}&_eventId=submit">Submit</a>
```

#### 清单 18 viewOrder.jsp

```
<h1>Order</h1>
<a href="${flowExecutionUrl}&_eventId=confirm">Confirm</a>
```

#### 清单 19 orderConfirmed.jsp

```
<h1>Order Confirmed</h1>
<a href="${flowExecutionUrl}&_eventId=returnToIndex">
Return to index
</a>
```

这几个页面都使用了变量 flowExecutionUrl，表示 flow 执行到当前状态时的 URL。flowExecutionUrl 的值已经由 Spring Web Flow 2.0 框架的代码进行赋值，并放入相应的 model 中供 view 访问。flowExecutionUrl 的值包含 flow 在执行过程中会为每一状态生成的唯一的 key，因此不可用其他手段来获取。请求参数中 \_eventId 的值与清单 16 中 transition 元素的 on 属性的值是对应的，在接收到 \_eventId 参数后，相应 transition 会被执行。

### 修改 index.jsp 页面

在 index.jsp 页面中添加启动 flow 的链接，从 webmvc-config.xml 配置文件中可以看出，要启动 flow，只需提供 /shopping.do 链接即可。

#### 清单 20 index.jsp

```
<h1>Hello!</h1><br/>
<a href="/shopping.do">View Cart</a>
```

### 运行应用程序

将应用程序发布到 Tomcat 服务器，访问 index.jsp，并启动 flow，测试页面的跳转。效果如图 4 所示：

图 4 flow 运行效果





## 用 Unified EL 实现业务逻辑

到现在为止，这个购物车应用只是实现了页面之间的跳转，接下来我们要实现与业务逻辑相关的功能。由于本教程的重点在于介绍如何应用 Spring Web Flow，所实现的业务比较简单，与实际应用有较大的距离，请读者谅解。

业务的逻辑涉及到数据的获取、传递、保存，相关的业务功能函数的调用等内容，这些功能的实现都可用 Java 代码来完成，但定义 Spring Web Flow 的语法与 Java 是无关的，这就要求 Spring Web Flow 提供与 Java 代码的整合机制。要了解这种机制，关键在于搞清楚两个问题：

- 业务逻辑代码在什么时候被调用？
- 业务逻辑代码在调用后得到的数据如何保存、传递？

### 业务逻辑代码在什么时候被调用？

在 Spring Web Flow 中，业务逻辑代码的执行可由以下三种情形来触发：

- 客户端请求中包含了 `_eventId` 参数
- 执行到框架自定义的切入点
- 执行到 `<action-state>` 元素

### 客户端请求中包含了 `_eventId` 参数

这种方式一般用在 state 之间的 transition，通过指定 `_eventId` 参数的值，表明了客户的行为，从而导致相应事件的发生，在 Spring Web Flow 的定义文件中可以通过 `evaluate` 元素来指定要处理的业务逻辑。参看清单 21：

#### 清单 21 transition 示例

```
<transition on="submit">
<evaluate expression="validator.validate()" />
</transition>
```

清单 21 的代码表示，当客户端的请求中包含 “\_eventId=submit”，则 evaluate 元素中 expression 属性所指明的表达式会被执行，即 validator 对象的 validate 方法会得到调用。

## 执行到框架自定义的切入点

Spring Web Flow 定义了 5 个切入点，通过 flow 定义文件的配置，可在这 5 个切入点插入相关业务逻辑代码。

**表 2 Spring Web Flow 自定义的切入点**

切入点名称	XML 元素名称	触发时刻
flow start	on-start	flow 执行之前
state entry	on-entry	进入某个 state 之后，做其他事情之前
view render	on-render	在进入 view 的 render 流程之后，在 view 真正 render 出来之前
state exit	on-exit	在退出 state 之前
flow end	on-end	flow 执行结束之后

清单 22 给出了在 view render 切入点插入业务逻辑代码的例子：

**清单 22 on-render 元素**

```
<view-state id="viewCart" view="viewCart" >
  <on-render>
    <evaluate expression="productService.getProducts()"
      result="viewScope.products"/>
  </on-render>
</view-state>
```

## 执行到 <action-state> 元素

Spring Web Flow 中的这个 <action-state> 是专为执行业务逻辑而设的 state。如果某个应用的业务逻辑代码即不适合放在 transition 中由客户端来触发，也不适合放在 Spring Web Flow 自定义的切入点，那么就可以考虑添加 <action-state> 元素专用于该业务逻辑的执行。示例代码参看清单 23：

**清单 23 action-state 示例**

```
<action-state id="addToCart">
  <evaluate
expression="cart.addItem(productService.getProduct(productId))"/>
  <transition to="productAdded"/>
</action-state>
```

**业务逻辑代码在调用后得到的数据如何保存、传递？**

Spring Web Flow 的定义中可直接使用表达式语言（ Expression Language ），前面的代码都是用的 Unified EL ，对于习惯用 OGNL 的开发人员，可通过 flow-builder-services 的配置改成使用 OGNL 。不管是哪一种表达式语言， Spring Web Flow 都提供了一些固定名称的变量，用于数据的保存、传递。在 [Spring Web Flow 的解决方案](#) 一节中，已经提到 Spring Web Flow 所着力解决的问题即是数据存取范围的问题，为此， Spring Web Flow 提供了两种比较重要的范围，一是 flow 范围，另一个是 conversation 范围。通过 flowScope 和 conversationScope 这两个变量， Spring Web Flow 提供了在这两种范围里存取数据的方法。清单 24 演示了如何将业务逻辑代码执行的结果存放到 flow 范围中。

#### 清单 24 flowScope 示例

```
<evaluate expression="productService.getProducts()"
    result="flowScope.products" />
```

#### 注意

Spring Web Flow 2.0 在默认配置下， flowScope 和 conversationScope 的实现依赖于 Java 序列化和反序列化技术，因此存放于 flowScope 或 conversationScope 中的对象需要实现 java.io.Serializable 接口。

Spring Web Flow 还提供了大量其他的变量，以方便数据的存取。如 viewScope 范围即是从进入 view-state 至退出 view-state 结束， requestScope 即和一般的 request 范围没什么区别，等等。另外还有一些用于获取 flow 以外数据的变量，如 requestParameters 、 messageContext 等等。具体变量的列表可参看 Spring Web Flow 自带的文档。

#### 为示例应用添加商品

接下来，我们要在示例应用的 viewCart.jsp 页面中添加商品，可按以下步骤操作：

- 添加 Product 类
- 添加 ProductService 类
- 修改 shopping.xml 文件
- 修改 viewCart.jsp 页面

#### 添加 Product 类

Product 类是个普通的 JavaBean ，用于定义商品（ Product ）的一般属性，同时也提供了构造方法。由于会把 Product 存放于 conversationScope 中， Product 实现了 Serializable 接口。具体见清单 25：

### 清单 25 Product 类

```
package samples.webflow;

import java.io.Serializable;

public class Product implements Serializable {

    private static final long serialVersionUID = 1951520003958305899L;
    private int id;
    private String description;
    private int price;

    public Product(int id, String description, int price) {
        this.id = id;
        this.description = description;
        this.price = price;
    }

    /*省略 getter 和 setter*/

}
```

### 添加 ProductService 类

ProductService 主要提供商品列表，并能根据商品的 id 查找出该商品，由于示例较简单，这里只添加了三条纪录。见清单 26：

### 清单 26 ProductService 类

```
package samples.webflow;

/*省略 import 语句*/

@Service("productService")
public class ProductService {
    /*products 用于存放多个商品 */
    private Map<Integer, Product> products =
        new HashMap<Integer, Product>();

    public ProductService() {
        products.put(1, new Product(1, "Bulldog", 1000));
        products.put(2, new Product(2, "Chihuahua", 1500));
        products.put(3, new Product(3, "Labrador", 2000));
    }

    public List<Product> getProducts() {
```

```

        return new ArrayList<Product>(products.values());
    }

    public Product getProduct(int productId) {
        return products.get(productId);
    }
}

```

Service 注解表示 Spring IoC 容器会初始化一个名为 productService 的 Bean，这个 Bean 可在 Spring Web Flow 的定义中直接访问。

### 修改 shopping.xml 文件

要在 viewCart 页面中显示商品，只需在 view-state 元素的 on-render 切入点调用 productService 的 getProducts 方法，并将所得结果保存到 viewScope 中即可。见清单 27：

#### 清单 27 shopping.xml 需修改的部分

```

<view-state id="viewCart" view="viewCart" >
    <on-render>
        <evaluate expression="productService.getProducts()"
                    result="viewScope.products"/>
    </on-render>
    <transition on="submit" to="viewOrder"> </transition>
</view-state>

```

### 修改 viewCart.jsp 页面

清单 27 表明 productService 的 getProducts 方法所得的结果会存放在 viewScope 中名为 products 的变量中，jsp 页面的代码可直接访问该变量。见清单 28：

#### 清单 28 修改后的 viewCart.jsp 页面

```

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>View Cart</title>
</head>
<body>
<h1>View Cart</h1>
<h2>Items in Your Cart</h2>
<a href="${flowExecutionUrl}&_eventId=submit">Submit</a>

```

```

<h2>Products for Your Choice</h2>
<table>
<c:forEach var="product" items="${products}">
<tr>
<td>${product.description}</td>
<td>${product.price}</td>
</tr>
</c:forEach>
</table>
</body>
</html>

```

## 运行应用程序

图 5 viewCart.jsp 页面效果



## 用 subflow 实现添加商品到购物车功能

商品已经有列表了，接下来就要增加把商品放入购物车的功能，在本示例中用 subflow 来实现这一功能，操作步骤如下：

- 实现 Cart 和 CartItem 两个业务类
- 在 shopping.xml 中添加配置
- 在 /WEB-INF/flows 目录下添加 addToCart.xml
- 在 webflow-config.xml 中添加 addToCart.xml 的位置
- 修改 viewCart.jsp 页面

## 实现 Cart 和 CartItem 两个业务类

CartItem 表示存放于购物车中的条目，主要记录相应商品及商品数量，同时不要忘记实现 java.io.Serializable 接口，见清单 29：

#### 清单 29 CartItem 类

```
package samples.webflow;

import java.io.Serializable;

public class CartItem implements Serializable {
    private static final long serialVersionUID = 8388627124326126637L;
    private Product product;
    private int quantity;

    public CartItem(Product product, int quantity) {
        this.product = product;
        this.quantity = quantity;
    }

    public int getTotalPrice() {
        return this.quantity * this.product.getPrice();
    }

    public void increaseQuantity() {
        this.quantity++;
    }

    /*省略 getter 和 setter*/
}
```

除去相应的属性外，CartItem 可根据商品的数量算出该商品的总价格（getTotalPrice），也可通过 increaseQuantity 增加商品数量。

Cart 是购物车的实现类，其同样要实现 java.io.Serializable 接口，但它没有像 ProductService 一样成为由 Spring IoC 容器管理的 Bean，每个客户的购物车是不同的，因此不能使用 Spring IoC 容器默认的 Singleton 模式。见清单 30：

#### 清单 30 Cart 类

```
package samples.webflow;

/* 省略 import 语句 */

public class Cart implements Serializable {
```

```

private static final long serialVersionUID = 7901330827203016310L;
private Map<Integer, CartItem> map =
    new HashMap<Integer, CartItem>();

public List<CartItem> getItems() {
    return new ArrayList<CartItem>(map.values());
}

public void addItem(Product product) {
    int id = product.getId();
    CartItem item = map.get(id);
    if (item != null)
        item.increaseQuantity();
    else
        map.put(id, new CartItem(product, 1));
}

public int getTotalPrice() {
    int total = 0;
    for (CartItem item : map.values())
        total += item.getProduct().getPrice() * item.getQuantity();
    return total;
}
}

```

Cart 主要实现三个业务函数， `getItems` 用于获取当前购物车里的物品， `addItem` 用于向购物车添加商品， `getTotalPrice` 用于获取购物车里所有商品的总价格。

## 在 shopping.xml 中添加配置

在 shopping flow 开始时必须分配一个 Cart 对象，由于要调用 subflow，这个 Cart 对象应存放于 conversationScope 中。同时要添加一个 subflow-state 用于执行添加商品到购物车的任务。

### 清单 31 shopping.xml 中添加的配置

```

<var name="mycart" class="samples.webflow.Cart"/>
<on-start>
    <set name="conversationScope.cart" value="mycart"></set>
</on-start>
<view-state id="viewCart" view="viewCart" >
    <on-render>
        <evaluate expression="productService.getProducts()"

```



```

result="viewScope.products"/>
</on-render>
<transition on="submit" to="viewOrder"/>
<transition on="addToCart" to="addProductToCart"/>
</view-state>
<subflow-state id="addProductToCart" subflow="addToCart">
  <transition on="productAdded" to="viewCart" />
</subflow-state>

```

在 `/WEB-INF/flows` 目录下添加 `addToCart.xml`

清单 31 中 `subflow-state` 元素的 `subflow` 属性即指明了这个被调用的 flow 的 id 为 “`addToCart`”，现在就要添加 `addToCart` flow 的定义。

### 清单 32 `addToCart.xml`

```

<?xml version="1.0" encoding="UTF-8"?>
<flow xmlns="http://www.springframework.org/schema/webflow"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/webflow
http://www.springframework.org/schema/webflow/spring-webflow-2.0.xsd">
  <on-start>
    <set name="requestScope.productId"
      value="requestParameters.productId"/>
  </on-start>
  <action-state id="addToCart">
    <evaluate
expression="cart.addItem(productService.getProduct(productId))"/>
    <transition to="productAdded"/>
  </action-state>
  <end-state id="productAdded"/>
</flow>

```

`addToCart` flow 主要由一个 `action-state` 构成，完成添加商品到购物车的功能，`addToCart` flow 的实现需要有输入参数，即 `productId`。在本示例中是通过请求参数来传递，通过 `requestParameters` 来获取该数值。这里还要注意清单 32 中的 `end-state` 的 id 为 “`productAdded`”，与清单 31 中 `subflow-state` 中的 `transition` 元素的 `on` 属性的名称是对应的。

在 `webflow-config.xml` 中添加 `addToCart.xml` 的位置

新增加的 flow 不要忘记在 `flow-registry` 中注册。

### 清单 33 `flow-registry` 中注册 `addToCart`

```

<webflow:flow-registry id="flowRegistry"
  flow-builder-services="flowBuilderServices">

```

```

    <webflow:flow-location path="/WEB-INF/flows/shopping.xml"
                        id="shopping"/>
    <webflow:flow-location path="/WEB-INF/flows/addToCart.xml"
                        id="addToCart"/>
</webflow:flow-registry>

```

## 修改 viewCart.jsp 页面

最后就可以来看在视图中如何显示相关的信息，并触发相应的 webflow 事件，见清单 34：

### 清单 34 完整的 viewCart.jsp 的代码

```

<h1>View Cart</h1>
<h2>Items in Your Cart</h2>
<c:choose>
    <c:when test="${empty cart.items}">
    <p>Your cart is empty.</p>
    </c:when>
    <c:otherwise>
    <table border="1" cellspacing="0">
    <tr>
    <th>Item</th>
    <th>Quantity</th>
    <th>Unit Price</th>
    <th>Total</th>
    </tr>

    <c:forEach var="item" items="${cart.items}">
    <tr>
    <td>${item.product.description}</td>
    <td>${item.quantity}</td>
    <td>${item.product.price}</td>
    <td>${item.totalPrice}</td>
    </tr>
    </c:forEach>

    <tr>
    <td>TOTAL:</td>
    <td></td>
    <td></td>
    <td>${cart.totalPrice}</td>
    </tr>
    </table>
    </c:otherwise>
    </c:choose>

```

```

<a href="${flowExecutionUrl}&_eventId=submit">Submit</a>
<h2>Products for Your Choice</h2>

<table>
<c:forEach var="product" items="${products}">
<tr>
<td>${product.description}</td>
<td>${product.price}</td>

<td>
<a
href="${flowExecutionUrl}&_eventId=addToCart&productId=${product.id}">[add
to cart]</a>
</td>

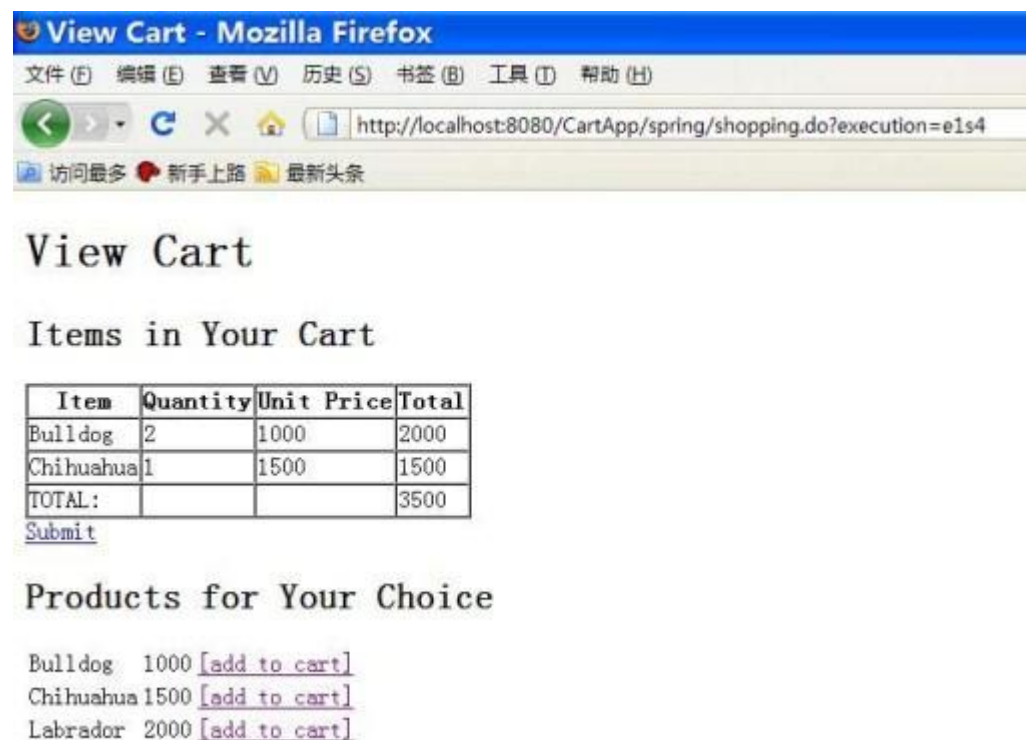
</tr>
</c:forEach>

</table>

```

## 运行效果

图 6 添加购物车后的效果



**View Cart**

Items in Your Cart

Item	Quantity	Unit Price	Total
Bulldog	2	1000	2000
Chihuahua	1	1500	1500
TOTAL:			3500

[Submit](#)

**Products for Your Choice**

Bulldog 1000 [\[add to cart\]](#)  
Chihuahua 1500 [\[add to cart\]](#)  
Labrador 2000 [\[add to cart\]](#)

## global transition 简介

顾名思义，global transition 是一种全局的 transition，可在 flow 执行的各个 state 中被触发。

### 清单 35 global-transitions

```
<global-transitions>
  <transition on="cancelShopping" to="returnToIndex"/>
</global-transitions>
```

客户端请求中如果包含 \_eventId=cancelShopping，则会重新回到 index.jsp 页面。

## 结束语

本教程通过一个简单的实例演示了 Spring Web Flow 的基本功能，通过本教程的学习，读者应当能对如何应用 Spring Web Flow 有了基本的了解。由于 2.0 版本的参考资料较为缺乏，一些比较深入的问题还有待进一步探讨。