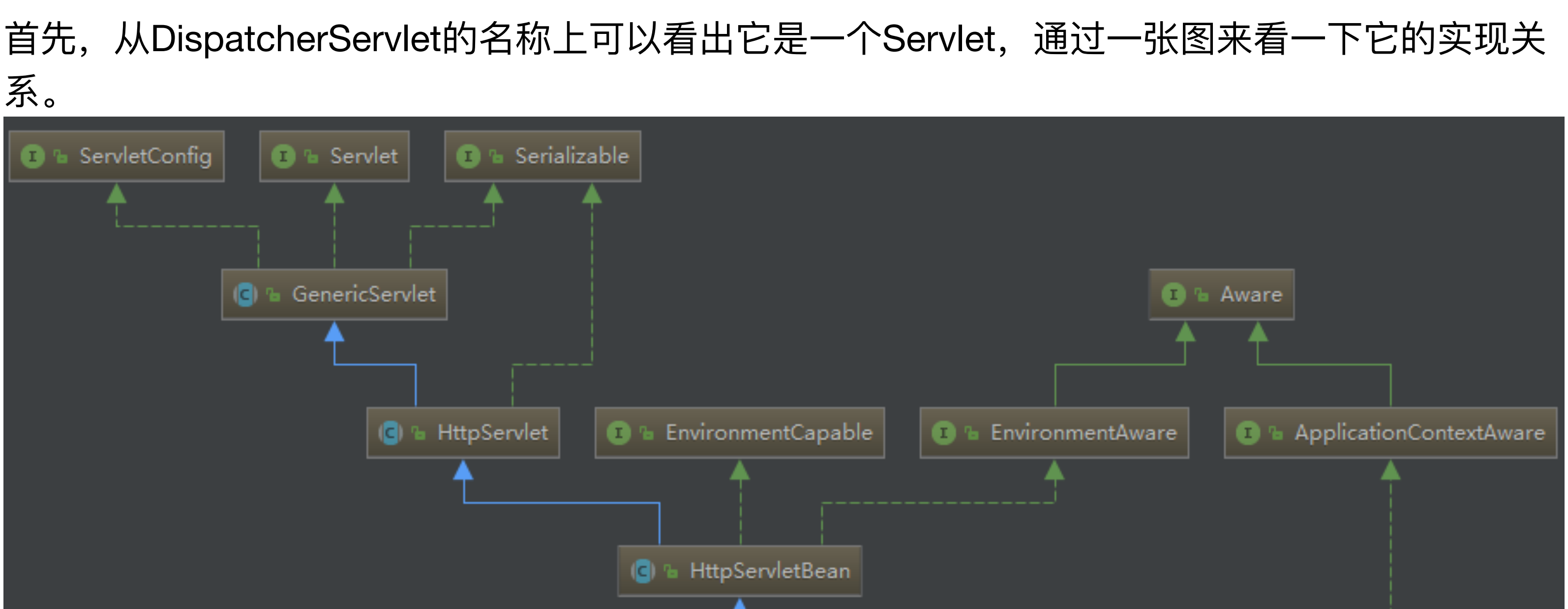
DispatcherServlet是SpringMVC的核心分发器，它实现了请求分发，是处理请求的入口，本篇将深入源码分析它的初始化过程。

首先，从DispatcherServlet的名称上可以看出它是一个Servlet，通过一张图来看一下它的实现关系。



既然DispatcherServlet是一个Servlet，那么初始化的时候一定会执行init方法，查看源码发现DispatcherServlet的init方法继承自HttpServletBean，具体代码如下图所示。

```
public final void init() throws ServletException {
    if (logger.isDebugEnabled()) {
        logger.debug("Initializing servlet '" + getServletName() + "'");
    }

    // Set bean properties from init parameters.
    try {从web.xml中获取初始化参数构建一个ServletConfigPropertyValues实例
        PropertyValues pvs = new ServletConfigPropertyValues(getServletConfig(), this.requiredProperties);
        BeanWrapper bw = PropertyAccessorFactory.forBeanPropertyAccess( target: this);
        ResourceLoader resourceLoader = new ServletContextResourceLoader(getServletContext());
        bw.registerCustomEditor(Resource.class, new ResourceEditor(resourceLoader, getEnvironment()));
        initBeanWrapper(bw);  初始化BeanWrapper，默认是空实现
        bw.setPropertyValues(pvs,  ignoreUnknown: true);      将DispatcherServlet构造成BeanWrapper
    }  将构造的属性实例设置到BeanWrapper中
    catch (BeansException ex) {
        logger.error( message: "Failed to set bean properties on servlet '" + getServletName() + "'", ex);
        throw ex;
    }

    // Let subclasses do whatever initialization they like.
    initServletBean();  默认是空实现，子类可重写，实现任何初始化操作

    if (logger.isDebugEnabled()) {
        logger.debug("Servlet '" + getServletName() + "' configured successfully");
    }
}
```

对于上面的代码，配合一个实际的web.xml配置样例更容易理解，具体如下图所示。

```
<servlet>
    <servlet-name>dispatcherServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath*:spring/mvc-servlet.xml</param-value>
    </init-param>
    <init-param>
        <param-name>encoding</param-name>
        <param-value>UTF-8</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>dispatcherServlet</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
```

上面示例配置中，配置了一个 contextConfigLocation参数，该参数用于构造SpringMVC上下文。对于initServletBean方法，FrameworkServlet中进行了重写，具体代码如下图所示。

```
protected final void initServletBean() throws ServletException {
    getServletContext().log( msg: "Initializing Spring FrameworkServlet '" + getServletName() + "'");
    if (this.logger.isInfoEnabled()) {
        this.logger.info("FrameworkServlet '" + getServletName() + "': initialization started");
    }
    long startTime = System.currentTimeMillis();

    try {  初始化WebApplicationContext，让Servlet和Spring容器关联
        this.webApplicationContext = initWebApplicationContext();
        initFrameworkServlet();  额外的初始化操作，默认是空实现，子类可重写
    }
    catch (ServletException ex) {
        this.logger.error( message: "Context initialization failed", ex);
        throw ex;
    }
    catch (RuntimeException ex) {
        this.logger.error( message: "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");
    }
}
```

initWebApplicationContext方法主要代码如下图所示。

```
protected WebApplicationContext initWebApplicationContext() {
    WebApplicationContext rootContext =
            WebApplicationContextUtils.getWebApplicationContext(getServletContext());
    WebApplicationContext wac = null;
  在ServletContext中查找WebApplicationContext,如果配置了ContextLoaderListener和
  contextConfigLocation,则rootContext代表这个监听器创建的上下文
    if (this.webApplicationContext != null) {
        // A context instance was injected at construction time -> use it
        wac = this.webApplicationContext;
        if (wac instanceof ConfigurableWebApplicationContext) {
            ConfigurableWebApplicationContext cwac = (ConfigurableWebApplicationContext) wac;
            if (!cwac.isActive()) {
                //...
                if (cwac.getParent() == null) {
                    //...
                    cwac.setParent(rootContext);
                }
                configureAndRefreshWebApplicationContext(cwac);
            }  DispatcherServlet有一个使用WebApplicationContext进行创建的构造方法，
        }      如果使用这个构造方法，这里将执行，设置父上下文和刷新
    }
    if (wac == null) {
        //...
        wac = findWebApplicationContext();  再次到ServletContext中查找，找不到报异常
    }
    if (wac == null) {
        // No context instance is defined for this servlet -> create a local one
        wac = createWebApplicationContext(rootContext);  创建一个局部的上下文
    }

    if (!this.refreshEventReceived) {
        //...
        onRefresh(wac);  DispatcherServlet会重写，进行一些初始化操作
    }

    if (this.publishContext) {
        // Publish the context as a servlet context attribute.
        String attrName = getServletContextAttributeName();
        getServletContext().setAttribute(attrName, wac);  将上下文放到ServletContext中
        if (this.logger.isDebugEnabled()) {
            this.logger.debug("Published WebApplicationContext of servlet '" + getServletName() +
                    "' as ServletContext attribute with name [" + attrName + "]");
        }
```

如果web.xml中配置了如下参数，将会在应用启动的时候初始化一个WebApplicationContext实例，并将其保存在ServletContext中。

```
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath*:application/applicationContext*.xml</param-value>
</context-param>

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

上图中的代码很容易理解，接下来进入DispatcherServlet的onRefresh方法，具体代码如下图所示。

```
protected void onRefresh(ApplicationContext context) {
    initStrategies(context);初始化各种策略
}

/**...*/
protected void initStrategies(ApplicationContext context) {
    initMultipartResolver(context);
    initLocaleResolver(context);
    initThemeResolver(context);
    initHandlerMappings(context);
    initHandlerAdapters(context);
    initHandlerExceptionResolvers(context);
    initRequestToViewNameTranslator(context);
    initViewResolvers(context);
    initFlashMapManager(context);
}
```

initStrategies方法中的各个方法，从方法名上可以很容易理解其作用，大部分的执行逻辑都是先从WebApplicationContext中查找，找不到的情况下再加载和DispatcherServlet同目录下的DispatcherServlet.properties中的各个策略，如初始化HandlerMapping，具体如下图所示。

```
private void initHandlerMappings(ApplicationContext context) {
    this.handlerMappings = null;

    if (this.detectAllHandlerMappings) {
        // Find all HandlerMappings in the ApplicationContext, including ancestor contexts.
        Map<String, HandlerMapping> matchingBeans =
                BeanFactoryUtils.beansOfTypeIncludingAncestors(context, HandlerMapping.class,  includeNonSingletons: true,  allowEagerInit: false);
        if (!matchingBeans.isEmpty()) {
            this.handlerMappings = new ArrayList<HandlerMapping>(matchingBeans.values());
            // We keep HandlerMappings in sorted order.
            OrderComparator.sort(this.handlerMappings);
        }
    }
    else {
        try {
            HandlerMapping hm = context.getBean(HANDLER_MAPPING_BEAN_NAME, HandlerMapping.class);
            this.handlerMappings = Collections.singletonList(hm);
        }
        catch (NoSuchBeanDefinitionException ex) {
            // Ignore, we'll add a default HandlerMapping later.
        }
    }

    // Ensure we have at least one HandlerMapping, by registering
    // a default HandlerMapping if no other mappings are found.
    if (this.handlerMappings == null) {
        this.handlerMappings = getDefaultStrategies(context, HandlerMapping.class);
        if (logger.isDebugEnabled()) {
            logger.debug("No HandlerMappings found in servlet '" + getServletName() + "': using default");
        }
    }  加载同目录下DispatcherServlet.properties中的HandlerMapping
}
```

```
META-INF
org.springframework.web.servlet
  config
  handler
  i18n
  mvc
  resource
  support
  tags
  theme
  view
  AsyncHandlerInterceptor
  DispatcherServlet
  DispatcherServlet.properties
  FlashMap
  FlashMapManager
  FrameworkServlet
  HandlerAdapter
  HandlerExceptionResolver
```

```
org.springframework.web.servlet.LocaleResolver=org.springframework.web.servlet.i18n.AcceptHeaderLocaleResolver

org.springframework.web.servlet.ThemeResolver=org.springframework.web.servlet.theme.FixedThemeResolver

org.springframework.web.servlet.HandlerMapping=org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping,\
    org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping

org.springframework.web.servlet.HandlerAdapter=org.springframework.web.servlet.mvc.HttpRequestHandlerAdapter,\
    org.springframework.web.servlet.mvc.SimpleControllerHandlerAdapter,\
    org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter

org.springframework.web.servlet.HandlerExceptionResolver=org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerExceptionResolver,\
    org.springframework.web.servlet.mvc.annotation.ResponseStatusExceptionResolver,\
    org.springframework.web.servlet.mvc.support.DefaultHandlerExceptionResolver

org.springframework.web.servlet.RequestToViewNameTranslator=org.springframework.web.servlet.view.DefaultRequestToViewNameTranslator

org.springframework.web.servlet.ViewResolver=org.springframework.web.servlet.view.InternalResourceViewResolver

org.springframework.web.servlet.FlashMapManager=org.springframework.web.servlet.support.SessionFlashMapManager
```