



[回到首页](#)

## 芋道源码 —— 知识星球

我是一段不羁的公告！

记得给芬芳这 3 个项目加油，添加一个 STAR 噢。

<https://github.com/YunaiV/SpringBoot-Labs>

<https://github.com/YunaiV/onemall>

<https://github.com/YunaiV/ruoyi-vue-pro>

[2020-02-18](#)

[Spring MVC](#)

## 精尽 Spring MVC 源码分析 —— 组件一览

### 1. 概述

在 [《精尽 Spring MVC 源码分析 —— 容器的初始化（二）之 Servlet WebApplicationContext 容器》](#) 一文中，我们看到，会调用 `DispatcherServlet` 的 `#initStrategies(ApplicationContext context)` 方法，初始化 Spring MVC 的各种组件。代码如下：

```
// DispatcherServlet.java

/** MultipartResolver used by this servlet. */
@Nullable
private MultipartResolver multipartResolver;

/** LocaleResolver used by this servlet. */
@Nullable
private LocaleResolver localeResolver;

/** ThemeResolver used by this servlet. */
@Nullable
private ThemeResolver themeResolver;

/** List of HandlerMappings used by this servlet. */
@Nullable
private List<HandlerMapping> handlerMappings;

/** List of HandlerAdapters used by this servlet. */
@Nullable
private List<HandlerAdapter> handlerAdapters;

/** List of HandlerExceptionResolvers used by this servlet. */
@Nullable
private List<HandlerExceptionResolver> handlerExceptionResolvers;

/** RequestToViewNameTranslator used by this servlet. */
@Nullable
private RequestToViewNameTranslator viewNameTranslator;

/** FlashMapManager used by this servlet. */
@Nullable
```

```

private FlashMapManager flashMapManager;

/** List of ViewResolvers used by this servlet. */
@Nullable
private List<ViewResolver> viewResolvers;

/**
 * This implementation calls {@link #initStrategies}.
 */
@Override
protected void onRefresh(ApplicationContext context) {
    initStrategies(context);
}

/**
 * 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) {
    // 初始化 MultipartResolver
    initMultipartResolver(context);
    // 初始化 LocaleResolver
    initLocaleResolver(context);
    // 初始化 ThemeResolver
    initThemeResolver(context);
    // 初始化 HandlerMappings
    initHandlerMappings(context);
    // 初始化 HandlerAdapters
    initHandlerAdapters(context);
    // 初始化 HandlerExceptionResolvers
    initHandlerExceptionResolvers(context);
    // 初始化 RequestToViewNameTranslator
    initRequestToViewNameTranslator(context);
    // 初始化 ViewResolvers
    initViewResolvers(context);
    // 初始化 FlashMapManager
    initFlashMapManager(context);
}

```

掐指一算，一共有 9 个组件。本文，我们对这 9 个组件，做一个简单的介绍。

当然，在具体介绍组件之前，我们还是通过一张图，来看看这些组件，在一次用户的请求中，扮演了什么样的角色。

FROM [《Spring MVC 原理探秘 —— 一个请求的旅行过程》](#)



当然，这个图并没有包括所有的组件，主要涉及最核心的组件。

## 2. MultipartResolver

`org.springframework.web.multipart.MultipartResolver`，内容类型( `Content-Type` )为 `multipart/*` 的请求的解析器接口。

例如，文件上传请求，`MultipartResolver` 会将 `HttpServletRequest` 封装成 `MultipartHttpServletRequest`，这样从 `MultipartHttpServletRequest` 中获得上传的文件。具体的使用示例，参见 [《spring-boot 上传文件 MultiPartFile 获取不到文件问题解决》](#)

关于内容类型( `Content-Type` )为 `multipart/*`，胖友可以看看 [《HTTP 协议之 multipart/form-data 请求分析》](#) 文章。

`MultipartResolver` 接口，代码如下：

```
// MultipartResolver.java

public interface MultipartResolver {

    /**
     * 是否为 multipart 请求
     */
    boolean isMultipart(HttpServletRequest request);

    /**
     * 将 HttpServletRequest 请求封装成 MultipartHttpServletRequest 对象
     */
    MultipartHttpServletRequest resolveMultipart(HttpServletRequest request) throws MultipartException;

    /**
     * 清理处理 multipart 产生的资源，例如临时文件
     *
     */
    void cleanupMultipart(MultipartHttpServletRequest request);
}
```

## 3. LocaleResolver

`org.springframework.web.servlet.LocaleResolver`，本地化( 国际化 )解析器接口。代码如下：

```
// LocaleResolver.java

public interface LocaleResolver {

    /**
     * 从请求中，解析出要使用的语言。例如，请求头的 "Accept-Language"
     */
    Locale resolveLocale(HttpServletRequest request);

    /**
     * 设置请求所使用的语言
     */
}
```

```

        */
        void setLocale(HttpServletRequest request, @Nullable HttpServletResponse response, @Nullable Locale locale);
    }

```

具体的使用示例，参见 [《SpringMVC学习系列（8）之 国际化》](#)。

## 4. ThemeResolver

`org.springframework.web.servlet.ThemeResolver`，主题解析器接口。代码如下：

```

// ThemeResolver.java

public interface ThemeResolver {

    /**
     * 从请求中，解析出使用的主题。例如，从请求头 User-Agent，判断使用 PC 端，还是移动端的主题
     */
    String resolveThemeName(HttpServletRequest request);

    /**
     * 设置请求，所使用的主题。
     */
    void setThemeName(HttpServletRequest request, @Nullable HttpServletResponse response, @Nullable String themeName);
}

```

具体的使用示例，参见 [《如何使用 Spring MVC 主题》](#)。

当然，因为现在的前端，基本和后端做了分离，所以这个功能已经越来越少用了。

## 5. HandlerMapping

`org.springframework.web.servlet.HandlerMapping`，处理器匹配接口，根据请求（handler）获得其的处理器（handler）和拦截器们（HandlerInterceptor 数组）。代码如下：

```

// HandlerMapping.java

public interface HandlerMapping {

    String PATH_WITHIN_HANDLER_MAPPING_ATTRIBUTE = HandlerMapping.class.getName() + ".pathWithinHandlerMapping";
    String BEST_MATCHING_PATTERN_ATTRIBUTE = HandlerMapping.class.getName() + ".bestMatchingPattern";
    String INTROSPECT_TYPE_LEVEL_MAPPING = HandlerMapping.class.getName() + ".introspectTypeLevelMapping";
    String URI_TEMPLATE_VARIABLES_ATTRIBUTE = HandlerMapping.class.getName() + ".uriTemplateVariables";
    String MATRIX_VARIABLES_ATTRIBUTE = HandlerMapping.class.getName() + ".matrixVariables";
    String PRODUCIBLE_MEDIA_TYPES_ATTRIBUTE = HandlerMapping.class.getName() + ".producibleMediaTypes";

    /**
     * 获得请求对应的处理器和拦截器们
     */
    @Nullable

```

```

HandlerExecutionChain getHandler(HttpServletRequest request) throws Exception;

}

```

返回的对象类型是 `HandlerExecutionChain`，它包含处理器（`handler`）和拦截器们（`HandlerInterceptor` 数组）。简单代码如下：

```

// HandlerExecutionChain.java

/**
 * 处理器
 */
private final Object handler;
/**
 * 拦截器数组
 */
@Nullable
private HandlerInterceptor[] interceptors;

```

- 注意，处理器的类型可能和我们想的不太一样，是个 `Object` 类型。

## 6. HandlerAdapter

`org.springframework.web.servlet.HandlerAdapter`，处理器适配器接口。代码如下：

```

// HandlerAdapter.java

public interface HandlerAdapter {

    /**
     * 是否支持该处理器
     */
    boolean supports(Object handler);

    /**
     * 执行处理器，返回 ModelAndView 结果
     */
    @Nullable
    ModelAndView handle(HttpServletRequest request, HttpServletResponse response, Object handler) throws Exception;

    /**
     * 返回请求的最新更新时间。
     *
     * 如果不支持该操作，则返回 -1 即可
     */
    long getLastModified(HttpServletRequest request, Object handler);

}

```

因为，处理器 `handler` 的类型是 `Object` 类型，需要有一个调用者来实现 `handler` 是怎么被使用，怎么被执行。而 `HandlerAdapter` 的用途就在于此。可能如果接口名改成 `HandlerInvoker`，笔者觉得会更好理解。三个接口，代码比较好理解，胖友瞅一眼，就不细讲了。

## 7. HandlerExceptionHandler

`org.springframework.web.servlet.HandlerExceptionHandler` ，处理器异常解析器接口，将处理器（handler）执行时发生的异常，解析（转换）成对应的 `ModelAndView` 结果。代码如下：

```
// HandlerExceptionHandler.java

public interface HandlerExceptionHandler {

    /**
     * 解析异常，转换成对应的 ModelAndView 结果
     */
    @Nullable
    ModelAndView resolveException(
        HttpServletRequest request, HttpServletResponse response, @Nullable Object handler, Exception ex);
}
```

## 8. RequestToViewNameTranslator

`org.springframework.web.servlet.RequestToViewNameTranslator` ，请求到视图名的转换器接口。代码如下：

```
// RequestToViewNameTranslator.java

public interface RequestToViewNameTranslator {

    /**
     * 根据请求，获得其视图名
     */
    @Nullable
    String getViewName(HttpServletRequest request) throws Exception;
}
```

粗略这么一看，有点不太好理解。捉摸了一下，还是放在后面一起讲解源码的时候，在详细讲解。

## 9. ViewResolver

`org.springframework.web.servlet.ViewResolver` ，实体解析器接口，根据视图名和国际化，获得最终的视图 `View` 对象。代码如下：

```
// ViewResolver.java

public interface ViewResolver {

    /**
     * 根据视图名和国际化，获得最终的 View 对象
     */
    @Nullable
```

```
View resolveViewName(String viewName, Locale locale) throws Exception;

}
```

ViewResolver 的实现类比较多，例如说，InternalResourceViewResolver 负责解析 JSP 视图，FreeMarkerViewResolver 负责解析 Freemarker 视图。当然，详细的，我们后续文章解析。

## 10. FlashMapManager

org.springframework.web.servlet.FlashMapManager，FlashMap 管理器接口，负责重定向时，保存参数到临时存储中。代码如下：

```
// FlashMapManager.java

public interface FlashMapManager {

    /**
     * 恢复参数，并将恢复过的和超时的参数从保存介质中删除
     */
    @Nullable
    FlashMap retrieveAndUpdate(HttpServletRequest request, HttpServletResponse response);

    /**
     * 将参数保存起来
     */
    void saveOutputFlashMap(FlashMap flashMap, HttpServletRequest request, HttpServletResponse response);

}
```

默认情况下，这个临时存储会是 Session。也就是说：

重定向前，保存参数到 Session 中。  
重定向后，从 Session 中获得参数，并移除。

具体使用示例，参见 [《Spring MVC Flash Attribute 的讲解与使用示例》](#) 一文。

当然，实际场景下，使用的非常少，特别是前后端分离之后。

## 666. 彩蛋

小小水文一篇，先一起了解下 Spring MVC 的组件。酱紫，我们下一篇好整体的了解一个用户的请求，DispatcherServlet 是如何使用上述的组件，对其进行处理。

参考和推荐如下文章：

田小波 [《Spring MVC 原理探秘 - 一个请求的旅行过程》](#)  
glmapper [《SpringMVC 源码系列：九大组件小记》](#)

郝佳 [《Spring 源码深度解析》](#) 的 [「11.3 DispatcherServlet」](#) 小节

韩路彪 [《看透 Spring MVC：源代码分析与实践》](#) 的 [「第9章 创建 Spring MVC 之器」](#) 小节

文章目录

1. [1. 1. 概述](#)
2. [2. 2. MultipartResolver](#)
3. [3. 3. LocaleResolver](#)
4. [4. 4. ThemeResolver](#)
5. [5. 5. HandlerMapping](#)
6. [6. 6. HandlerAdapter](#)
7. [7. 7. HandlerExceptionResolver](#)
8. [8. 8. RequestToViewNameTranslator](#)
9. [9. 9. ViewResolver](#)
10. [10. 10. FlashMapManager](#)
11. [11. 666. 彩蛋](#)

2014 - 2023 芋道源码 |  
总访客数 次 && 总访问量 次  
[回到首页](#)