WebMvcConfigurationSupport 提供的能力

// 提供基于@RequestMapping注解的路径解析能力，提前加载了所有@RequestMapping

注解包含的所有路径

*/\*\*  
 \* Return a {****@link*** *RequestMappingHandlerMapping} ordered at 0 for mapping  
 \* requests to annotated controllers.  
 \*/*@Bean  
**public** RequestMappingHandlerMapping requestMappingHandlerMapping() {  
 RequestMappingHandlerMapping mapping = createRequestMappingHandlerMapping();  
 mapping.setOrder(0);  
 mapping.setInterceptors(getInterceptors());  
 mapping.setContentNegotiationManager(mvcContentNegotiationManager());  
 mapping.setCorsConfigurations(getCorsConfigurations());  
  
 PathMatchConfigurer configurer = getPathMatchConfigurer();  
  
 Boolean useSuffixPatternMatch = configurer.isUseSuffixPatternMatch();  
 **if** (useSuffixPatternMatch != **null**) {  
 mapping.setUseSuffixPatternMatch(useSuffixPatternMatch);  
 }  
 Boolean useRegisteredSuffixPatternMatch = configurer.isUseRegisteredSuffixPatternMatch();  
 **if** (useRegisteredSuffixPatternMatch != **null**) {  
 mapping.setUseRegisteredSuffixPatternMatch(useRegisteredSuffixPatternMatch);  
 }  
 Boolean useTrailingSlashMatch = configurer.isUseTrailingSlashMatch();  
 **if** (useTrailingSlashMatch != **null**) {  
 mapping.setUseTrailingSlashMatch(useTrailingSlashMatch);  
 }  
  
 UrlPathHelper pathHelper = configurer.getUrlPathHelper();  
 **if** (pathHelper != **null**) {  
 mapping.setUrlPathHelper(pathHelper);  
 }  
 PathMatcher pathMatcher = configurer.getPathMatcher();  
 **if** (pathMatcher != **null**) {  
 mapping.setPathMatcher(pathMatcher);  
 }  
 Map<String, Predicate<Class<?>>> pathPrefixes = configurer.getPathPrefixes();  
 **if** (pathPrefixes != **null**) {  
 mapping.setPathPrefixes(pathPrefixes);  
 }  
  
 **return** mapping;  
}

// 提供一个Ant风格的路径匹配器

效果

可以做URLs匹配，规则如下

？匹配一个字符

\*匹配0个或多个字符

\*\*匹配0个或多个目录

用例如下

/trip/api/\*x    匹配 /trip/api/x，/trip/api/ax，/trip/api/abx ；但不匹配 /trip/abc/x；

/trip/a/a?x    匹配 /trip/a/abx；但不匹配 /trip/a/ax，/trip/a/abcx

/\*\*/api/alie    匹配 /trip/api/alie，/trip/dax/api/alie；但不匹配 /trip/a/api

/\*\*/\*.htmlm   匹配所有以.htmlm结尾的路径

*/\*\*  
 \* Return a global {****@link*** *PathMatcher} instance for path matching  
 \* patterns in {****@link*** *HandlerMapping HandlerMappings}.  
 \* This instance can be configured using the {****@link*** *PathMatchConfigurer}  
 \* in {****@link*** *#configurePathMatch(PathMatchConfigurer)}.  
 \** ***@since*** *4.1  
 \*/*@Bean  
**public** PathMatcher mvcPathMatcher() {  
 PathMatcher pathMatcher = getPathMatchConfigurer().getPathMatcher();  
 **return** (pathMatcher != **null** ? pathMatcher : **new** AntPathMatcher());  
}

// url解析器，包括路径的编解码

*/\*\*  
 \* Return a global {****@link*** *UrlPathHelper} instance for path matching  
 \* patterns in {****@link*** *HandlerMapping HandlerMappings}.  
 \* This instance can be configured using the {****@link*** *PathMatchConfigurer}  
 \* in {****@link*** *#configurePathMatch(PathMatchConfigurer)}.  
 \** ***@since*** *4.1  
 \*/*@Bean  
**public** UrlPathHelper mvcUrlPathHelper() {  
 UrlPathHelper pathHelper = getPathMatchConfigurer().getUrlPathHelper();  
 **return** (pathHelper != **null** ? pathHelper : **new** UrlPathHelper());  
}

// 内容协商管理器

*/\*\*  
 \* Return a {****@link*** *ContentNegotiationManager} instance to use to determine  
 \* requested {****@linkplain*** *MediaType media types} in a given request.  
 \*/*@Bean  
**public** ContentNegotiationManager mvcContentNegotiationManager() {  
 **if** (**this**.**contentNegotiationManager** == **null**) {  
 ContentNegotiationConfigurer configurer = **new** ContentNegotiationConfigurer(**this**.**servletContext**);  
 configurer.mediaTypes(getDefaultMediaTypes());  
 configureContentNegotiation(configurer);  
 **this**.**contentNegotiationManager** = configurer.buildContentNegotiationManager();  
 }  
 **return this**.**contentNegotiationManager**;  
}

// 视图控制器注册中心，不需要我们而外写纯java代码的视图控制

// 实际上返回是一个SimpleUrlHandlerMapping

*/\*\*  
 \* Return a handler mapping ordered at 1 to map URL paths directly to  
 \* view names. To configure view controllers, override  
 \* {****@link*** *#addViewControllers}.  
 \*/*@Bean  
@Nullable  
**public** HandlerMapping viewControllerHandlerMapping() {  
 ViewControllerRegistry registry = **new** ViewControllerRegistry(**this**.**applicationContext**);  
 addViewControllers(registry);  
  
 AbstractHandlerMapping handlerMapping = registry.buildHandlerMapping();  
 **if** (handlerMapping == **null**) {  
 **return null**;  
 }  
 handlerMapping.setPathMatcher(mvcPathMatcher());  
 handlerMapping.setUrlPathHelper(mvcUrlPathHelper());  
 handlerMapping.setInterceptors(getInterceptors());  
 handlerMapping.setCorsConfigurations(getCorsConfigurations());  
 **return** handlerMapping;  
}

//早期controller映射处理方案，会加载所有bean名称以“/”开头的bean,

当我们请求该bean的名称的映射路径时，就由该bean处理请求，但是该bean还需要实现Controller接口或者Servlet接口，或者HttpRequestHandler接口

*/\*\*  
 \* Return a {****@link*** *BeanNameUrlHandlerMapping} ordered at 2 to map URL  
 \* paths to controller bean names.  
 \*/*@Bean  
**public** BeanNameUrlHandlerMapping beanNameHandlerMapping() {  
 BeanNameUrlHandlerMapping mapping = **new** BeanNameUrlHandlerMapping();  
 mapping.setOrder(2);  
 mapping.setInterceptors(getInterceptors());  
 mapping.setCorsConfigurations(getCorsConfigurations());  
 **return** mapping;  
}

// 返回一个SimpleUrlHandlerMapping，映射 /\*\* 路径，默认没有handler,handler的创建需要在WebMvcConfigurer实例类中重写configureDefaultServletHandling方法，并调用configurer.enable();

则会创建一个DefaultServletHttpRequestHandler

@Bean  
@Nullable  
**public** HandlerMapping defaultServletHandlerMapping() {  
 Assert.*state*(**this**.**servletContext** != **null**, **"No ServletContext set"**);  
 DefaultServletHandlerConfigurer configurer = **new** DefaultServletHandlerConfigurer(**this**.**servletContext**);  
 configureDefaultServletHandling(configurer);  
 **return** configurer.buildHandlerMapping();  
}

// 返回一个HandlerAdapter，处理HandlerMethod类型的handler

*/\*\*  
 \* Returns a {****@link*** *RequestMappingHandlerAdapter} for processing requests  
 \* through annotated controller methods. Consider overriding one of these  
 \* other more fine-grained methods:  
 \* <ul>  
 \* <li>{****@link*** *#addArgumentResolvers} for adding custom argument resolvers.  
 \* <li>{****@link*** *#addReturnValueHandlers} for adding custom return value handlers.  
 \* <li>{****@link*** *#configureMessageConverters} for adding custom message converters.  
 \* </ul>  
 \*/*@Bean  
**public** RequestMappingHandlerAdapter requestMappingHandlerAdapter() {  
 RequestMappingHandlerAdapter adapter = createRequestMappingHandlerAdapter();  
 adapter.setContentNegotiationManager(mvcContentNegotiationManager());  
 adapter.setMessageConverters(getMessageConverters());  
 adapter.setWebBindingInitializer(getConfigurableWebBindingInitializer());  
 adapter.setCustomArgumentResolvers(getArgumentResolvers());  
 adapter.setCustomReturnValueHandlers(getReturnValueHandlers());  
  
 **if** (***jackson2Present***) {  
 adapter.setRequestBodyAdvice(Collections.*singletonList*(**new** JsonViewRequestBodyAdvice()));  
 adapter.setResponseBodyAdvice(Collections.*singletonList*(**new** JsonViewResponseBodyAdvice()));  
 }  
  
 AsyncSupportConfigurer configurer = **new** AsyncSupportConfigurer();  
 configureAsyncSupport(configurer);  
 **if** (configurer.getTaskExecutor() != **null**) {  
 adapter.setTaskExecutor(configurer.getTaskExecutor());  
 }  
 **if** (configurer.getTimeout() != **null**) {  
 adapter.setAsyncRequestTimeout(configurer.getTimeout());  
 }  
 adapter.setCallableInterceptors(configurer.getCallableInterceptors());  
 adapter.setDeferredResultInterceptors(configurer.getDeferredResultInterceptors());  
  
 **return** adapter;  
}

// 返回一个HandlerAdater，用于处理HttpRequestHandler类型的handler

*/\*\*  
 \* Returns a {****@link*** *HttpRequestHandlerAdapter} for processing requests  
 \* with {****@link*** *HttpRequestHandler HttpRequestHandlers}.  
 \*/*@Bean  
**public** HttpRequestHandlerAdapter httpRequestHandlerAdapter() {  
 **return new** HttpRequestHandlerAdapter();  
}

// 返回一个HandlerAdapter，用于处理Controller类型的handler

*/\*\*  
 \* Returns a {****@link*** *SimpleControllerHandlerAdapter} for processing requests  
 \* with interface-based controllers.  
 \*/*@Bean  
**public** SimpleControllerHandlerAdapter simpleControllerHandlerAdapter() {  
 **return new** SimpleControllerHandlerAdapter();  
}

// 获取异常解析器，会扫描使用了@ControllerAdvice注解的类，并搜集使用了@ExceptionHandler注解的方法，还会搜集该类是否实现了RequestBodyAdvice，ResponseBodyAdvice

*/\*\*  
 \* Returns a {****@link*** *HandlerExceptionResolverComposite} containing a list of exception  
 \* resolvers obtained either through {****@link*** *#configureHandlerExceptionResolvers} or  
 \* through {****@link*** *#addDefaultHandlerExceptionResolvers}.  
 \* <p><strong>Note:</strong> This method cannot be made final due to CGLIB constraints.  
 \* Rather than overriding it, consider overriding {****@link*** *#configureHandlerExceptionResolvers}  
 \* which allows for providing a list of resolvers.  
 \*/*@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;  
}

// 搜集所有的视图解析器

*/\*\*  
 \* Register a {****@link*** *ViewResolverComposite} that contains a chain of view resolvers  
 \* to use for view resolution.  
 \* By default this resolver is ordered at 0 unless content negotiation view  
 \* resolution is used in which case the order is raised to  
 \* {****@link*** *org.springframework.core.Ordered#HIGHEST\_PRECEDENCE  
 \* Ordered.HIGHEST\_PRECEDENCE}.  
 \* <p>If no other resolvers are configured,  
 \* {****@link*** *ViewResolverComposite#resolveViewName(String, Locale)} returns null in order  
 \* to allow other potential {****@link*** *ViewResolver} beans to resolve views.  
 \** ***@since*** *4.1  
 \*/*@Bean  
**public** ViewResolver mvcViewResolver() {  
 ViewResolverRegistry registry = **new** ViewResolverRegistry(  
 mvcContentNegotiationManager(), **this**.**applicationContext**);  
 configureViewResolvers(registry);  
  
 **if** (registry.getViewResolvers().isEmpty() && **this**.**applicationContext** != **null**) {  
 String[] names = BeanFactoryUtils.*beanNamesForTypeIncludingAncestors*(  
 **this**.**applicationContext**, ViewResolver.**class**, **true**, **false**);  
 **if** (names.**length** == 1) {  
 registry.getViewResolvers().add(**new** InternalResourceViewResolver());  
 }  
 }  
  
 ViewResolverComposite composite = **new** ViewResolverComposite();  
 composite.setOrder(registry.getOrder());  
 composite.setViewResolvers(registry.getViewResolvers());  
 **if** (**this**.**applicationContext** != **null**) {  
 composite.setApplicationContext(**this**.**applicationContext**);  
 }  
 **if** (**this**.**servletContext** != **null**) {  
 composite.setServletContext(**this**.**servletContext**);  
 }  
 **return** composite;  
}