Spring MVC

参考文档：

http://jinnianshilongnian.iteye.com/blog/pdf

1. webMVC 介绍

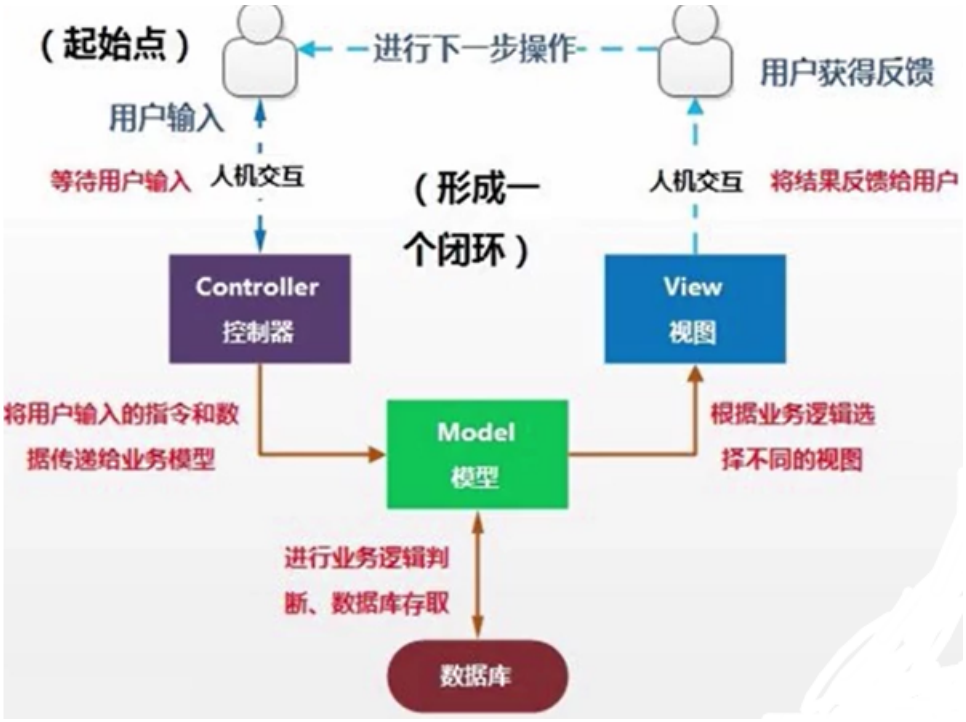
1.1MVC 是什么

MVC 是一种使用 MVC(Model View Controller 模式-视图-控制器)设计创建 web 应用程序的模式

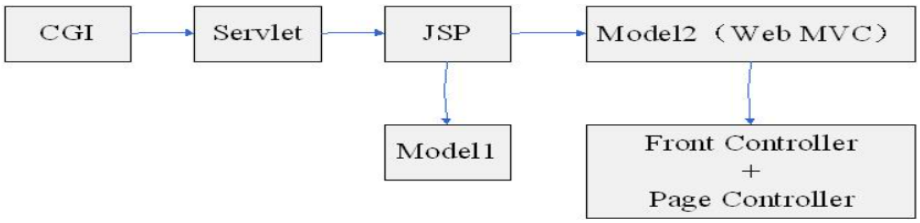
Model(模型)表示应用程序核心，用于处理数据逻辑的对象，通常处理数据库存取

View(视图)显示数据，依据模型数据创建

Cotroller(控制器)处理用户交互，从视图读取数据，控制用户输入，并向模型发送数据



1.2web 端开发发展历程



1. CGI：（Common Gateway Interface）公共网关接口

一种在 web 服务端使用的脚本技术，使用 C 或 Perl 语言编写，用于接收 web 用户请求并处理，最后动态产生响应给用户，但每次请求将产生一个进程，重量级。

2. Servlet

一种 JavaEE web 组件技术，是一种在服务器端执行的 web 组件，用于接收 web 用户请求并处理，最后动态产生响应给用户。但每次请求

只产生一个线程（而且有线程池），轻量级。而且能利用许多 JavaEE 技术（如 JDBC 等）。本质就是在 java 代码里面输出 html 流。但表现逻辑、控制逻辑、业务逻辑调用混杂。如下图：

这种做法是绝对不可取的，控制逻辑、表现代码、业务逻辑对象调用混杂在一起，最大的问题是直接在 Java 代码里面输出 Html，这样前端开发人员无法进行页面风格等的设计与修改，即使修改也是很麻烦，因此实际项目这种做法不可取。

```
public class LoginServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
            throws ServletException, IOException {
        doPost(req, resp);//为了简单，直接委托给 doPost进行处理
    }
    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
        String submitFlag = req.getParameter("submitFlag");
        if("toLogin".equals(submitFlag)) {
            toLogin(req, resp);return;
        } else if("login".equals(submitFlag)) {
            login(req, resp);return;
        }
        toLogin(req, resp); //默认到登录页面
    }
    private void toLogin(HttpServletRequest req, HttpServletResponse resp)
        throws IOException {
        resp.setContentType("text/html");
        String loginPath = req.getContextPath() + "/servletLogin";
        PrintWriter write = resp.getWriter();
        write.write("<form action='" + loginPath + "' method='post'>");
        write.write("<input type='text' name='submitFlag' value='login'/>");
        write.write("username:<input type='text' name='username'/>");
        write.write("password:<input type='password' name='password'/>");
        write.write("<input type='submit' value='login'/>");
        write.write("</form>");
    }
    private void login(HttpServletRequest req, HttpServletResponse resp)
        throws IOException {
        //1收集参数
        String username = req.getParameter("username");
        String password = req.getParameter("password");
        //2验证并封装参数（重复的步骤）
        UserBean user = new UserBean();
        user.setUsername(username);
        user.setPassword(password);
        //3调用javabean对象（业务方法）
        if(user.login()) {
            //4根据返回值选择下一个页面
            resp.getWriter().write("login success");
        } else {
            resp.getWriter().write("login fail");
        }
    }
}
```

1、控制逻辑：根据请求参数选择要执行的功能方法

2、表现代码：页面展示直接放在我们的servlet里边

3、调用业务对象(javabean对象)进行登录：即模型，不仅包含数据还包含行为

3. JSP：（Java Server Page）

一种在服务器端执行的 web 组件，是一种运行在标准的 HTML 页面中嵌入脚本语言（现在只支持 Java）的模板页面技术。本质就是在

html 代码中嵌入 java 代码。JSP 最终还是会被编译为 Servlet，只不过比纯 Servlet 开发页面更简单、方便。但表现逻辑、控制逻辑、业务逻辑调用还是混杂。

这种做法也是绝对不可取的，控制逻辑、表现代码、业务逻辑对象调用混杂在一起，但比直接在 servlet 里输出 html 要好一点，前端开发人员可以进行简单的页面风格等的设计与修改(但如果嵌入的 java 脚本太多也是很难修改的)，因此实际项目这种做法不可取。

JSP 本质还是 Servlet，最终在运行时会生成一个 Servlet(如 tomcat，将在 tomcat\work\Catalina\web 应用名\org\apache\jsp 下生成)，但这种使得写 html 简单点，但仍是控制逻辑、表现代码、业务逻辑对象调用混杂在一起。

```
<%@page import="cn.javass.chapter1.javabean.UserBean"%>
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>登录</title>
</head>
<body>
<%
    String submitFlag = request.getParameter("submitFlag");
    if("login".equals(submitFlag)) {//登录
        //1收集参数
        String username = request.getParameter("username");
        String password = request.getParameter("password");
        //2验证并封装参数
        UserBean user = new UserBean();
        user.setUsername(username);
        user.setPassword(password);
        //33调用javabean对象（业务方法）
        if(user.login()) {
              //4根据返回值选择下一个页面
            out.write("login success");
        } else {
            out.write("login fail");
        }
    } else {
%>
                <form action="" method="post">
                    <input type="hidden" name="submitFlag" value="login"/>
                    username:<input type="text" name="username"/><br/>
                    password:<input type="password" name="password"/><br/>
                    <input type="submit" value="login"/>
                </form>
<%      } %>
</body>
</html>
```

1、控制逻辑：根据请求参数选择要执行的功能方法

3、调用业务对象(javabean对象)进行登录：即模型，不仅包含数据还包含行为

2、表现代码：页面展示直接放在我们的servlet里边

4. Model1

可以认为是 JSP 的增强版，可以认为是 jsp+javabean

特点：使用<jsp:useBean>标准动作，自动将请求参数封装为 JavaBean 组件；还必须使用 java 脚本执行控制逻辑。

此处我们可以看出，使用<jsp:useBean>标准动作可以简化 javabean 的获取/创建，及将请求参数封装到 javabean

```
<%@page import="cn.javass.chapter1.javabean.UserBean"%>
<%@ page language="java" contentType="text/html; charset=UTF-8"
 //1收集参数
       String username = request.getParameter("username");
       String password = request.getParameter("password");
       //2验证并封装参数
       UserBean user = new UserBean();
       user.setUsername(username);
       user.setPassword(password);
" "http://www.w3.org/TR/html4/loose.dtd">
TF-8">

<%-- 创建javabean --%>
<jsp:useBean id="user" class="cn.javass.chapter1.javabean.UserBean"/>
<%--1、收集参数并封装参数（比直接使用jsp，在这块是简单的） --%>
<jsp:setProperty name="user" property="*"/>

<%
    String submitFlag = request.getParameter("submitFlag");
    if("login".equals(submitFlag)) {//登录
        //3调用javabean对象（业务方法）
        if(user.login()) {
              //4根据返回值选择下一个页面
            out.write("login success");
        } else {
            out.write("login fail");
        }
    } else {
%>

            <form action="" method="post">
                <input type="hidden" name="submitFlag" value="login"/>
                username:<input type="text" name="username"/><br/>
                password:<input type="password" name="password"/><br/>
                <input type="submit" value="login"/>
            </form>

<%      } %>

</body>
</html>
```

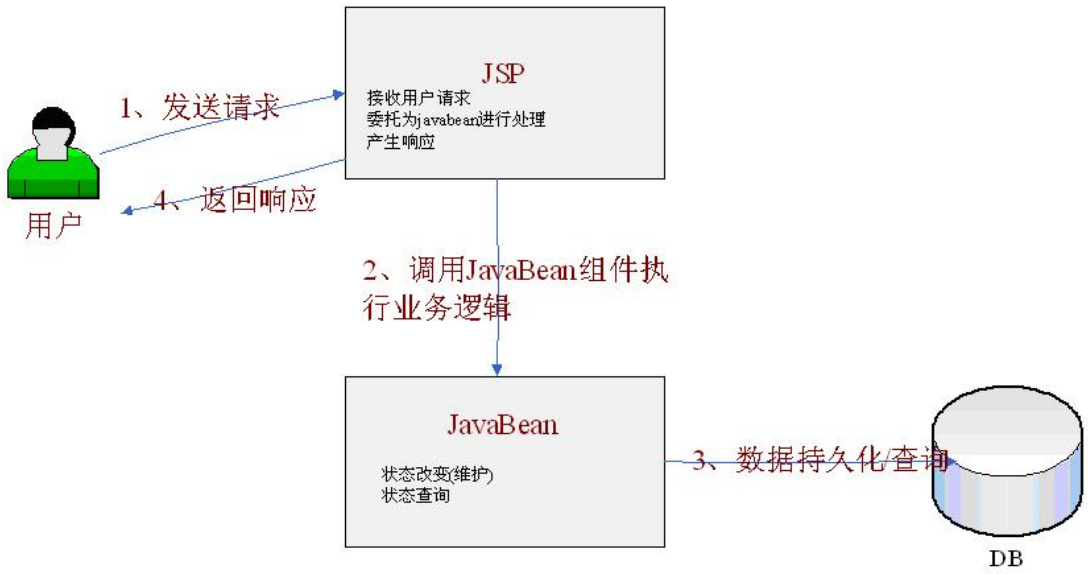
收集参数和组织参数简单了许多

1、控制逻辑：根据请求参数选择要执行的功能方法

3、调用业务对象(javabean对象)进行登录：即模型，不仅包含数据还包含行为

2、表现代码：页面展示直接放在我们的servlet里边

Mode1 架构：



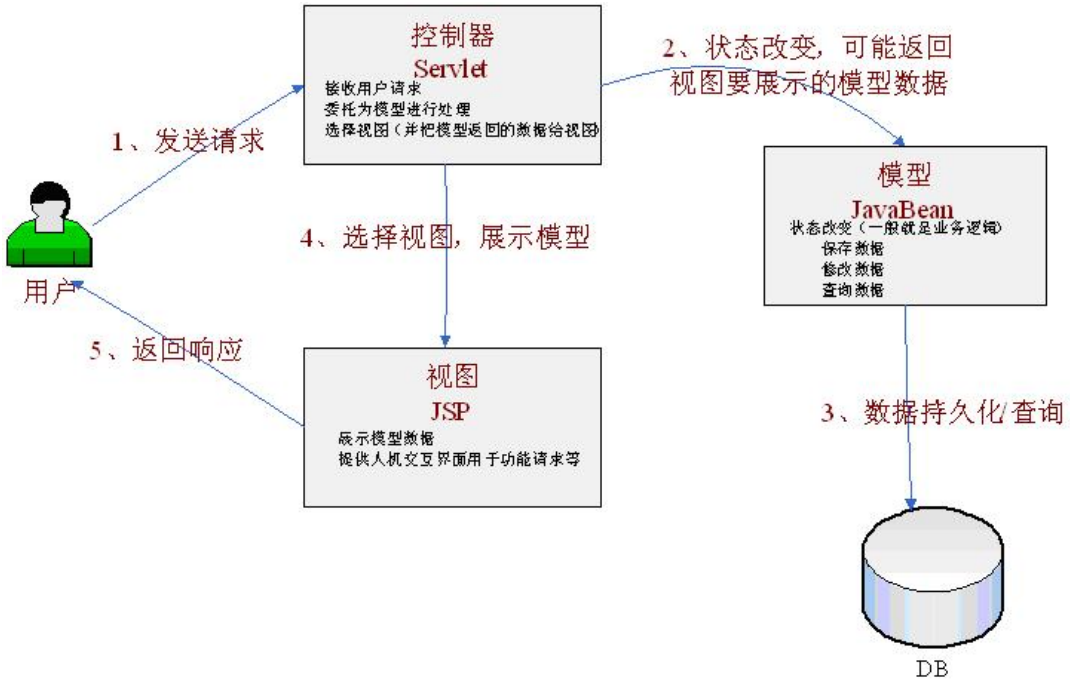
Model1 架构中，JSP 负责控制逻辑、表现逻辑、业务对象（javabean）的调用，只是比纯 JSP 简化了获取请求参数和封装请求参数。同样是不好的，在项目中应该严禁使用（或最多再 demo 里使用）。

5. Model2

在 JavaEE 世界里，它可以认为就是 Web MVC 模型

Model2 架构其实可以认为就是我们所说的 Web MVC 模型，只是控制器采用 Servlet、模型采用 JavaBean、视图采用 JSP

Model2 架构：



示例：

模型

```
public class UserBean implements java.io.Serializable {

	private String username;
	private String password;
	public String getUsername() {
		return username;
	}
	public void setUsername(String username) {
		this.username = username;
	}
	public String getPassword() {
		return password;
	}
	public void setPassword(String password) {
		this.password = password;
	}

	/**
	 * 因为我们只关注表现层，此处只是模拟，实际项目需要改掉！
	 * @param username 用户名
	 * @param password 密码
	 * @return
	 */
	public boolean login() {
		if("zhang".equals(this.username) && "123".equals(this.password)) {
			return true;
		}
		return false;
	}

}
```

模型: (业务对象 JavaBean对象)
包含设置/获取数据的方法
包含业务方法

视图

```jsp
<%@page import="cn.javass.chapter1.javabean.UserBean"%>
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>登录</title>
</head>
<body>

<form action="${pageContext.request.contextPath}/model2Login" method="post">
    <input type="hidden" name="submitFlag" value="login"/>
    username:<input type="text" name="username" value="${user.username}"/><br/>
    password:<input type="password" name="password"/><br/>
    <input type="submit" value="login"/>
</form>

</body>
</html>
```

模型展示,可能包含一些展示逻辑

展示逻辑如在网站首页
如果用户已登陆,显示”欢迎访问,sishuok”
如果用户未登陆,显示”欢迎访问,游客”

控制器

```java
public class Model2Servlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
            throws ServletException, IOException {
        doPost(req, resp);//为了简单，直接委托给 doPost进行处理
    }
    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
        String submitFlag = req.getParameter("submitFlag");
        if("toLogin".equals(submitFlag)) {
            toLogin(req, resp);
            return;
        } else if("login".equals(submitFlag)) {
            login(req, resp);
            return;
        }
        toLogin(req, resp); //默认到登录页面
    }
    private void toLogin(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
        //此处和JSP视图技术紧密耦合，更换其他视图技术几乎不可能
        req.getRequestDispatcher("/mvc/login.jsp").forward(req, resp);
    }
    private void login(HttpServletRequest req, HttpServletResponse resp)
        throws IOException, ServletException {
        //1收集参数
        String username = req.getParameter("username");
        String password = req.getParameter("password");
        //2验证并封装参数（重复的步骤）
        UserBean user = new UserBean();
        user.setUsername(username);
        user.setPassword(password);
        //3调用javabean对象（业务方法）
        if(user.login()) {
                //4根据返回值选择下一个页面
                resp.sendRedirect(req.getContextPath() + "mvc/success.jsp");
        } else {
                //登陆失败再返回登陆页面  并显示上次输入的用户名

                //将视图要显示的模型数据放在请求里传递给视图   视图再来展示
                //此处也可以看出和Servlet API紧密耦合，更换视图技术也需要重新设计请求数据
                req.setAttribute("user", user);
                toLogin(req, resp);
                return;
        }
    }
}
```

1、控制逻辑：根据请求参数选择要执行的功能方法

2、调用业务对象(javabean对象)进行登录：即模型，不仅包含数据还包含行为

3、选择下一个视图进行模型展示，模型数据直接放在request里

从 Model2 架构可以看出，视图和模型分离了，控制逻辑和展示逻辑分离了。

但我们也看到严重的缺点：

1、控制器：

1、1 控制逻辑可能比较复杂，其实我们可以按照规约，如请求参数

submitFlag=toAdd，我们其实可以直接调用 toAdd 方法，来简化控制逻辑；而且每个模块基本需要一个控制器，造成控制逻辑可能很复杂；

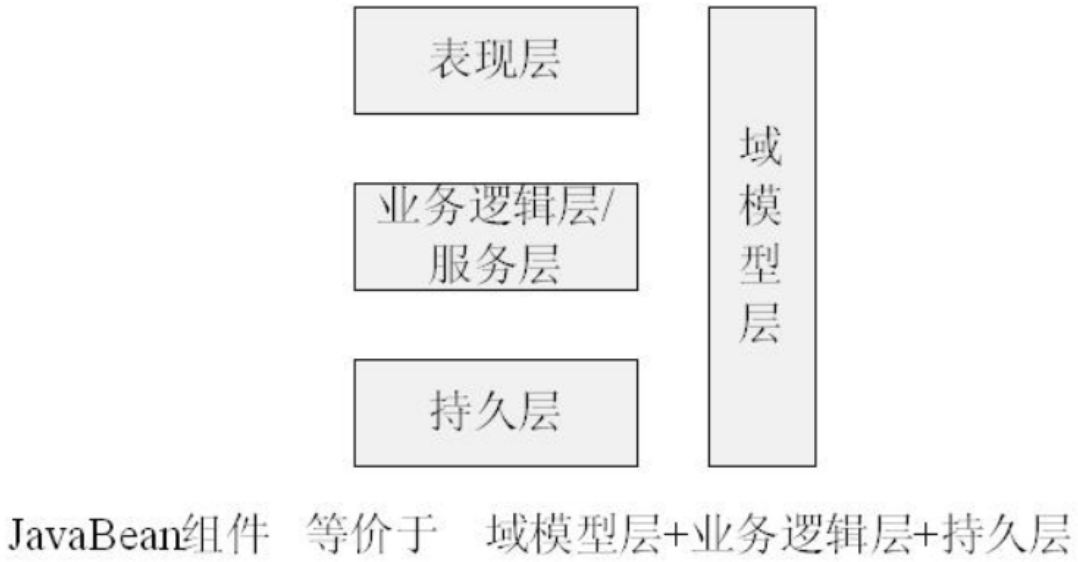
1、2、请求参数到模型的封装比较麻烦，如果能交给框架来做这件事情，我们可以从中得到解放；

1、3、选择下一个视图，严重依赖 Servlet API，这样很难或基本不可能更换视图；

1、4、给视图传输要展示的模型数据，使用 Servlet API，更换视图技术也要一起更换，很麻烦。

2. 模型：

2、1、此处模型使用 JavaBean，可能造成 JavaBean 组件类很庞大，一般现在项目都是采用三层架构，而不采用 JavaBean

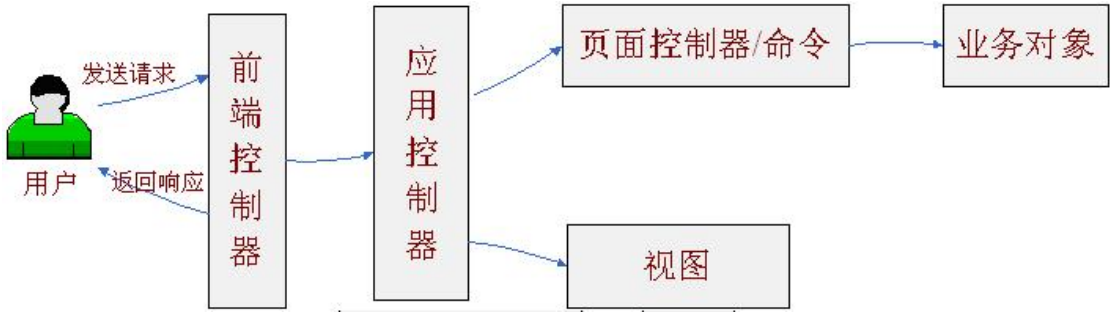


3、视图

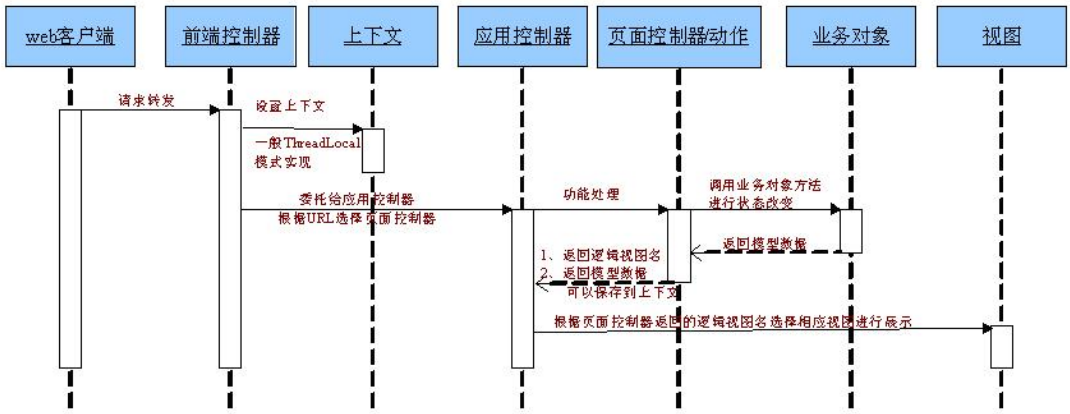
现在被绑定在 JSP，很难更换视图，比如 Velocity、FreeMarker；比如我要支持 Excel、PDF 视图等等。

6.服务到工作者：Front Controller + Application Controller + Page Controller + Context

即，前端控制器+应用控制器+页面控制器（也有称其为动作）+上下文，也是 Web MVC，只是责任更加明确



运行流程：



职责：

Front Controller：前端控制器，负责为表现层提供统一访问点，从而避免 Model2 中出现的重复的控制逻辑（由前端控制器统一回调相应的功能方法，如前边的根据 submitFlag=login 转调 login 方法）；并且可以为多个请求提供共用的逻辑（如准备上下文等等），将选择具体视图和具体的功能处理（如 login 里边封装请求参数到模型，并调用

业务逻辑对象）分离。

Application Controller：应用控制器，前端控制器分离选择具体视图和具体的功能处理之后，需要有人来管理，应用控制器就是用来选择具体视图技术（视图的管理）和具体的功能处理（页面控制器/命令对象/动作管理），一种策略设计模式的应用，可以很容易的切换视图/页面控制器，相互不产生影响。

Page Controller(Command)：页面控制器/动作/处理器：功能处理代码，收集参数、封装参数到模型，转调业务对象处理模型，返回逻辑视图名交给前端控制器（和具体的视图技术解耦），由前端控制器委托给应用控制器选择具体的视图来展示，可以是命令设计模式的实现。页面控制器也被称为处理器或动作。

Context：上下文，还记得 Model2 中为视图准备要展示的模型数据吗，我们直接放在 request 中（Servlet API 相关），有了上下文之后，我们就可以将相关数据放置在上下文，从而与协议无关（如 Servlet API）的访问/设置模型数据，一般通过 ThreadLocal 模式实现。

1.2 常用 MVC 框架

Strusts 和 SpringMVC:

•Struts 是Java Web MVC框架中不争的王者。经过长达九年的发展，Struts已经逐渐成长为一个稳定、成熟的框架，并且占有了MVC框架中最大的市场份额。但是Struts某些技术特性上已经落后于新兴的MVC框架。面对Spring MVC、Webwork2这些设计更精密，扩展性更强的框架，Struts受到了前所未有的挑战。

•Spring MVC通过一套MVC注解，让POJO成为处理请求的控制器，无须实现任何接口，同时，Spring MVC还支持REST风格的URL请求：注解驱动及REST风格的Spring MVC是Spring3.0最出彩的功能之一。此外，Sping MVC在数据绑定、视图解析、本地化处理及静态资源处理上都有许多不俗的表现。它在框架设计、扩展性、灵活性等方面全面超越了Struts、WebWork等MVC框架，从原来的追赶者一跃成为MVC的领跑者。

1.3 MVC 模式优缺点

优点：耦合性低、重用性高、生命周期成本低、部署快、可维护性高、有利于软件工程化管理

缺点：没有明确的定义，不适合小型，中等规模的应用程序、增加系统结构和实现的复杂性、视图与控制器间的过于紧密的连接、视图对模型数据的低效率访问、一般高级的界面工具或构造器不支持模

2. Spring MVC 简介

Spring Web MVC 是一种基于 Java 的实现了 Web MVC 设计模式的请求驱动类型的轻量级 Web 框架，即使用了 MVC 架构模式的思想，将 web 层进行职责解耦，基于请求驱动指的就是使用请求-响应模型，框架的目的就是帮助我们简化开发，Spring Web MVC 也是要简化我们日常 Web 开发的。

另外还有一种基于组件的、事件驱动的 Web 框架在此就不介绍了，如 Tapestry、JSF 等。

Spring Web MVC 也是服务到工作者模式的实现，但进行可优化。前端控制器是 DispatcherServlet；应用控制器其实拆为处理器映射器

(Handler Mapping)进行处理器管理和视图解析器(View Resolver)进行视图管理；页面控制器/动作/处理器为 Controller 接口（仅包含 ModelAndView handleRequest(request, response) 方法）的实现（也可以是任何的 POJO 类）；支持本地化（Local）解析、主题（Theme）解析及文件上传等；提供了非常灵活的数据验证、格式化和数据绑定机制；提供了强大的约定大于配置（惯例优先原则）的契约式编程支持

3. Spring MVC 作用

✓让我们能非常简单的设计出干净的 Web 层和薄薄的 Web 层；

✓进行更简洁的 Web 层的开发；

✓天生与 Spring 框架集成（如 IoC 容器、AOP 等）；

✓提供强大的约定大于配置的契约式编程支持；

✓能简单的进行 Web 层的单元测试；

✓支持灵活的 URL 到页面控制器的映射；

✓非常容易与其他视图技术集成，如 Velocity、FreeMarker 等等，因为模型数据不放在特定的 API 里，而是放在一个 Model 里（Map 数据结构实现，因此很容易被其他框架使用）；

✓非常灵活的数据验证、格式化和数据绑定机制，能使用任何对象进行数据绑定，不必实现特定框架的 API；

✓提供一套强大的 JSP 标签库，简化 JSP 开发；

✓支持灵活的本地化、主题等解析；

✓更加简单的异常处理；

✓对静态资源的支持；

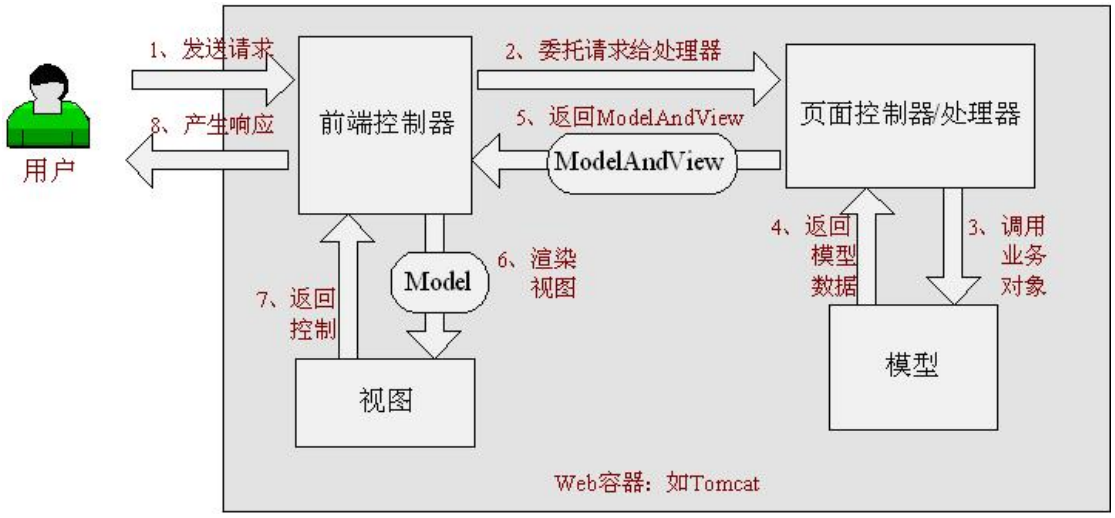
✓支持 Restful 风格。

4. SpringMVC 体系结构

SpringMVC 是基于 Model2（jsp+servlet+java bean）实现的框架

Spring Web MVC 框架也是一个基于请求驱动的 Web 框架，并且也使用了前端控制器模式来进行设计，再根据请求映射规则分发给相应的页面控制器（动作/处理器）进行处理。

4.1、Spring Web MVC 处理请求的流程：



具体执行步骤如下：

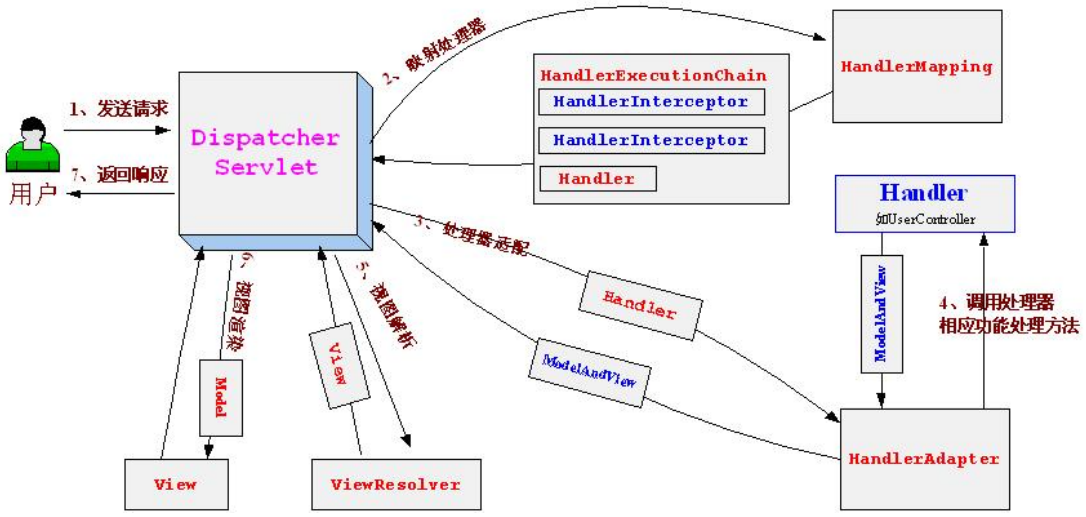
1、首先用户发送请求—>前端控制器，前端控制器根据请求信息（如 URL）来决定选择哪一个页面控制器进行处理并把请求委托给它，即以前的控制器的控制逻辑部分；图中的 1、2 步骤；

2、页面控制器接收到请求后，进行功能处理，首先需要收集和绑定请求参数到一个对象，这个对象在 Spring Web MVC 中叫命令对象，并进行验证，然后将命令对象委托给业务对象进行处理；处理完毕后返回一个 ModelAndView（模型数据和逻辑视图名）；图中的 3、4、5 步骤；

3、前端控制器收回控制权，然后根据返回的逻辑视图名，选择相应的视图进行渲染，并把模型数据传入以便视图渲染；图中的步骤 6、7；

4、前端控制器再次收回控制权，将响应返回给用户，图中的步骤 8；至此整个结束。

4.2. Spring Web MVC 架构



4.2.1 核心架构的具体流程步骤：

1、首先用户发送请求—>DispatcherServlet，前端控制器收到请求后自己不进行处理，而是委托给其他的解析器进行处理，作为统一访问

点，进行全局的流程控制；

2、DispatcherServlet—>HandlerMapping，HandlerMapping 将会把请求映射为 HandlerExecutionChain 对象（包含一个 Handler 处理器（页面控制器）对象、多个 HandlerInterceptor 拦截器）对象，通过这种策略模式，很容易添加新的映射策略；

3、DispatcherServlet——>HandlerAdapter，HandlerAdapter 将会把处理器包装为适配器，从而支持多种类型的处理器，即适配器设计模式的应用，从而很容易支持很多类型的处理器；

4、HandlerAdapter——>处理器功能处理方法的调用，HandlerAdapter 将会根据适配的结果调用真正的处理器的功能处理方法，完成功能处理；并返回一个 ModelAndView 对象（包含模型数据、逻辑视图名）；

5、ModelAndView 的逻辑视图名——> ViewResolver，ViewResolver 将把逻辑视图名解析为具体的 View，通过这种策略模式，很容易更换其他视图技术；

6、View——>渲染，View 会根据传进来的 Model 模型数据进行渲染，此处的 Model 实际是一个 Map 数据结构，因此很容易支持其他视图技术；

7、返回控制权给 DispatcherServlet，由 DispatcherServlet 返回响应给用户，到此一个流程结束。

此处我们只是讲了核心流程，没有考虑拦截器、本地解析、文件上传解析等，后边再细述。

4.2.2 几个问题：

1、请求如何给前端控制器？这个应该在 web.xml 中进行部署描述，在 HelloWorld 中详细讲解。

2、前端控制器如何根据请求信息选择页面控制器进行功能处理？

我们需要配置 HandlerMapping 进行映射

3、如何支持多种页面控制器呢？

配置 HandlerAdapter 从而支持多种类型的页面控制器

4、如何页面控制器如何使用业务对象？

可以预料到，肯定利用 Spring IoC 容器的依赖注入功能

5、页面控制器如何返回模型数据？

使用 ModelAndView 返回

6、前端控制器如何根据页面控制器返回的逻辑视图名选择具体的视图进行渲染？

使用 ViewResolver 进行解析

7、不同的视图技术如何使用相应的模型数据？

因为 Model 是一个 Map 数据结构，很容易支持其他视图技术

4.2.3 核心开发步骤：

1、DispatcherServlet 在 web.xml 中的部署描述，从而拦截请求到 Spring Web MVC

2、HandlerMapping 的配置，从而将请求映射到处理器

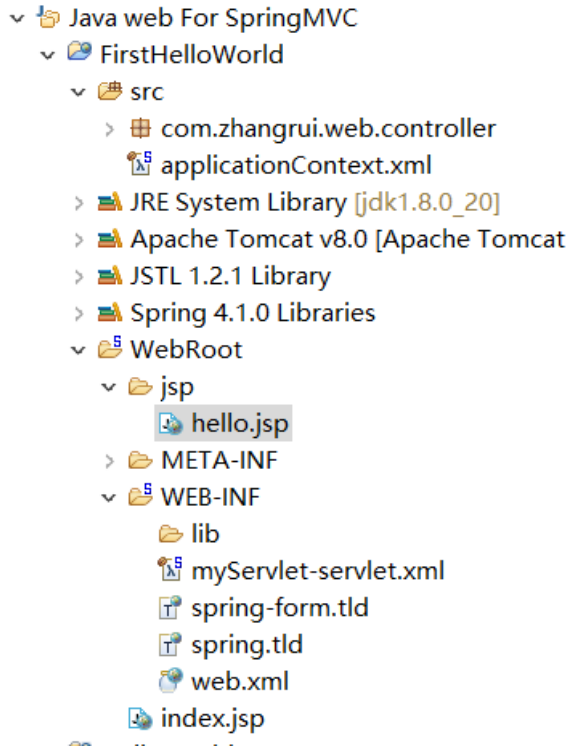
3、HandlerAdapter 的配置，从而支持多种类型的处理器

4、viewResolver 的配置，从而将逻辑视图名解析为具体视图技术

5、　处理器（页面控制器）的配置，从而进行功能处理

5. HelloWorld 入门

工程完整结构图



5.1、配置框架

依赖 Spring 框架，新建 web Project,导入 Spring 框架

5.2、前端控制器的配置

在 web.xml 中配置 DispatcherServlet

```
<servlet>
   <servlet-name>myServlet</servlet-name>
   <servletclass>org.springframework.web.servlet.DispatcherSer
vlet</servlet-class>
```

```
    <load-on-startup>1</load-on-startup>
 </servlet>
    <servlet-mapping>
        <servlet-name> myServlet </servlet-name>
        <url-pattern>/*</url-pattern>
</servlet-mapping>
```

load-on-startup：表示启动容器时初始化该 Servlet；

url-pattern：表示哪些请求交给 Spring Web MVC 处理，“/*”是用来定义默认 servlet 映射的。也可以如“*.html”表示拦截所有以 html 为扩展名的请求。

自此请求已交给 Spring Web MVC 框架处理，因此我们需要配置 Spring 的配置文件，默认 DispatcherServlet 会加载 WEB-INF/[DispatcherServlet 的 Servlet 名字]-servlet.xml 配置文件。本示例为 WEB-INF/ myServlet -servlet.xml。

DispatcherServlet 也可以配置自己的初始化参数，覆盖默认配置：

参数	描述
contextClass	实现 WebApplicationContext 接口的类，当前的 servlet 用它来创建上下文。如果这个参数没有指定，默认使用 XmlWebApplicationContext。
contextConfigLocation	传给上下文实例（由 contextClass 指定）的字符串，用来指定上下文的位置。这个字符串可以被分成多个字符串（使用逗号作为分隔符）来支持多个上下文（在多上下文的情况下，如

	果同一个 bean 被定义两次，后面一个优先)。
namespace	WebApplicationContext 命名空间。默认值是 [server-name]-servlet。

Java代码

```
1. <servlet>
2.     <servlet-name>chapter2</servlet-name>
3.     <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
4.     <load-on-startup>1</load-on-startup>
5.     <init-param>
6.         <param-name>contextConfigLocation</param-name>
7.         <param-value>classpath:spring-servlet-config.xml</param-value>
8.     </init-param>
9. </servlet>
```

如果使用如上配置，Spring Web MVC框架将加载“classpath:spring-servlet-config.xml”来进行初始化上下文而不是“/WEB-INF/[servlet名字]-servlet.xml”。

5.3. 配置 HandlerMapping、HandlerAdapter

在 WEB-INF/ 下新建 xml 文件 myServlet -servlet.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:p="http://www.springframework.org/schema/p"

xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-
4.0.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-
4.0.xsd">

</beans>
```

在 myServlet -servlet.xml 添加代码：

```
<!-- HandlerMapping -->
<bean
class="org.springframework.web.servlet.handler.BeanNameUrlHand
lerMapping"/>
<!-- HandlerAdapter -->
<bean
class="org.springframework.web.servlet.mvc.SimpleControllerHan
dlerAdapter"/>
```

BeanNameUrlHandlerMapping：表示将请求的 URL 和 Bean 名字映射，如 URL 为“上下文/hello”，则 Spring 配置文件必须有一个名字为“/hello”的 Bean，上下文默认忽略。

SimpleControllerHandlerAdapter ： 表 示 所 有 实 现 了 org.springframework.web.servlet.mvc.Controller 接口的 Bean 可以作为 Spring Web MVC 中的处理器。如果需要其他类型的处理器可以通过实现 HadlerAdapter 来解决。

5.4．配置 ViewResolver

在 myServlet -servlet.xml 添加代码：

```
<bean
class="org.springframework.web.servlet.view.InternalResourceVi
ewResolver">
   <property name="viewClass"
value="org.springframework.web.servlet.view.JstlView"/>
    <property name="prefix" value="/jsp/"/>
    <property name="suffix" value=".jsp"/>
</bean>
```

InternalResourceViewResolver：用于支持 Servlet、JSP 视图解析；

viewClass：JstlView 表示 JSP 模板页面需要使用 JSTL 标签库，classpath 中必须包含 jstl 的相关 jar 包；

prefix 和 suffix：查找视图页面的前缀和后缀（前缀[逻辑视图名]后

缀），比如传进来的逻辑视图名为 hello，则该该 jsp 视图页面应该存放在“/jsp/hello.jsp”；

5.5 编写开发处理器/页面控制器

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.Controller;//导这个包
public class HelloWorldController implements Controller {
    @Override
    public ModelAndView handleRequest(HttpServletRequest arg0,
            HttpServletResponse arg1) throws Exception {
            //1、收集参数、验证参数
            //2、绑定参数到命令对象
            //3、将命令对象传入业务对象进行业务处理
            //4、选择下一个页面
             ModelAndView mv = new ModelAndView();
             //添加模型数据 可以是任意的POJO对象
              mv.addObject("message", "Hello World!");
              //设置逻辑视图名，视图解析器会根据该名字解析到具体的视图页面
              mv.setViewName("hello");
              return mv;
    }
}
```

org.springframework.web.servlet.mvc.Controller：页面控制器/处理器必须实现 Controller 接口

public ModelAndView handleRequest(HttpServletRequest req, HttpServletResponse resp)：功能处理方法，实现相应的功能处理，比如收集参数、验证参数、绑定参数到命令对象、将命令对象传入业务对象进行业务处理、最后返回 ModelAndView 对象；

ModelAndView：包含了视图要实现的模型数据和逻辑视图名；

“mv.addObject("message", "Hello World!");”表示添加模型数据，此处可以是任意 POJO 对象；

“mv.setViewName("hello");”表示设置逻辑视图名为“hello”，视图解析器会将其解析为具体的视图，如前边的视图解析器 InternalResourceViewResolver

会将其解析为“/jsp/hello.jsp”。

我们需要将其添加到 Spring 配置文件(WEB-INF/myServlet-servlet.xml)，让其接受 Spring IoC 容器管理：

```
<!-这里必须加，扫描包-->
<context:component-scan base-
package="com.zhangrui.web.controller"></context:component-
scan>
<bean name="/hello"
class="com.zhangrui.web.controller.HelloWorldController"/>
```

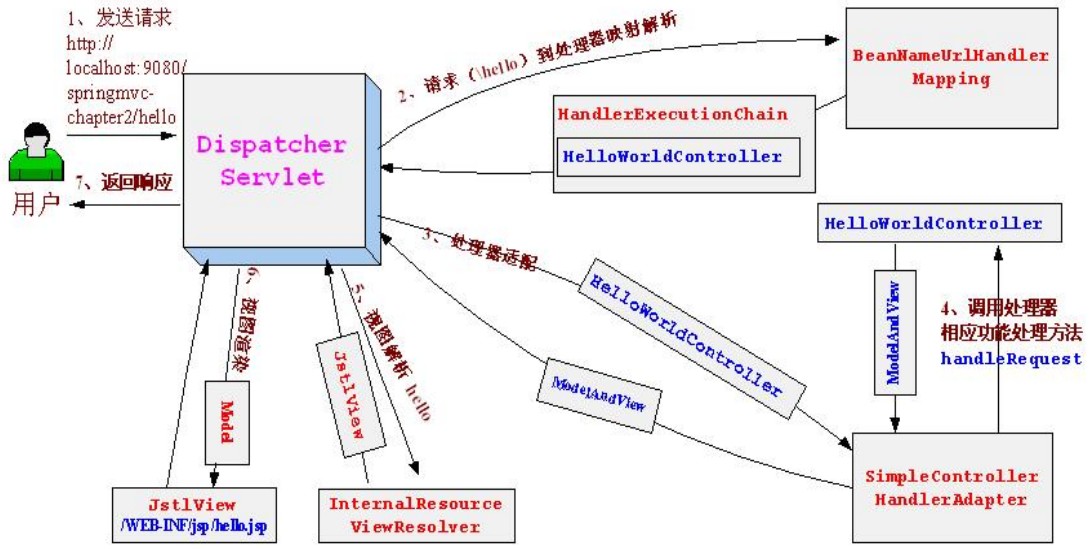
name="/hello"：前边配置的 BeanNameUrlHandlerMapping，表示如过请求的 URL 为“上下文/hello”，则将会交给该 Bean 进行处理。

5.6 视图层

创建 /jsp/hello.jsp 视图页面：

```
<body>
  ${message}<br>
</body>
```

5.7 运行流程分析



运行步骤：

1. 首先用户发送请求 http://localhost:8080/FirstHelloWorld/jsp/hello.do——>web 容器，web 容器根据“/hello.do”路径映射到 DispatcherServlet（url-pattern 为/*）进行处理；

2、DispatcherServlet——>BeanNameUrlHandlerMapping 进行请求到处理的映射，BeanNamxUrlHandlerMapping 将“/jsp/hello.do”路径直接映射到名字为“/jsp/hello.do”的 Bean 进行处理，即 HelloWorldController，BeanNameUrlHandlerMapping 将其包装为 HandlerExecutionChain（只包括 HelloWorldController 处理器，没有拦截器）；

3、DispatcherServlet——> SimpleControllerHandlerAdapter，SimpleControllerHandlerAdapter 将 HandlerExecutionChain 中的处理器（HelloWorldController）适配为

SimpleControllerHandlerAdapter；

4、 SimpleControllerHandlerAdapter——> HelloWorldController 处理器功能处理方法的调用，SimpleControllerHandlerAdapter 将会调用处理器的 handleRequest 方法进行功能处理，该处理方法返回一个 ModelAndView 给 DispatcherServlet；

5、hello（ModelAndView 的逻辑视图名）——>InternalResourceViewResolver，InternalResourceViewResolver 使用 JstlView，具体视图页面在/ jsp/hello.jsp；

6、JstlView（/jsp/hello.jsp）——>渲染，将在处理器传入的模型数据(message=HelloWorld！)在视图中展示出来；

7、返回控制权给 DispatcherServlet，由 DispatcherServlet 返回响应给用户，到此一个流程结束。

5.8 POST 中文乱码解决方案

spring Web MVC 框架提供了 org.springframework.web.filter.CharacterEncodingFilter 用于解决 POST 方式造成的中文乱码问题

```
<filter>
<filter-name>CharacterEncodingFilter</filter-name>
<filter-class>
org.springframework.web.filter.CharacterEncodingFilter
</filter-class>
 <init-param>
      <param-name>encoding</param-name>
      <param-value>utf-8</param-value>
  </init-param>
```

```
</filter>
<filter-mapping>
   <filter-name>CharacterEncodingFilter</filter-name>
     <url-pattern>/*</url-pattern>
</filter-mapping>
```

6. Spring3.1 新特性

一、Spring2.5 之前，我们都是通过实现 Controller 接口或其实现来定义我们的处理器类。

二、Spring2.5 引入注解式处理器支持，通过@Controller 和@RequestMapping 注解定义我们的处理器类。并且提供了一组强大的注解：

需要通过处理器映射 DefaultAnnotationHandlerMapping 和处理器适配器 AnnotationMethodHandlerAdapter 来开启支持@Controller 和 @RequestMapping 注解的处理器。

@Controller：用于标识是处理器类；

@RequestMapping：请求到处理器功能方法的映射规则；

@RequestParam：请求参数到处理器功能处理方法的方法参数上的绑定；

@ModelAttribute：请求参数到命令对象的绑定；

@SessionAttributes：用于声明 session 级别存储的属性，放置在处理器类上，通常列出模型属性（如@ModelAttribute）对应的名称，则这些属性会透明的保存到 session 中；

@InitBinder：自定义数据绑定注册支持，用于将请求参数转换到命

令对象属性的对应类型；

三、Spring3.0 引入 RESTful 架构风格支持(通过@PathVariable 注解和一些其他特性支持),且又引入了更多的注解支持：

@CookieValue：cookie 数据到处理器功能处理方法的方法参数上的绑定；

@RequestHeader：请求头（header）数据到处理器功能处理方法的方法参数上的绑定；

@RequestBody：请求的 body 体的绑定（通过 HttpMessageConverter 进行类型转换）；

@ResponseBody：处理器功能处理方法的返回值作为响应体（通过 HttpMessageConverter 进行类型转换）；

@ResponseStatus：定义处理器功能处理方法/异常处理器返回的状态码和原因；

@ExceptionHandler：注解式声明异常处理器；

@PathVariable：请求 URI 中的模板变量部分到处理器功能处理方法的方法参数上的绑定，从而支持 RESTful 架构风格的 URI；

四、还有比如：

JSR-303 验证框架的无缝支持（通过@Valid 注解定义验证元数据）；

使用 Spring 3 开始的 ConversionService 进行类型转换（PropertyEditor 依然有效），支持使用@NumberFormat 和 @DateTimeFormat 来进行数字和日期的格式化；

HttpMessageConverter（Http 输入/输出转换器，比如 JSON、XML

等的数据输出转换器）；

ContentNegotiatingViewResolver，内容协商视图解析器，它还是视图解析器，只是它支持根据请求信息将同一模型数据以不同的视图方式展示（如 json、xml、html 等），RESTful 架构风格中很重要的概念（同一资源，多种表现形式）；

Spring 3 引入一个 mvc XML 的命名空间用于支持 mvc 配置，包括如：

<mvc:annotation-driven>：

自动注册基于注解风格的处理器需要的 DefaultAnnotationHandlerMapping 、AnnotationMethodHandlerAdapter

支持 Spring3 的 ConversionService 自动注册

支持 JSR-303 验证框架的自动探测并注册（只需把 JSR-303 实现放置到 classpath）

自动注册相应的 HttpMessageConverter（用于支持@RequestBody 和 @ResponseBody）（如 XML 输入输出转换器（只需将 JAXP 实现放置到 classpath）、JSON 输入输出转换器（只需将 Jackson 实现放置到 classpath））等。

<mvc:interceptors>：注册自定义的处理器拦截器；

<mvc:view-controller>：和 ParameterizableViewController 类似，收到相应请求后直接选择相应的视图；

<mvc:resources>：逻辑静态资源路径到物理静态资源路径的支持；

<mvc:default-servlet-handler>：当在 web.xml 中 DispatcherServlet 使用<url-pattern>/</url-pattern> 映射时，能映射静态资源（当 Spring Web MVC 框架没有处理请求对应的控制器时（如一些静态资源），转交给默认的 Servlet 来响应静态文件，否则报 404 找不到资源错误，）。

……等等。

五、Spring3.1 新特性：

对 Servlet 3.0 的全面支持。

@EnableWebMvc：用于在基于 Java 类定义 Bean 配置中开启 MVC 支持，和 XML 中的<mvc:annotation-driven>功能一样；

新的@Contoller 和@RequestMapping 注解支持类：处理器映射 RequestMappingHandlerMapping 和 处理器适配器 RequestMappingHandlerAdapter 组合来代替 Spring2.5 开始的处理器映射 DefaultAnnotationHandlerMapping 和处理器适配器 AnnotationMethodHandlerAdapter，提供更多的扩展点，它们之间的区别我们在处理器映射一章介绍。

新的 @ExceptionHandler 注解支持类：ExceptionHandlerExceptionResolver 来代替 Spring3.0 的 AnnotationMethodHandlerExceptionResolver，在异常处理器一章我们再详细讲解它们的区别。

@RequestMapping 的"consumes" 和 "produces" 条件支持：用于支持@RequestBody 和 @ResponseBody，

1、consumes 指定请求的内容是什么类型的内容，即本处理方法消费什么类型的数据，如 consumes="application/json"表示 JSON 类型的内容，Spring 会根据相应的 HttpMessageConverter 进行请求内容区数据到@RequestBody 注解的命令对象的转换；

2、produces 指定生产什么类型的内容，如 produces="application/json"表示 JSON 类型的内容，Spring 的根据相应的 HttpMessageConverter 进行请求内容区数据到@RequestBody 注解的命令对象的转换，Spring 会根据相应的 HttpMessageConverter 进行模型数据（返回值）到 JSON 响应内容的转换。

URI 模板变量增强：URI 模板变量可以直接绑定到@ModelAttribute 指定的命令对象、@PathVariable 方法参数在视图渲染之前被合并到模型数据中（除 JSON 序列化、XML 混搭场景下）。

lidated：JSR-303 的 javax.validation.Valid 一种变体（非 JSR-303 规范定义的，而是 Spring 自定义的），用于提供对 Spring 的验证器（org.springframework.validation.Validator）支持，需要 Hibernate Validator 4.2 及更高版本支持；

@RequestPart：提供对“multipart/form-data”请求的全面支持，支持 Servlet 3.0 文件上传（javax.servlet.http.Part）、支持内容的 HttpMessageConverter（即根据请求头的 Content-Type，来判断内容区数据是什么类型，如 JSON、XML，能自动转换为命令对象），比@RequestParam 更强大（只能对请求参数数据绑定，key-alue 格式），

而@RequestPart 支持如 JSON、XML 内容区数据的绑定；

Flash 属性 和 RedirectAttribute：通过 FlashMap 存储一个请求的输出，当进入另一个请求时作为该请求的输入，典型场景如重定向（POST-REDIRECT-GET 模式，1、POST 时将下一次需要的数据放在 FlashMap；2、重定向；3、通过 GET 访问重定向的地址，此时 FlashMap 会把 1 放到 FlashMap 的数据取出放到请求中，并从 FlashMap 中删除；从而支持在两次请求之间保存数据并防止了重复表单提交）。

Spring Web MVC 提供 FlashMapManager 用于管理 FlashMap，默认使用 SessionFlashMapManager，即数据默认存储在 session 中

7. DispatcherServlet 详解

7.1 DispatcherServlet 作用

DispatcherServlet 是前端控制器设计模式的实现，提供 Spring Web MVC 的集中访问点，而且负责职责的分派，而且与 Spring IoC 容器无缝集成，从而可以获得 Spring 的所有好处。

DispatcherServlet 主要用作职责调度工作，本身主要用于控制流程，主要职责如下：

1、文件上传解析，如果请求类型是 multipart 将通过 MultipartResolver 进行文件上传解析；

2、通过 HandlerMapping，将请求映射到处理器（返回一个

HandlerExecutionChain，它包括一个处理器、多个HandlerInterceptor 拦截器)；

3、通过 HandlerAdapter 支持多种类型的处理器(HandlerExecutionChain 中的处理器)；

4、通过 ViewResolver 解析逻辑视图名到具体视图实现；

5、本地化解析；

6、渲染具体的视图等；

7、如果执行过程中遇到异常将交给 HandlerExceptionResolver 来解析。

从以上我们可以看出 DispatcherServlet 主要负责流程的控制(而且在流程中的每个关键点都是很容易扩展的)。

7.2 上下文关系

String 在 web.xml 通用配置

```
<listener>
    <listener-
class>org.springframework.web.context.ContextLoaderListener</l
istener-class>
  </listener>
  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath:applicationContext.xml</param-value>
  </context-param>
```

其中 applicationContext.xml 位于 src 目录下

contextConfigLocation：表示用于加载 Bean 的配置文件；

contextClass：表示用于加载 Bean 的 ApplicationContext 实现类，默

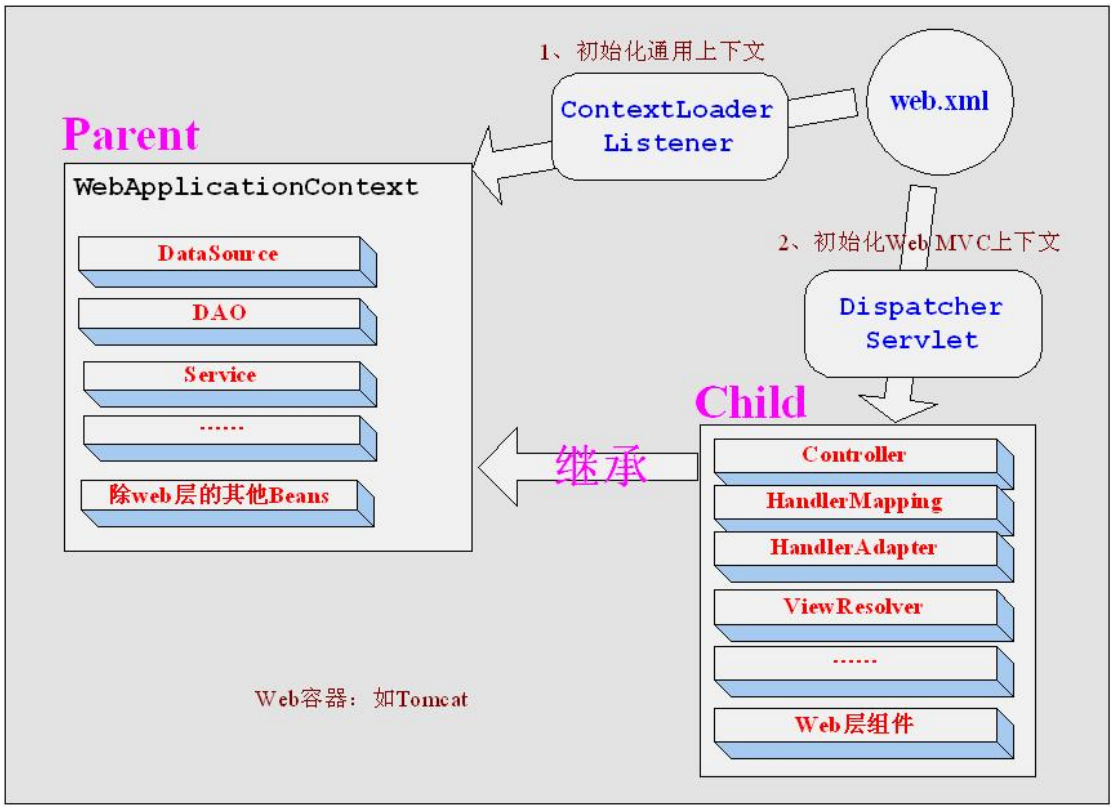
认 WebApplicationContext。

创建完毕后会将该上下文放在

ServletContext：

servletContext.setAttribute(
WebApplicationContext.ROOT_WEB_APPLICATION_CONTEXT_ATTRIBUTE,this.context);

ContextLoaderListener 初始化的上下文和 DispatcherServlet 初始化的上下文关系



从图中可以看出：

ContextLoaderListener 初始化的上下文加载的 Bean 是对于整个应用程序共享的，不管是使用什么表现层技术，一般如 DAO 层、Service 层 Bean；

DispatcherServlet 初始化的上下文加载的 Bean 是只对 Spring Web MVC 有效的 Bean，如 Controller、HandlerMapping、HandlerAdapter

等等，该初始化上下文应该只加载 Web 相关组件。

7.3 DispatcherServlet 初始化顺序

继承体系结构如下所示：

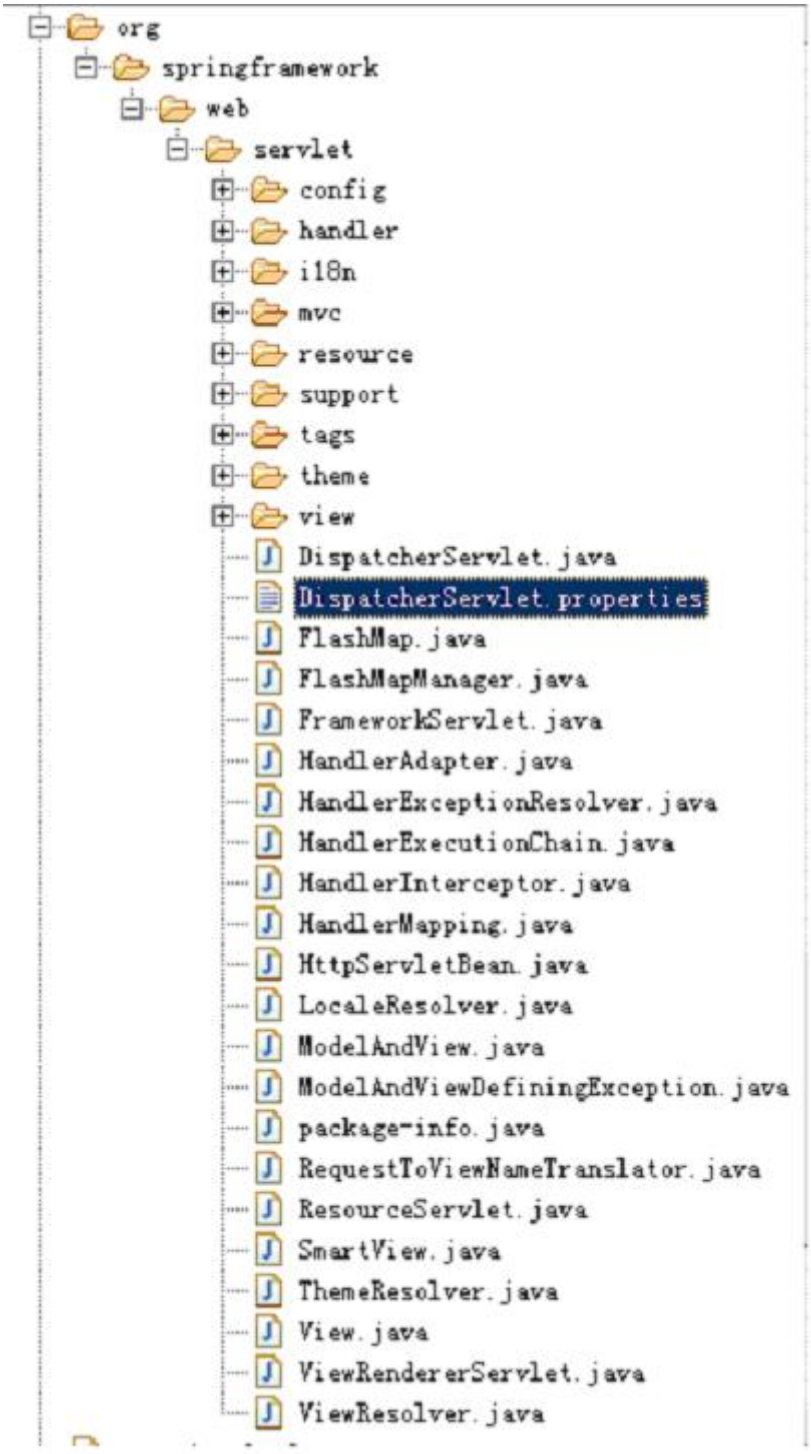
DispatcherServlet 启动时会进行我们需要的 Web 层 Bean 的配置，如 HandlerMapping、HandlerAdapter 等，而且如果我们没有配置，还会给我们提供默认的配置。整个 DispatcherServlet 初始化的过程具体主要做了如下两件事情：

1、初始化 Spring Web MVC 使用的 Web 上下文，并且可能指定父容器为（ContextLoaderListener 加载了根上下文）；

2、初始化 DispatcherServlet 使用的策略，如 HandlerMapping、HandlerAdapter 等。

7.3 DispatcherServlet 默认配置

DispatcherServlet 的默认配置在 DispatcherServlet.properties（和 DispatcherServlet 类在一个包下）中，而且是当 Spring 配置文件中没有指定配置时使用的默认策略：



7.3 DispatcherServlet 中使用的特殊的 Bean

1、Controller：处理器/页面控制器，做的是 MVC 中的 C 的事情，但控制逻辑转移到前端控制器了，用于对请求进行处理；

2、HandlerMapping：请求到处理器的映射，如果映射成功返回一个

HandlerExecutionChain 对象（包含一个 Handler 处理器（页面控制器）对象、多个 HandlerInterceptor 拦截器）对象；如 BeanNameUrlHandlerMapping 将 URL 与 Bean 名字映射，映射成功的 Bean 就是此处的处理器；

3、HandlerAdapter：HandlerAdapter 将会把处理器包装为适配器，从而支持多种类型的处理器，即适配器设计模式的应用，从而很容易支持很多类型的处理器；如 SimpleControllerHandlerAdapter 将对实现了 Controller 接口的 Bean 进行适配，并且掉处理器的 handleRequest 方法进行功能处理；

4、ViewResolver：ViewResolver 将把逻辑视图名解析为具体的 View，通过这种策略模式，很容易更换其他视图技术；如 InternalResourceViewResolver 将逻辑视图名映射为 jsp 视图；

5、LocalResover：本地化解析，因为 Spring 支持国际化，因此 LocalResover 解析客户端的 Locale 信息从而方便进行国际化；

6、ThemeResovler：主题解析，通过它来实现一个页面多套风格，即常见的类似于软件皮肤效果；

7、MultipartResolver：文件上传解析，用于支持文件上传；

8、HandlerExceptionResolver：处理器异常解析，可以将异常映射到相应的统一错误界面，从而显示用户友好的界面（而不是给用户看到具体的错误信息）；

9、RequestToViewNameTranslator：当处理器没有返回逻辑视图名等相关信息时，自动将请求 URL 映射为逻辑视图名；

10、FlashMapManager：用于管理 FlashMap 的策略接口，FlashMap 用于存储一个请求的输出，当进入另一个请求时作为该请求的输入，通常用于重定向场景。