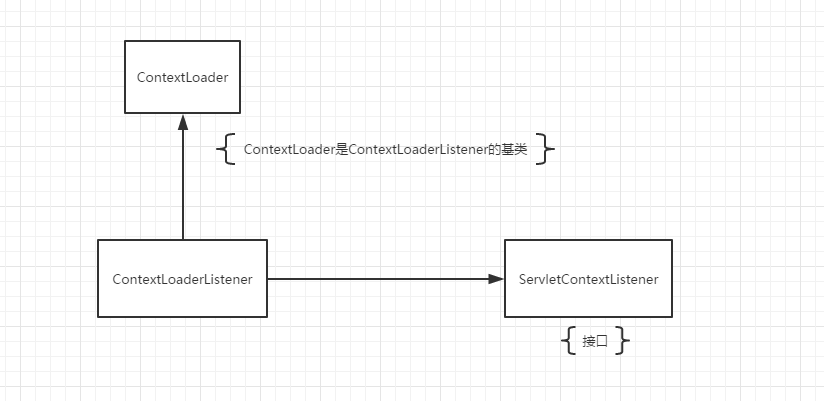
1. web.xml是入口文件(引用一下web.xml文件)
2. ioc是spring 的模块，如果在web容器中想要用的话，需要的是srping设计一个启动程序，将ioc容器导入进去。总的来说，启动过程分两部分，一个是web容器的启动，另一个是ioc容器的导入和初始化。
3. DispatcherServlet：用来处理请求分发

这边有一段描述：DispatcherServlet和ContextLoaderListener提供了在Web容器中对Spring的接口，ServletContext为Spring的IOC容器提供了一个宿主环境，在宿主环境中，SpringMvc建立起一个Ioc的容器体系。这个Ioc的容器体系是通过ContextLoaderListener的初始化建立起来的，在建立IOC容器体系之后，把DispatcherServlet作为SpringMVC处理web请求的处理器建立起来(请求的分发)，

至此，建立在Ioc容器基础上的SpringMVC就可以正常的开始作用了

1. IOC容器的启动过程就是建立上下文的过程，该上下文是与ServletContext相伴而生的，由ContextLoaderListener启动的上下文是根上下文，关于上下文是有一个体系的，包括层次性的子上下文。Web容器中启动Spring应用程序时，首先就是建立根上下文，一个完整的上下文体系是通过ContextLoader来完成的。
2. 关于ContextLoaderListener，它是spring提供的类，实现的是ServletContextListener这个接口，接口在Servlet API中定义，



在ContextLoader中，完成了两个IOC容器的建立的过程，一个是在Web容器的建立起双亲IOC容器，另一个是生成相应的WebApplicationContext并将其初始化。

(在jar包中找到相应的类的结构)

<servlet>

<servlet-name>SpringMVC</servlet-name>

<servlet-class>

org.springframework.web.servlet.DispatcherServlet

</servlet-class>

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>

classpath\*:mvc-\*.xml

</param-value>

</init-param>

</servlet>

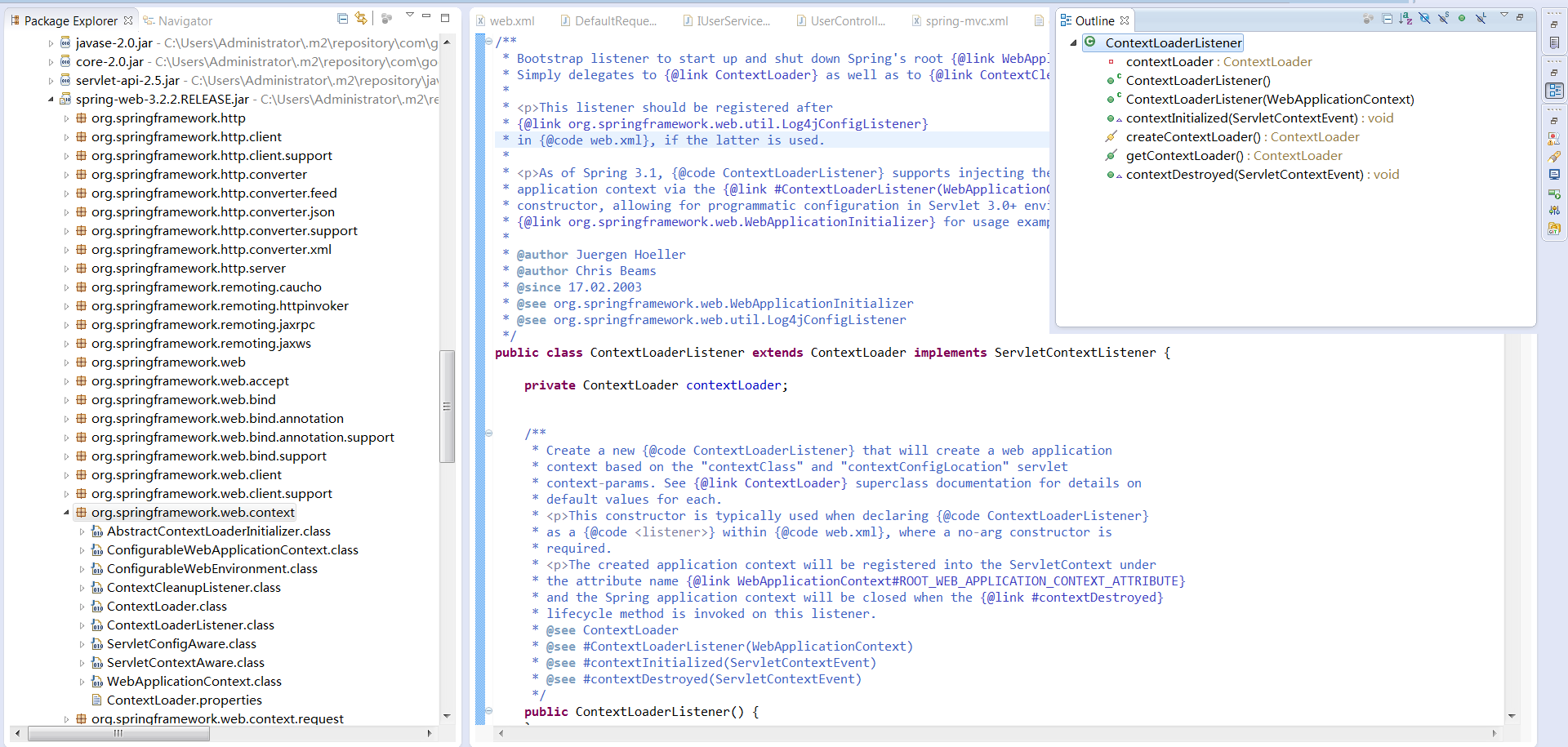
<listener>

<listener-class>

org.springframework.web.context.ContextLoaderListener

</listener-class>

</listener>



关于ContextLoader

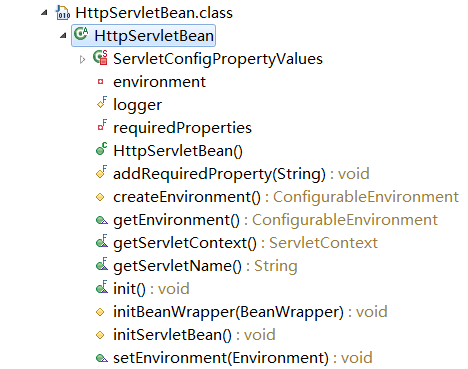
ContextLoaderListener是整个springweb加载IOC容器的第一步，从Servlet中获取ServletContext，读取web.xml中的配置，接着由ContextLoader实例化并初始化一个WebApplicationContext，既根上下文。然后这个根上下文又被保存到ServletContext中。

DispatcherServlet会根据根上下文创建一个子上下文然后存放到ServletContext中，

DispatcherServlet的工作主要是分成两个：初始化工作(如handlerMapping,ViewResolver等)，另一个是对Http请求进行响应。

DispatcherServlet的初始化:

读取配置在ServletContext中的Bean属性参数，接着会执行DispatcherServlet持有的IOC容器的初始化过程，其持有的上下文会被作为子上下文存起来(是一个servlet上下文)，根上下文是web容器的上下文，一个web应用中可以有多个Servlet存在。



public abstract class FrameworkServlet extends HttpServletBean {}

public class DispatcherServlet extends FrameworkServlet {}

@Override

**public** **final** **void** init() **throws** ServletException {

**if** (logger.isDebugEnabled()) {

logger.debug("Initializing servlet '" + getServletName() + "'");

}

// Set bean properties from init parameters.

**try** {

PropertyValues pvs = **new** ServletConfigPropertyValues(getServletConfig(), **this**.requiredProperties);

BeanWrapper bw = PropertyAccessorFactory.*forBeanPropertyAccess*(**this**);

ResourceLoader resourceLoader = **new** ServletContextResourceLoader(getServletContext());

bw.registerCustomEditor(Resource.**class**, **new** ResourceEditor(resourceLoader, getEnvironment()));

initBeanWrapper(bw);

bw.setPropertyValues(pvs, **true**);

}

**catch** (BeansException ex) {

logger.error("Failed to set bean properties on servlet '" + getServletName() + "'", ex);

**throw** ex;

}

// Let subclasses do whatever initialization they like.

initServletBean();

**if** (logger.isDebugEnabled()) {

logger.debug("Servlet '" + getServletName() + "' configured successfully");

}

}

接着是FrameworkServlet中对initServletBean();的实现

@Override

**protected** **final** **void** initServletBean() **throws** ServletException {

getServletContext().log("Initializing Spring FrameworkServlet '" + getServletName() + "'");

**if** (**this**.logger.isInfoEnabled()) {

**this**.logger.info("FrameworkServlet '" + getServletName() + "': initialization started");

}

**long** startTime = System.*currentTimeMillis*();

**try** {

**this**.webApplicationContext = initWebApplicationContext();//这一步中最终会调用到DispatcherServlet中的onRefresh

initFrameworkServlet();

}

**catch** (ServletException ex) {

**this**.logger.error("Context initialization failed", ex);

**throw** ex;

}

**catch** (RuntimeException ex) {

**this**.logger.error("Context initialization failed", ex);

**throw** ex;

}

**if** (**this**.logger.isInfoEnabled()) {

**long** elapsedTime = System.*currentTimeMillis*() - startTime;

**this**.logger.info("FrameworkServlet '" + getServletName() + "': initialization completed in " +

elapsedTime + " ms");

}

}

但是最终都是为了实现这样一段代码：

/\*\*

\* Initialize the strategy objects that this servlet uses.

\* <p>May be overridden in subclasses in order to initialize further strategy objects.

\*/

**protected** **void** initStrategies(ApplicationContext context) {

initMultipartResolver(context);

initLocaleResolver(context);

initThemeResolver(context);

initHandlerMappings(context);

initHandlerAdapters(context);

initHandlerExceptionResolvers(context);

initRequestToViewNameTranslator(context);

initViewResolvers(context);

initFlashMapManager(context);

}

然后就ok了，至此就可以处理http的请求了(分发)；

接下来的问题就是，HandlerMapping的配置和设计原理。当然这是后话。以上。

参考：计文柯先生的spring技术内幕这本书，

另，源码读起来也并不是特别的枯燥。