# 1 springboot 项目配置文件加载位置:

1. file:/config/ 在IDEA创建的项目中在项目的根目录下的config文件夹下
2. file:/
3. classpath:/config/ 在IDEA创建的项目中在项目的resource下的config文件夹下
4. classpath:/

不同位置优先级由高到低，相同位置互补

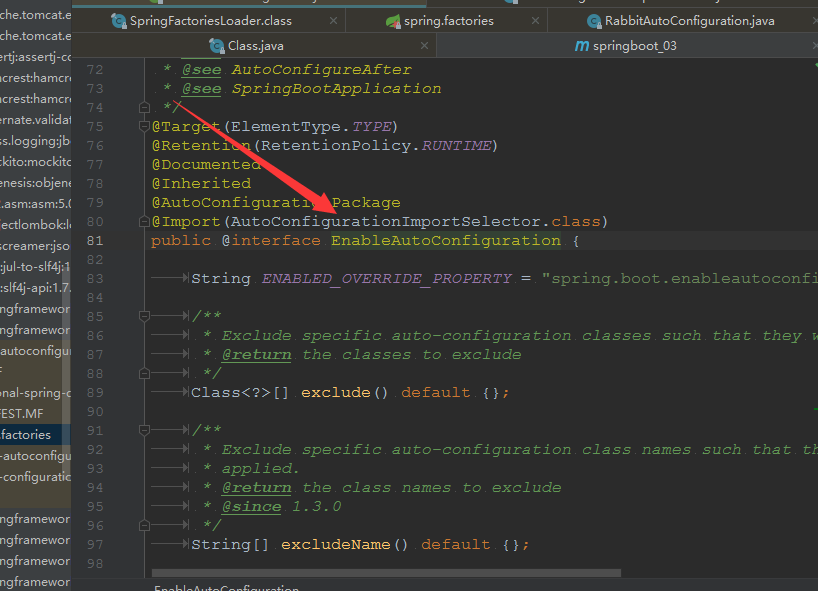
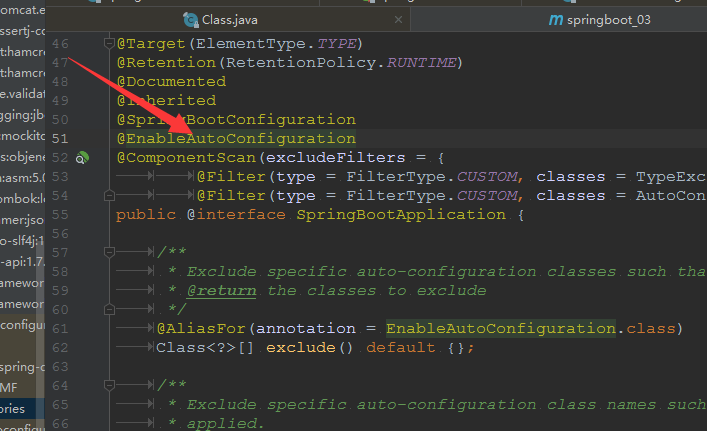
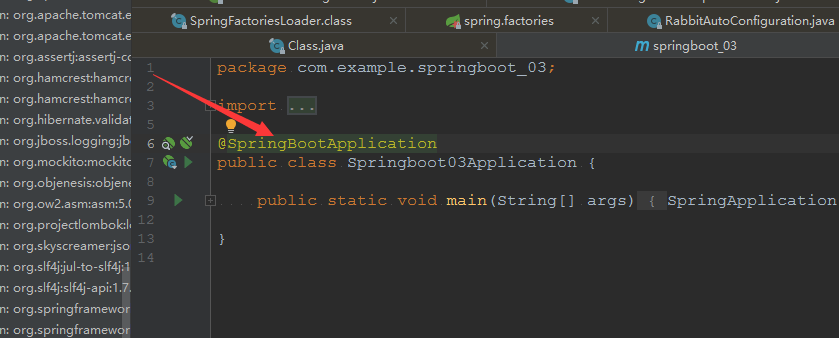
# 2 springboot 配置文件详解,配置文件中能写的东西

<https://docs.spring.io/spring-boot/docs/2.1.3.RELEASE/reference/htmlsingle/#common-application-properties>

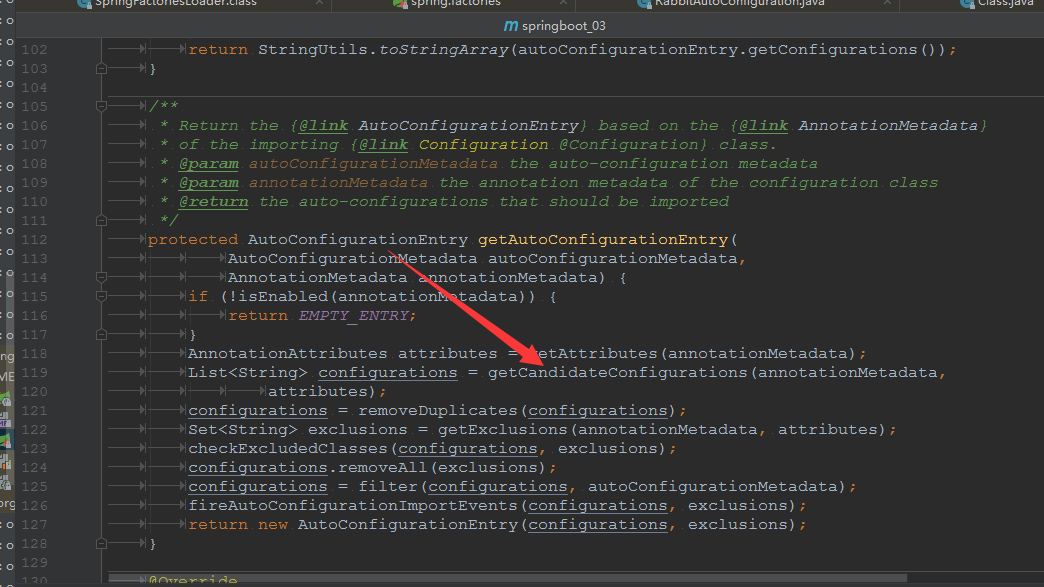
# 3 springboot 自动配置原理

1. 查看源码步骤:
2. 新建一个springboot工程,打开启动类

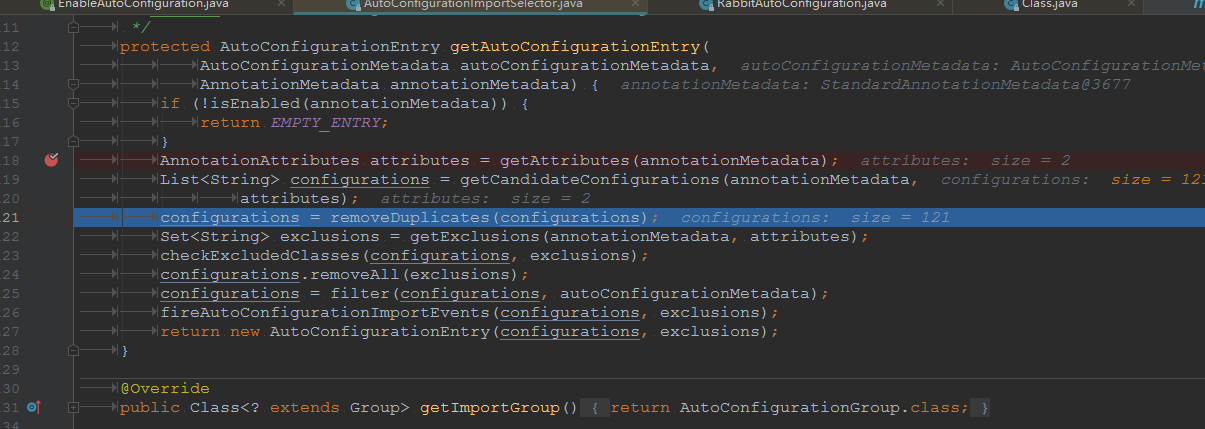
按住ctrl键点击注解:

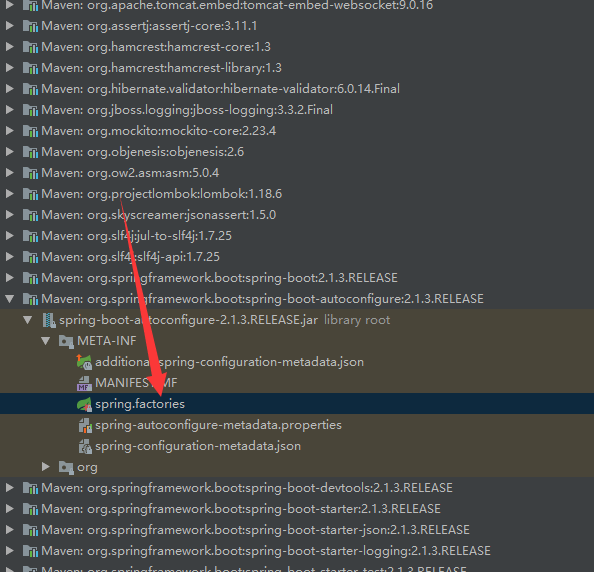
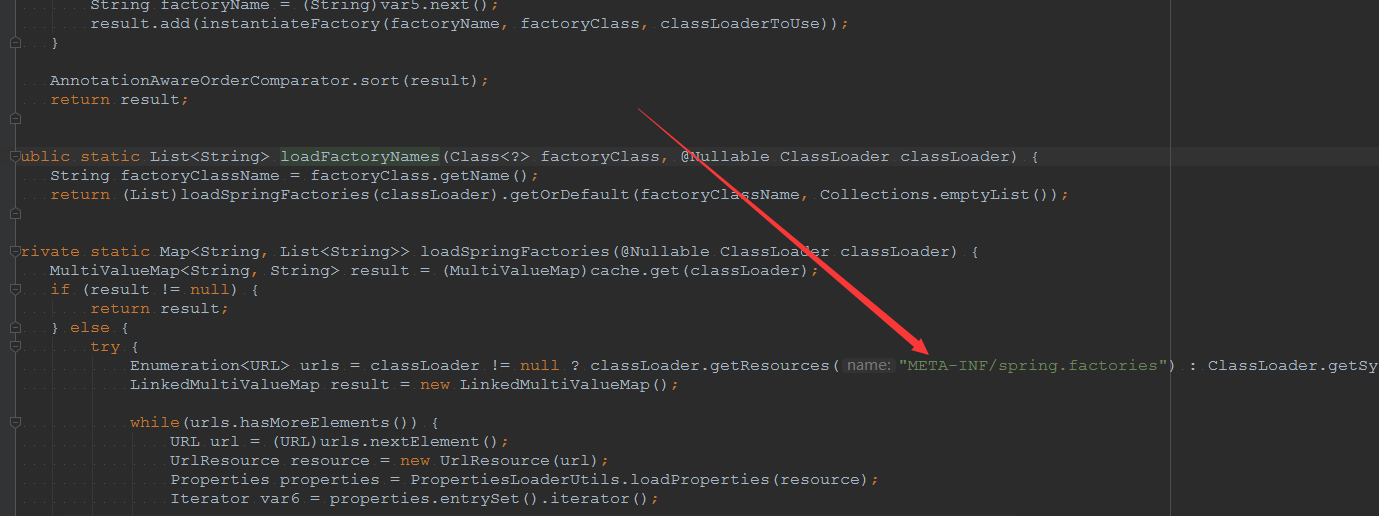
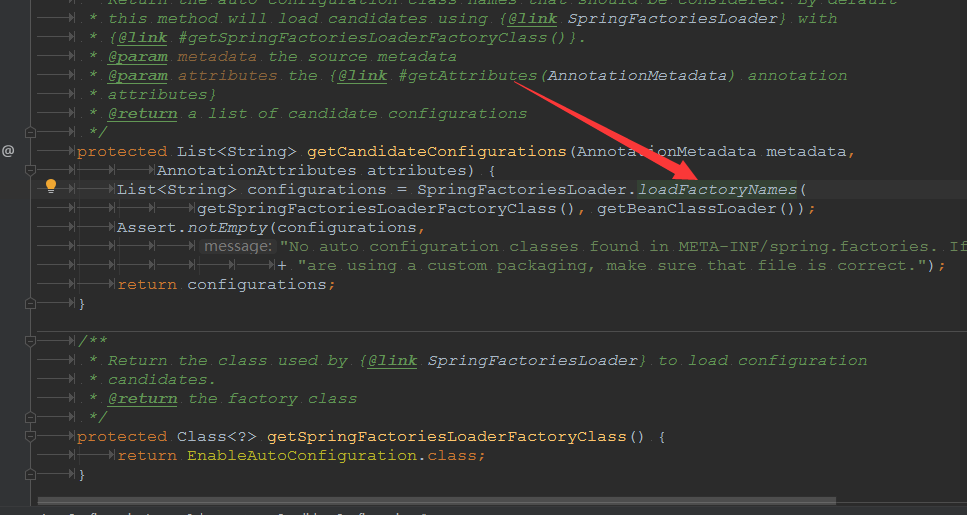
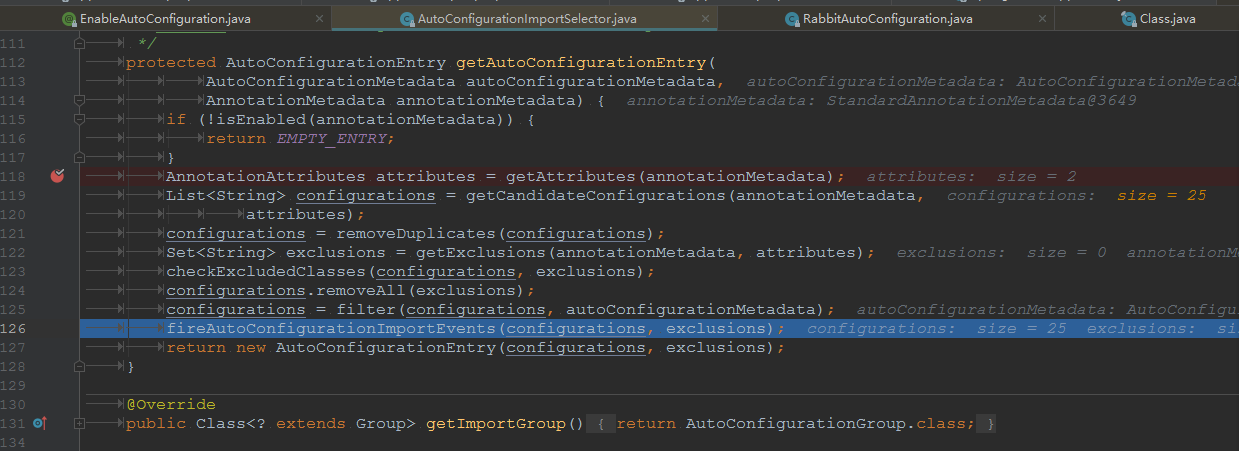


下面这个方法表示获取要进行配置的类

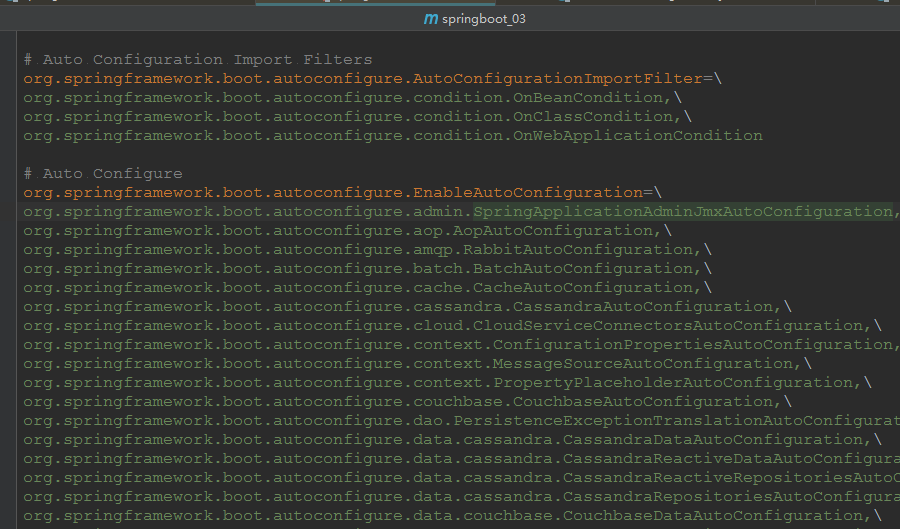


打断点运行项目可以看到加载进来的本来有121个，根据maven pom文件中的依赖配置，最后进行排除只有25个





点击下面的任何一个都可以看到实际配置到容器中的bean



## 在上面图片的自动配置类中找到MultipartAutoConfiguration，点击进去可以看到

// 启用自动配置属性，

@EnableConfigurationProperties(MultipartProperties.class)

点击MultipartProperties.class,进去可以看到我们可以在配置文件(.properties/.yml)中进行设置的属性，里面已经写好了默认值

// 注解表示从配置文件中注入属性，prefix表示配置文件中属性的前缀

@ConfigurationProperties(prefix = "spring.servlet.multipart", ignoreUnknownFields = false)  
public class MultipartProperties {  
  
 */\*\*  
 \* Whether to enable support of multipart uploads.  
 \*/* private boolean enabled = true;  
  
 */\*\*  
 \* Intermediate location of uploaded files.  
 \*/* private String location;  
  
 */\*\*  
 \* Max file size.  
 \*/* private DataSize maxFileSize = DataSize.*ofMegabytes*(1);  
  
 */\*\*  
 \* Max request size.  
 \*/* private DataSize maxRequestSize = DataSize.*ofMegabytes*(10);

接下来我们可以在根据前缀和属性名在配置文件中设置属性，例如:

spring.servlet.multipart.max-file-size=10

在配置文件中可以出现提示是因为在pom文件中使用了devtools工具类帮我们进行了加载:

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-devtools</artifactId>  
 <scope>runtime</scope>  
</dependency>

# 4 Springboot中扩展SpringMVC

## 1 消息转换器配置

首先定义一个实体类

@Data  
public class Person {  
  
 private String name;  
  
 //自定义转换给前端的json格式，此时添加的阿里的fastjson消息转换器会失效  
 //@JSONField(format = "yyyy-MM-dd")  
 private Date birthday;  
}

接着在pom文件中导入我们要使用的依赖

<!--阿里巴巴的json转换工具-->  
<dependency>  
 <groupId>com.alibaba</groupId>  
 <artifactId>fastjson</artifactId>  
 <version>1.2.47</version>  
</dependency>

然后在springboot中进行注册

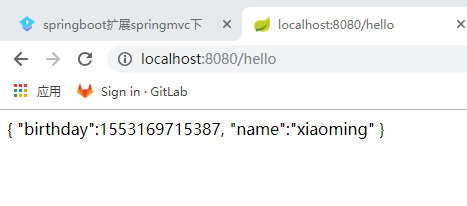
@Configuration  
public class MVCConfig implements WebMvcConfigurer {  
 //@Override  
 //public void addViewControllers(ViewControllerRegistry registry) {  
 // registry.addViewController("/intform").setViewName("form");  
 //}  
  
 // 注册fastjson 作为消息转换器  
 @Override  
 public void configureMessageConverters(List<HttpMessageConverter<?>> converters) {  
 FastJsonHttpMessageConverter fc = new FastJsonHttpMessageConverter();  
 FastJsonConfig fastJsonConfig = new FastJsonConfig();  
 fastJsonConfig.setSerializerFeatures(SerializerFeature.*PrettyFormat*);  
 fc.setFastJsonConfig(fastJsonConfig);  
 converters.add(fc);  
 }

使用效果:

进行ctroller的配置

@RestController  
public class controller {  
  
 @GetMapping("/hello")  
 public Person hello(){  
 Person person = new Person();  
 person.setBirthday(new Date());  
 person.setName("xiaoming");  
 return person;  
 }

进行访问



## 2 拦截器配置

首先自定义拦截器类:

public class MyInterceptor implements HandlerInterceptor {  
  
 @Override  
 public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) throws Exception {  
 System.*out*.println("--------拦截前--------");  
 return true;// 要返回true才能执行后面的步骤, 返回false时，postHandle,afterCompletion方法均不能执行  
 }  
  
 @Override  
 public void postHandle(HttpServletRequest request, HttpServletResponse response, Object handler, ModelAndView modelAndView) throws Exception {  
 System.*out*.println("--------拦截后--------");  
 }  
  
 @Override  
 public void afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler, Exception ex) throws Exception {  
 System.*out*.println("--------拦截中--------");  
 }  
}

然后注册进springboot中

@Configuration  
public class MVCConfig implements WebMvcConfigurer {  
 // 注册拦截器  
 @Override  
 public void addInterceptors(InterceptorRegistry registry) {  
 registry.addInterceptor(new MyInterceptor())  
 .addPathPatterns("/\*\*")//要拦截的请求  
 .excludePathPatterns("/hello");//不拦截的请求  
 }

使用效果:

访问：localhost:8080/hello时控制台没有打印拦截信息

访问：localhost:8080/hsfs时控制台信息如下所示

--------拦截前--------

--------拦截后--------

--------拦截中--------

--------拦截前--------

--------拦截后--------

--------拦截中--------