Spring MVC启动流程

1.初始化应用部署描述文件中每一个listener。

2.初始化ServletContextListener实现类，调用contextInitialized（）方法。

3.初始化应用部署描述文件中每一个filter，并执行每一个的init（）方法。

4.按照顺序<load-on-startup>来初始化servlet，并执行init（）方法。

先初始化lisener，再filter，最后servlet

```
常见SpringMVC配置：
<web-app>
  <display-name>Web Application</display-name>
  <!--全局变量配置-->
  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath:applicationContext-*.xml</param-value>
  </context-param>
  <!--监听器-->
  <listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
  </listener>
  <!--解决乱码问题的filter-->
  <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>
  <!--Restful前端控制器-->
  <servlet>
    <servlet-name>mvc</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>classpath:spring-mvc.xml</param-value>
    </init-param>
  </servlet>
  <servlet-mapping>
    <servlet-name>mvc</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>
</web-app>
```

DispatcherServlet的工作分为2部分：

一部分是初始化（也就是图的上半部分）：
由initServletBean()启动，通过initWebApplicationContext()方法最终调用DispatcherServlet中的initStrategies()方法

另一部分(也就是图的下半部分)：
对HTTP请求进行响应，作为Servlet，Web容器会调用Servlet的doGet()和doPost()方法,在经过FrameworkServlet的processRequest()简单处理后，会调用DispatcherServlet的doService方法，在这个方法调用中封装了doDispatch(),继续调用processDispatchResult方法返回调用信息。

HttpServletBean继承HttpServlet，因此在Web容器启动时将调用它的init方法，该初始化方法的主要作用

：：：将Servlet初始化参数（init-param）设置到该组件上（如contextAttribute、contextClass、namespace、contextConfigLocation），通过BeanWrapper简化设值过程，方便后续使用；
：：：提供给子类初始化扩展点，initServletBean()，该方法由FrameworkServlet覆盖。

```
public abstract class HttpServletBean extends HttpServlet implements EnvironmentAware{
@Override
    public final void init() throws ServletException {
        //省略部分代码
        //1、如下代码的作用是将Servlet初始化参数设置到该组件上
        //如contextAttribute、contextClass、namespace、contextConfigLocation;
        try {
            PropertyValues pvs = new ServletConfigPropertyValues(getServletConfig(),
            BeanWrapper bw = PropertyAccessorFactory.forBeanPropertyAccess(this);
            ResourceLoader resourceLoader = new ServletContextResourceLoader(getServletContext());
            bw.registerCustomEditor(Resource.class, new ResourceEditor(resourceLoader, this.environment));
            initBeanWrapper(bw);
            bw.setPropertyValues(pvs, true);
        }
        catch (BeansException ex) {
            //.....省略其他代码
        }
        //2、提供给子类初始化的扩展点，该方法由FrameworkServlet覆盖
        initServletBean();
        if (logger.isDebugEnabled()) {
            logger.debug("Servlet '" + getServletName() + "' configured successfully");
        }
    }
    //.....省略其他代码
}
```

FrameworkServlet继承HttpServletBean，通过initServletBean()进行Web上下文初始化，该方法主要覆盖一下两件事情：
1.初始化Web上下文；
2.提供给子类初始化扩展点

```
public abstract class FrameworkServlet extends HttpServletBean {
@Override
    protected final void initServletBean() throws ServletException {
        //省略部分代码
        try {
            //1、初始化Web上下文
            this.webApplicationContext = initWebApplicationContext();
            //2、提供给子类初始化的扩展点
            initFrameworkServlet();
        }
        //省略部分代码
    }
}
```

FrameworkServlet通过initServletBean进行web上下文初始化，调用initWebApplicationContext给子类初始化过程中所作的事情：

1：创建servlet注入的上下文
2：查找已经绑定的上下文
3：如果没有找到相应的上下文，则制定父类为ContextLoaderListener
4、刷新上下文（执行一些初始化）

```
protected WebApplicationContext initWebApplicationContext() {
        //ROOT上下文（ContextLoaderListener加载的）
        WebApplicationContext rootContext =
                WebApplicationContextUtils.getWebApplicationContext(getServletContext());
        WebApplicationContext wac = null;
        if (this.webApplicationContext != null) {
            // 1、在创建该Servlet注入的上下文
            wac = this.webApplicationContext;
            if (wac instanceof ConfigurableWebApplicationContext) {
                ConfigurableWebApplicationContext cwac = (ConfigurableWebApplicationContext) wac;
                if (!cwac.isActive()) {
                    if (cwac.getParent() == null) {
                        cwac.setParent(rootContext);
                    }
                    configureAndRefreshWebApplicationContext(cwac);
                }
            }
        }
        if (wac == null) {
            //2、查找已经绑定的上下文
            wac = findWebApplicationContext();
        }
        if (wac == null) {
            //3、如果没有找到相应的上下文，并指定父类为ContextLoaderListener
            wac = createWebApplicationContext(rootContext);
        }
        if (!this.refreshEventReceived) {
            //4、刷新上下文（执行一些初始化）
            onRefresh(wac);
        }
        if (this.publishContext) {
            // Publish the context as a servlet context attribute.
            String attrName = getServletContextAttributeName();
            getServletContext().setAttribute(attrName, wac);
            //省略部分代码
        }
        return wac;
    }
```

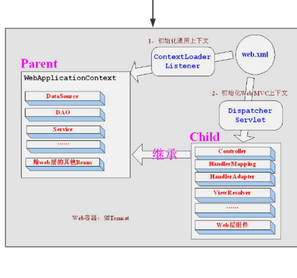
DispatcherServlet继承FrameworkServlet，并实现了onRefresh()方法提供一些前端控制器相关的配置

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初始化的过程和做了些什么事情，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。

```
public class DispatcherServlet extends FrameworkServlet {
    //实现子类的onRefresh()方法，该方法委托为initStrategies()方法。
    @Override
    protected void onRefresh(ApplicationContext context) {
        initStrategies(context);
    }

    //初始化默认的Spring Web MVC框架使用的策略（如HandlerMapping）
    protected void initStrategies(ApplicationContext context) {
        initMultipartResolver(context);
        initLocaleResolver(context);
        initThemeResolver(context);
        initHandlerMappings(context);
        initHandlerAdapters(context);
        initHandlerExceptionResolvers(context);
        initRequestToViewNameTranslator(context);
        initViewResolvers(context);
        initFlashMapManager(context);
    }
```



Spring Web MVC 主要有哪些特殊的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3。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.ThemeResolver：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