SpringMVC源码剖析（二）- DispatcherServlet的前世今生 - 相见欢

上一篇文章《SpringMVC源码剖析（一）- 从抽象和接口说起》中，我介绍了一次典型的SpringMVC请求处理过程中，相继粉墨登场的各种核心类和接口。我刻意忽略了源码中的处理细节，只列出最简单的类甚至是接口类，目的就是让大家先从最高层次的抽象意义上来审视SpringMVC这个框架；我也刻意将SpringMVC和Struts2做对比，目的是让大家看到，SpringMVC究竟吸取了Sturts2设计思想中的哪些精华，又弥补了它的哪些遗憾。

DispatcherServlet作为SpringMVC的核心之中的核心类，再怎么强调它的重要性也不为过。SpringMVC所有的核心类和接口，都密集地出现在DispatcherServlet的源码中，SpringMVC源码剖析，很大程度上可以说也是在剖析DispatcherServlet这一个类。这一篇文章里，我先说几点关于DispatcherServlet的前世今生，希望能帮助你更好的理解它。

1.对扩展开放，对修改封闭

SpringMVC是一个基于著名的Open-Closed，即开闭原则进行设计的框架。在Spring官方文档里面关于SpringMVC的介绍开宗明义地进行了说明：

```
A key design principle in Spring Web MVC and in Spring in general is the “O
```

开闭原则是一个很宽泛的原则，具体体现到DispatcherServlet的源码中，我们可以大致摸得到一些线索：

- 类中所有的变量声明，几乎都以接口的形式给出，并没有绑定在具体的实现类上。
- 使用模版方法模式，在父类中对基础行为进行定义，让子类实现模版方法扩展行为。

其中第一点，在一个框架的设计中尤为重要，也是贯彻开闭原则最重要的一点。因为当你通过一些高层次的接口或者抽象类，将一个类完成的逻辑或流程编写完成后（具体点说，是通过一个接口的引用调用接口方法），整个逻辑或流程的功能就被确实的在源码中固定下来了。可是这时，这些接口或抽

象类的具体实现者是谁，还没有固定！这就给了你的系统或框架近乎无限的扩展性，因为你可以任意安排和实现这些类。

我认为，面向对象设计的精髓，是对现实世界中“行为和契约”的描述。这个“行为和契约”，体现在接口和抽象类的方法声明中。软件设计师要用面向对象的眼光去观察和抽象这个世界中的事物，这里的事物可以是一些商业逻辑、可以是一些处理流程，然后用高层次的接口去描述这些行为和契约。当你在越抽象的层次上将这些行为和契约描述清楚后，你所设计的系统就是越符合开闭原则的。

SpringMVC框架在面向对象设计上，做出了绝佳的示范。它通过高度抽象的接口，描述出了一次请求处理的流程，从而让整个框架从一开始就是符合开闭原则的。同时它也提供了这些接口的一系列默认实现类，让你不需要很复杂的配置，就能很好的使用SpringMVC进行开发。抽象的确是个利器，但是框架绝不能运行在空中楼阁中，SpringMVC提供的的这一系列默认实现类必须要有容身之所。聪明的你可能早已想到：Spring IOC容器。这就引出了我要说的第二点。

2.配置元素的对象化

所有的框架，都需要有这样一个功能，叫做：配置元素的对象化。因为几乎所有的框架，都将配置元素集中到外部的xml配置文件中，然后在框架的初始化流程中，对这些配置文件进行解析，再变成java世界中的一个个对象供框架使用，这整个过程，可以被称为配置元素的对象化。为什么要有配置文件呢？这个问题的回答也是很简单，因为没有人会想要使用一个配置散布在框架中各个java类源码里面的框架。框架也不允许使用者这样子做，因为框架在发布的时候，提供的是一个个jar包文件，jar包内是已经编译好的class文件。配置文件由使用者外部提供，框架对它进行解析，使用者能得到集中配置的好处，框架也乐于这样子，可以说是合情合理。

那么作为Spring产品族的新成员，SpringMVC在设计的时候，相信设计者们不做它想，这一个“配置元素的对象化”功能既然不可避免，那么使用Spring IOC容器，通过bean配置文件来配置SpringMVC，绝对是不二之选。不可能像Struts2一样，内部再搞一个别的容器，因为Spring容器本身已经是被高度设计，而且已经在java世界获得巨大成功。从推广的角度上来说，如果对spring容器的所有知识，都可以完整的应用到SpringMVC，那么对于开发者

无疑是一个极大的吸引力。

剩下的问题就只有：到底该如何将Spring容器和SpringMVC的初始化过程整合起来呢？

答案就是WebApplicationContext接口，更具体点说，是XmlWebApplicationContext这个Spring上下文实现类。SpringMVC也使用了这一个为了将Spring容器和Web环境整合而特意设计的Spring上下文类。我们打开WebApplicationContext的源码：

```
package org.springframework.web.context;

import javax.servlet.ServletContext;

import org.springframework.context.ApplicationContext;

public interface WebApplicationContext extends ApplicationContext {

        String ROOT_WEB_APPLICATION_CONTEXT_ATTRIBUTE = WebApplicationConte

        String SCOPE_REQUEST = "request";

        String SCOPE_SESSION = "session";

        String SCOPE_GLOBAL_SESSION = "globalSession";

        String SCOPE_APPLICATION = "application";

        String SERVLET_CONTEXT_BEAN_NAME = "servletContext";

        String CONTEXT_PARAMETERS_BEAN_NAME = "contextParameters";

        String CONTEXT_ATTRIBUTES_BEAN_NAME = "contextAttributes";

        ServletContext getServletContext();

}
```

发现它是继承于ApplicationContext这个普通Spring容器所使用的上下文接口类，除了一些常量的声明，只多了一个可以获取到ServletContext的getServletContext()方法。回到上面提到的“行为和契约的描述”上，我们可以大胆的断言，Spring容器和Web环境的整合，是在ServletContext上做文章。

打开所有使用了Spring的Web项目的web.xml文件，必定有这样一段配置：

```
<listener>
                <listener-class>org.springframework.web.context.ContextLoad
        </listener>
```

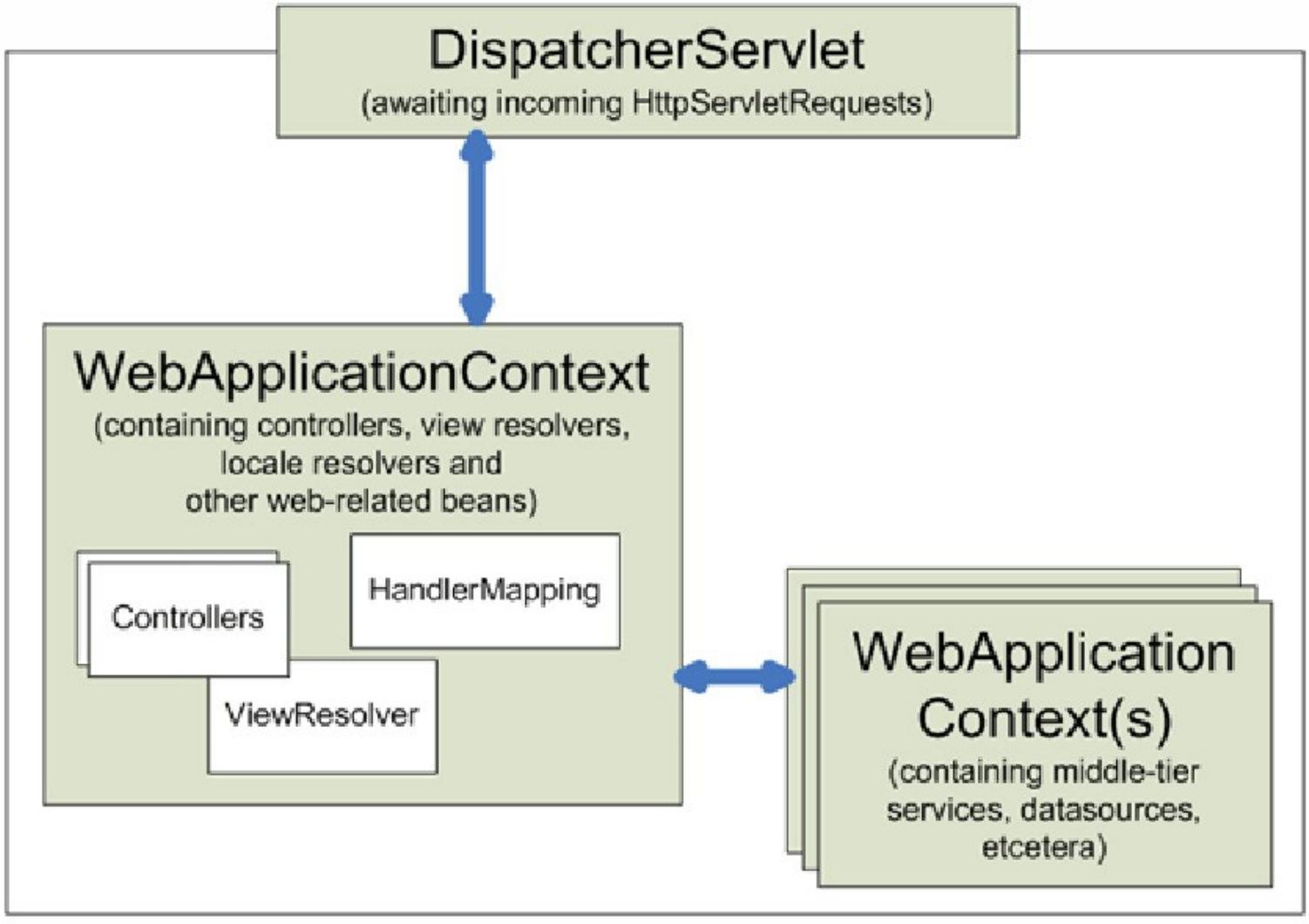
ContextLoaderListener实现了ServletContextListener接口，在Servlet容器启动的时候，会初始化一个WebApplicationContext的实现类，并将其作为ServletContext的一个属性设置到Servlet环境中，摘抄源码如下：

```
servletContext.setAttribute(WebApplicationContext.ROOT_WEB_APPLICATION_CONT
```

WebApplicationContext.ROOT_WEB_APPLICATION_CONTEXT_ATTRIBUTE的值，在上面WebApplicationContext的源码中的第一个常量中就被声明，是WebApplicationContext.class.getName() + ".ROOT"，更直接一点，它是“org.springframework.web.context.WebApplicationContext.ROOT”。ContextLoaderListener所初始化的这个Spring容器上下文，被称为根上下文。

SpringMVC在DispatcherServlet的初始化过程中，同样会初始化一个WebApplicationContext的实现类，作为自己独有的上下文，这个独有的上下文，会将上面的根上下文作为自己的父上下文，来存放SpringMVC的配置元素，然后同样作为ServletContext的一个属性，被设置到ServletContext中，只不过它的key就稍微有点不同，key和具体的DispatcherServlet注册在web.xml文件中的名字有关，从这一点也决定了，我们可以在web.xml文件中注册多个DispatcherServlet，因为Servlet容器中注册的Servlet名字肯定不一样，设置到Servlet环境中的key也肯定不同。

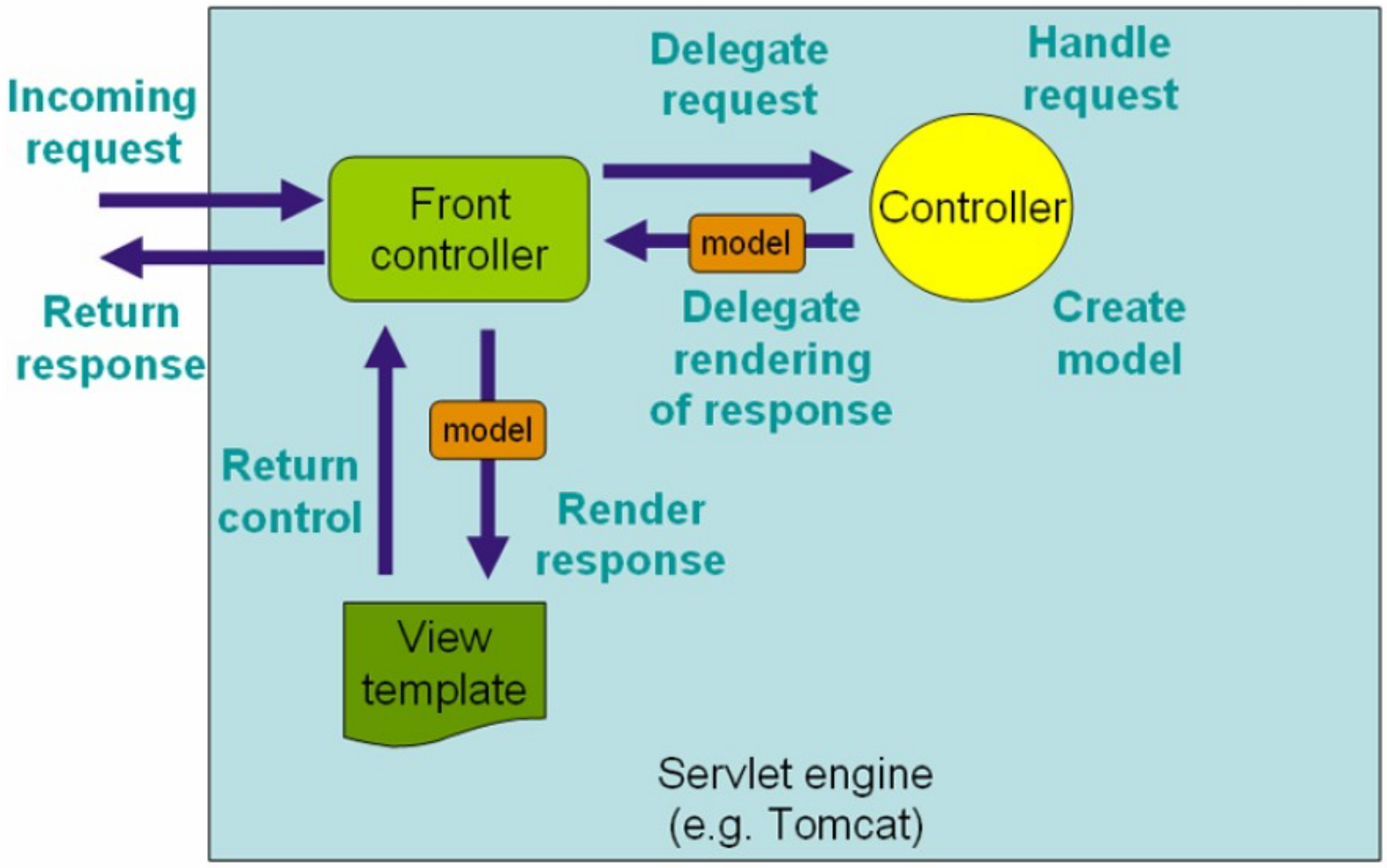
由于在Spring容器中，子上下文可以访问到所有父上下文中的信息，而父上下文访问不到子上下文的信息，这个根上下文，就很适合作为多个子上下文配置的集中点。以官方文档中的图来说明：



3.前端控制器

前端控制器，即所谓的Front Controller，体现的是设计模式中的前端控制器模式。前端控制器处理所有从用户过来的请求。所有用户的请求都要通过前端控制器。SpringMVC框架和其他请求驱动的表示层框架一样，也是围绕一个将请求分发到相应控制器的核心Servlet来设计的。DispatcherServlet和其他框架中的Servlet不一样的地方在于，它和Spring容器无缝整合在了一起，因此你可以在SpringMVC中使用Spring容器所有的特性。

DispatcherServlet这个前端控制器，在SpringMVC中的作用，以官方文档中的配图来说明：



整个流程可以被大致描述为：一个http请求到达服务器，被DispatcherServlet接收。DispatcherServlet将请求委派给合适的处理器Controller，此时处理控制权到达Controller对象。Controller内部完成请求的数据模型的创建和业务逻辑的处理，然后再将填充了数据后的模型即model和控制权一并交还给DispatcherServlet，委派DispatcherServlet来渲染响应。DispatcherServlet再将这些数据和适当的数据模版视图结合，向Response输出响应。

可以看到Model-View-Controller这三样东西协同合作，共同体现出MVC的设计理念，三个层次可以分别独立演化，整个系统架构又清晰又简洁。这是SpringMVC为我们描述的美好愿景，后面我们也将看到，SpringMVC为了实现这一承诺，究竟做出了什么样的努力。

标签： SpringMVC