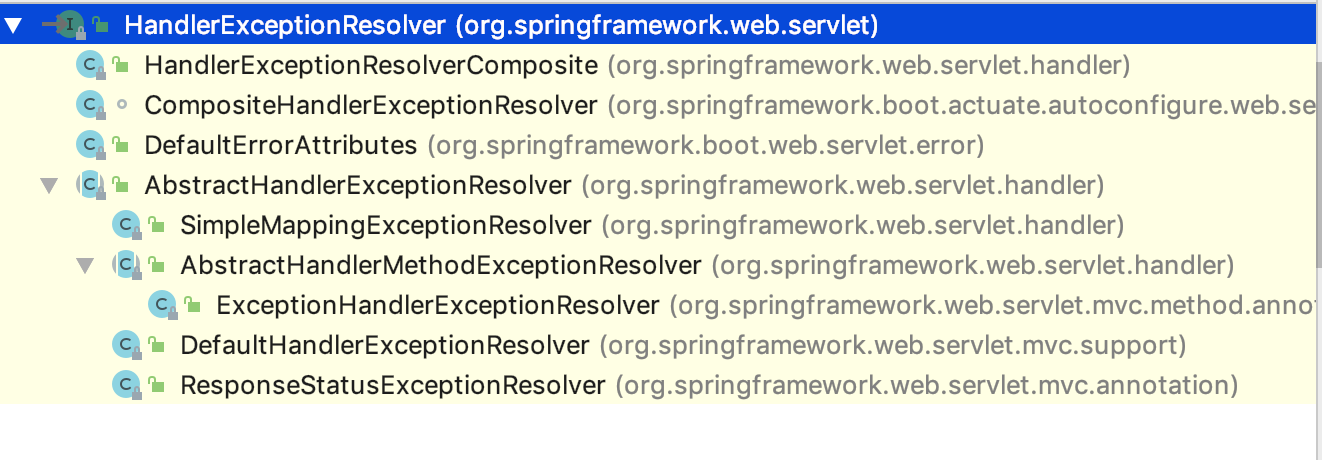
ControllerAdvise 全局异常处理流程

在DispatchServlet 1055行 processDispatchResult(processedRequest, response, mappedHandler, mv, dispatchException);

dispatchException 为业务抛出的异常,收到异常后就进行异常处理

异常解析器有如下这些：



异常解析器通过WebMvcConfigurationSupport自动注入，代码如下：

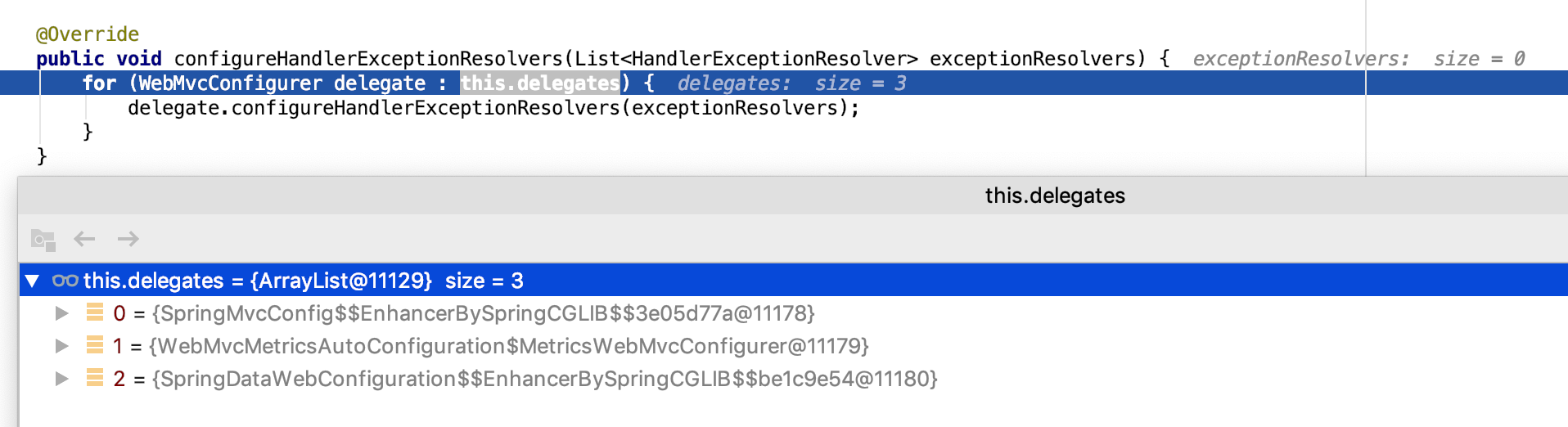
@Bean  
**public** HandlerExceptionResolver handlerExceptionResolver() {  
 List<HandlerExceptionResolver> exceptionResolvers = **new** ArrayList<>();  
 configureHandlerExceptionResolvers(exceptionResolvers);  
 **if** (exceptionResolvers.isEmpty()) {  
 addDefaultHandlerExceptionResolvers(exceptionResolvers);  
 }  
 extendHandlerExceptionResolvers(exceptionResolvers);  
 HandlerExceptionResolverComposite composite = **new** HandlerExceptionResolverComposite();  
 composite.setOrder(0);  
 composite.setExceptionResolvers(exceptionResolvers);  
 **return** composite;  
}

configureHandlerExceptionResolvers(exceptionResolvers);  
这行代码会去加载实现了WebMvcConfigurer

接口的实现类，并寻找异常解析器，当前配置类有如下3个：

第一个是我们自己配置的，第二个是springboot-actuator模块提供的

第三个是spring-data-common 模块提供的



addDefaultHandlerExceptionResolvers(exceptionResolvers);会去加载默认的，如果没有我们自己指定的异常解析器。

加载默认解析器如下：ExceptionHandlerExceptionResolver，ResponseStatusExceptionResolver，DefaultHandlerExceptionResolver

加载了这3个，并使用HandlerExceptionResolverComposite进行委托，也就是HandlerExceptionResolverComposite包含这三个解析器

**protected final void** addDefaultHandlerExceptionResolvers(List<HandlerExceptionResolver> exceptionResolvers) {  
 ExceptionHandlerExceptionResolver exceptionHandlerResolver = createExceptionHandlerExceptionResolver();  
 exceptionHandlerResolver.setContentNegotiationManager(mvcContentNegotiationManager());  
 exceptionHandlerResolver.setMessageConverters(getMessageConverters());  
 exceptionHandlerResolver.setCustomArgumentResolvers(getArgumentResolvers());  
 exceptionHandlerResolver.setCustomReturnValueHandlers(getReturnValueHandlers());  
 **if** (***jackson2Present***) {  
 exceptionHandlerResolver.setResponseBodyAdvice(  
 Collections.*singletonList*(**new** JsonViewResponseBodyAdvice()));  
 }  
 **if** (**this**.**applicationContext** != **null**) {  
 exceptionHandlerResolver.setApplicationContext(**this**.**applicationContext**);  
 }  
 exceptionHandlerResolver.afterPropertiesSet();  
 exceptionResolvers.add(exceptionHandlerResolver);  
  
 ResponseStatusExceptionResolver responseStatusResolver = **new** ResponseStatusExceptionResolver();  
 responseStatusResolver.setMessageSource(**this**.**applicationContext**);  
 exceptionResolvers.add(responseStatusResolver);  
  
 exceptionResolvers.add(**new** DefaultHandlerExceptionResolver());  
}

ExceptionHandlerExceptionResolver在创建时，还会去加载使用了@ControllerAdvice注解的类，并使用ExceptionHandlerMethodResolver，去匹配使用了@ExceptionHandler

注解的方法，然后将bean与resover绑定到一个map中，供后续反射调用方法时使用

processHandlerException解析异常如下：使用上述的异常解析器依次解析异常

ExceptionHandlerExceptionResolver（处理异常类型由业务指定），ResponseStatusExceptionResolver（处理ResponseStatusException异常），DefaultHandlerExceptionResolver处理如下异常：

**try** {  
 **if** (ex **instanceof** HttpRequestMethodNotSupportedException) {  
 **return** handleHttpRequestMethodNotSupported(  
 (HttpRequestMethodNotSupportedException) ex, request, response, handler);  
 }  
 **else if** (ex **instanceof** HttpMediaTypeNotSupportedException) {  
 **return** handleHttpMediaTypeNotSupported(  
 (HttpMediaTypeNotSupportedException) ex, request, response, handler);  
 }  
 **else if** (ex **instanceof** HttpMediaTypeNotAcceptableException) {  
 **return** handleHttpMediaTypeNotAcceptable(  
 (HttpMediaTypeNotAcceptableException) ex, request, response, handler);  
 }  
 **else if** (ex **instanceof** MissingPathVariableException) {  
 **return** handleMissingPathVariable(  
 (MissingPathVariableException) ex, request, response, handler);  
 }  
 **else if** (ex **instanceof** MissingServletRequestParameterException) {  
 **return** handleMissingServletRequestParameter(  
 (MissingServletRequestParameterException) ex, request, response, handler);  
 }  
 **else if** (ex **instanceof** ServletRequestBindingException) {  
 **return** handleServletRequestBindingException(  
 (ServletRequestBindingException) ex, request, response, handler);  
 }  
 **else if** (ex **instanceof** ConversionNotSupportedException) {  
 **return** handleConversionNotSupported(  
 (ConversionNotSupportedException) ex, request, response, handler);  
 }  
 **else if** (ex **instanceof** TypeMismatchException) {  
 **return** handleTypeMismatch(  
 (TypeMismatchException) ex, request, response, handler);  
 }  
 **else if** (ex **instanceof** HttpMessageNotReadableException) {  
 **return** handleHttpMessageNotReadable(  
 (HttpMessageNotReadableException) ex, request, response, handler);  
 }  
 **else if** (ex **instanceof** HttpMessageNotWritableException) {  
 **return** handleHttpMessageNotWritable(  
 (HttpMessageNotWritableException) ex, request, response, handler);  
 }  
 **else if** (ex **instanceof** MethodArgumentNotValidException) {  
 **return** handleMethodArgumentNotValidException(  
 (MethodArgumentNotValidException) ex, request, response, handler);  
 }  
 **else if** (ex **instanceof** MissingServletRequestPartException) {  
 **return** handleMissingServletRequestPartException(  
 (MissingServletRequestPartException) ex, request, response, handler);  
 }  
 **else if** (ex **instanceof** BindException) {  
 **return** handleBindException((BindException) ex, request, response, handler);  
 }  
 **else if** (ex **instanceof** NoHandlerFoundException) {  
 **return** handleNoHandlerFoundException(  
 (NoHandlerFoundException) ex, request, response, handler);  
 }  
 **else if** (ex **instanceof** AsyncRequestTimeoutException) {  
 **return** handleAsyncRequestTimeoutException(  
 (AsyncRequestTimeoutException) ex, request, response, handler);  
 }

使用@ExceptionHandler处理异常时，一样会使用参数解析器，返回值解析器处理一遍。