06 SpringMVC-视图解析流程分析

`SpringMVC`

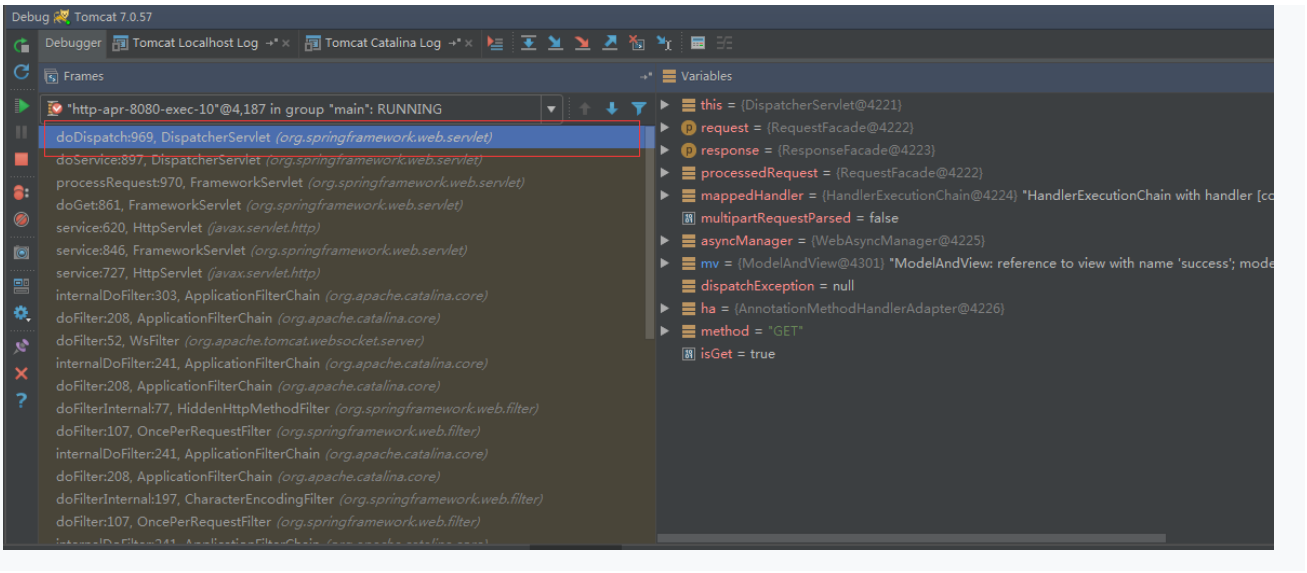
一：视图解析流程分析

> 1、请求处理方法执行完成后，最终返回一个 ModelAndView对象
>
> 2、对于那些返回 String，View 或 ModeMap 等类型的处理方法，Spring MVC 也会在内部将它们装配成一个ModelAndView 对象，它包含了逻辑名和模型对象的视图

```
//Controller层
    @RequestMapping("/testModelAndView1")
    public String testModelAndView1(){
        System.out.println("testModelAndView:");
        //调用用Serivce层
        /**
        打上断点，执行Dug程序
        */
        return "success";
    }
```

二：源码解析

> 1、执行Dug程序 ，并进入到DispatcherServlet类中doDispatch方法中




```
 protected void doDispatch(HttpServletRequest request, ......
 ){
 //其他代码省略.............
 /**
  在此处打DUG断点，获取ModelAndView对象
 */
 mv = ha.handle(processedRequest, response, mappedHandler.getHandler())
;
                    if (asyncManager.isConcurrentHandlingStarted()) {
                        return;
                    }
 }
  /**
  在此处打DUG断点，进行处理视图，点击查询怎么处理视图
 */
 this.applyDefaultViewName(processedRequest, mv);
                    mappedHandler.applyPostHandle(processedRequest, res
ponse, mv);
 //其他代码省略.............
/**
  在此处打DUG断点
 */
   this.processDispatchResult(processedRequest, response,
mappedHandler, mv, (Exception)dispatchException);
            } catch (Exception var22) {
     //其他代码省略.............
```

1、applyDefaultViewName

> 1、点击this.applyDefaultViewName(processedRequest, mv);
> 2、查询处理的视图

```
private void processDispatchResult(HttpServletRequest request,
...){
    //其他代码省略.....
if (mv != null && !mv.wasCleared()) {
/**
当前模型视图不为空，就进行渲染视图
在此处打DUG断点
，点击查询怎么处理视图
*/
            this.render(mv, request, response);
            if (errorView) {
}
   //其他代码省略.....
```

2、render

> 1、进行渲染视图

```
protected void render(ModelAndView mv,....
){
          //其他代码省略.....
    if (mv.isReference()) {
    /**
  解析视图的名称
在此处打DUG断点
，点击查询怎么解析视图
*/
            view = this.resolveViewName(mv.getViewName(), mv.getModelIn
ternal(), locale, request);
            if (view == null) {
        //其他代码省略.....
 try {
            if (mv.getStatus() != null) {
                response.setStatus(mv.getStatus().value());
            }
/**
  真正的解析渲染一个视图
```

```
在此处打DUG断点，点击查询怎么渲染的
*/
            view.render(mv.getModelInternal(), request, response);
        } catch
}
```

1、this.resolveViewName(mv.getViewName()

> 1、进行点击resolveViewName解析视图名称

```
protected View resolveViewName(String viewName, Map<String, Object>
model, Locale locale, HttpServletRequest request) throws Exception {
        Iterator var5 = this.viewResolvers.iterator();
        View view;
        do {
            if (!var5.hasNext()) {
                return null;
            }
            //通过视图解析器得到一个视图
            ViewResolver viewResolver = (ViewResolver)var5.next();
            view = viewResolver.resolveViewName(viewName, locale);
        } while(view == null);
//返回一个视图
        return view;
    }
```

2、 view.render(mv.getModelInternal(),

> 1、查询怎么真正的渲染一个视图
> 2、点击是一个抽象类，查询视图类(Ctrl+alt+B),在AbstractView类中

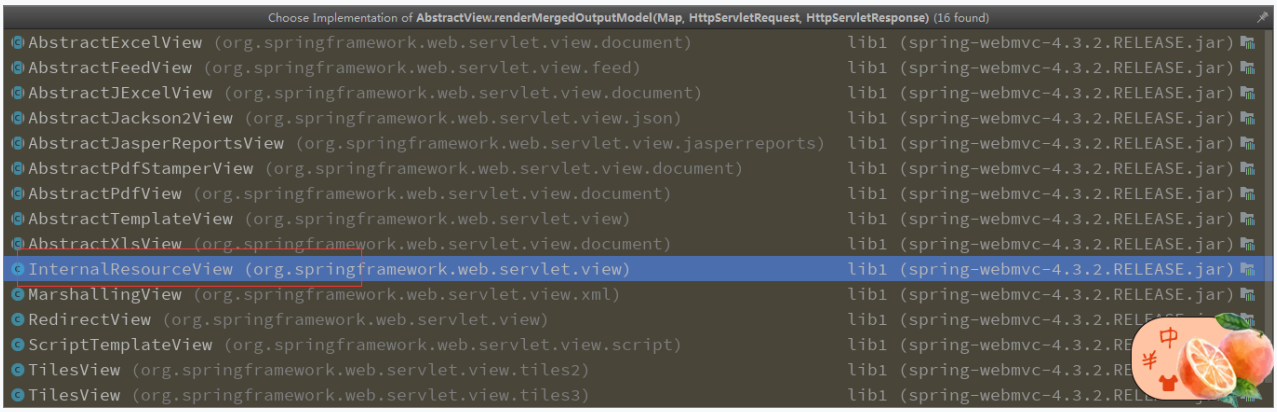
```
public void render(Map<String, ?> model, HttpServletRequest request, Ht
tpServletResponse response) throws Exception {

  //其他代码省略.....
        this.prepareResponse(request, response);
    /**
  进行合并视图模型
在此处打DUG断点，点击查询怎么合并视图模型
*/
        this.renderMergedOutputModel(mergedModel, this.getRequestToExpo
se(request), response);
```

```
10.         }
```

1、this.renderMergedOutputModel(mergedModel,

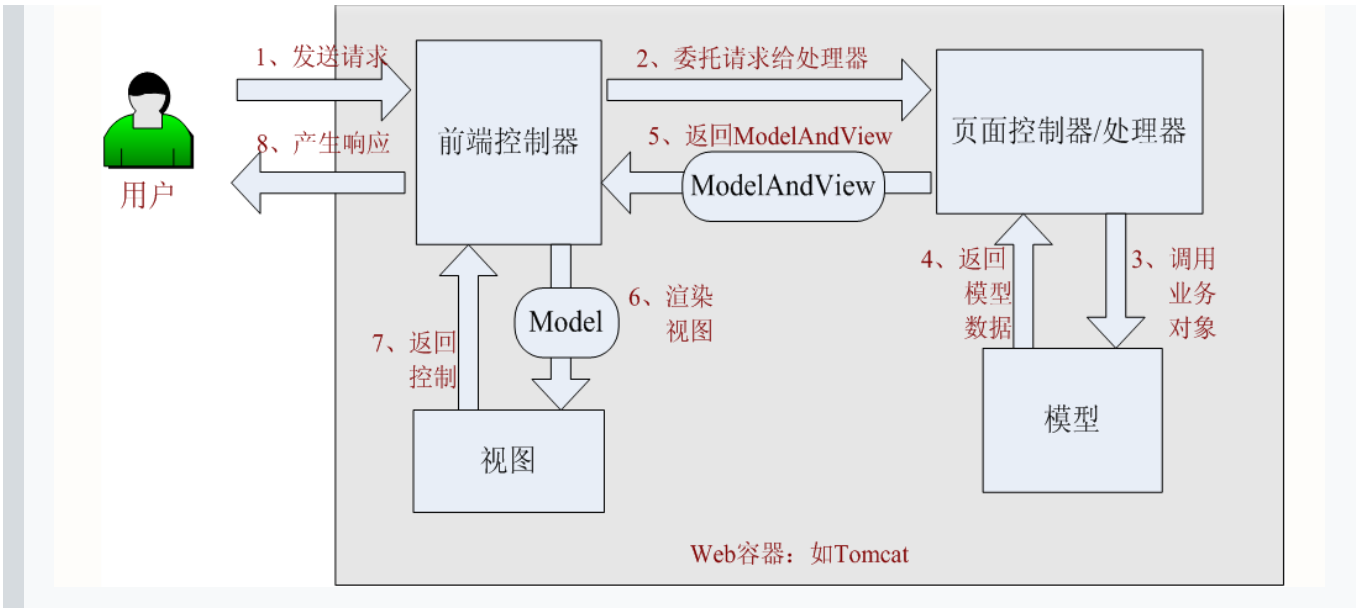
1、点击是一个抽象类，查询视图类(Ctrl+alt+B)，InternalResourceView实现类




```
1.  protected void renderMergedOutputModel(Map<String, Object> model, HttpS
    ervletRequest request, HttpServletResponse response) throws Exception
    {
2.          this.exposeModelAsRequestAttributes(model, request);
3.    //其他代码省略.............
4.       /**
5.    进行合并视图模型
6.  在此处打DUG断点，点击查询怎么合并视图模型
7.  */
8.      rd.forward(request, response);
```

2、Spring MVC 借助视图解析器（ViewResolver）得到最终的视图对象（View），最终的视图可以是 JSP ，也可能是 html、JFreeChart 等各种表现形式的视图

3、模型图


用户
1、发送请求
8、产生响应
前端控制器
2、委托请求给处理器
5、返回ModelAndView
ModelAndView
页面控制器/处理器
4、返回
模型
数据
3、调用
业务
对象
Model
6、渲染
视图
7、返回
控制
视图
模型
Web容器：如Tomcat