主讲老师：Fox

> 有道笔记链接： https://note.youdao.com/s/Hvb8rWlb

学习本课程前提：

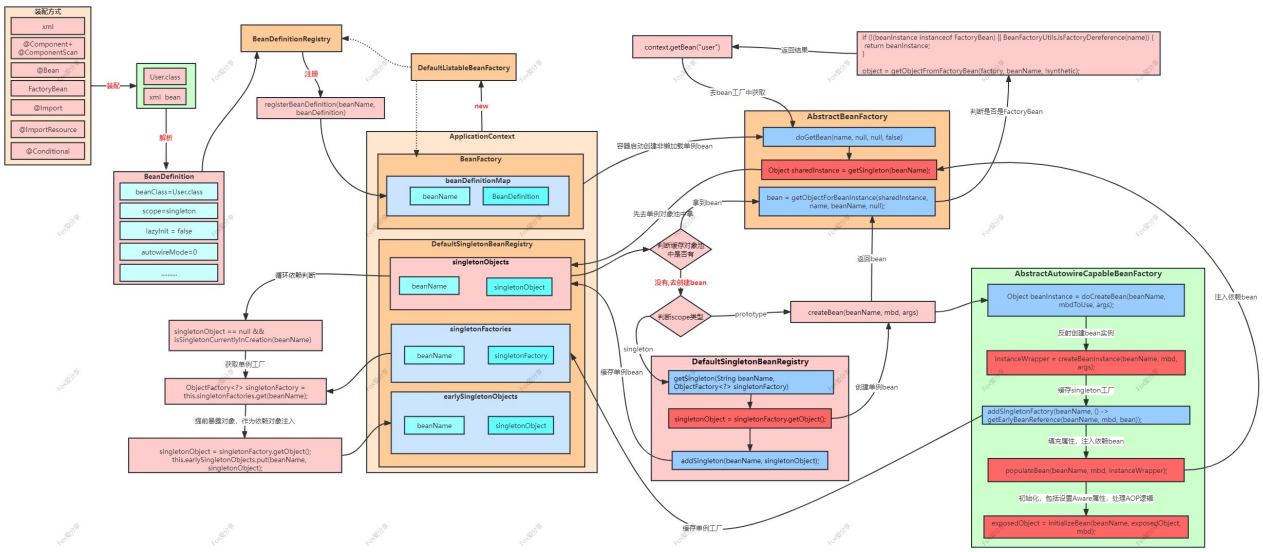
- 掌握Spring主线流程源码
- 掌握Spring Boot主线流程源码
- 熟悉Spring Cloud&Spring Cloud Alibaba中间件核心功能源码

1. Spring扩展点梳理

- **BeanFactoryPostProcessor**
  - **BeanDefinitionRegistryPostProcessor**
- **BeanPostProcessor**
  - **InstantiationAwareBeanPostProcessor**
  - **AbstractAutoProxyCreator**
- **@Import**
  - **ImportBeanDefinitionRegistrar**
  - **ImportSelector**
- **Aware**
  - **ApplicationContextAware**
  - **BeanFactoryAware**
- **InitializingBean || @PostConstruct**
- **FactoryBean**
- **SmartInitializingSingleton**
- **ApplicationListener**
- **Lifecycle**
  - **SmartLifecycle**
  - **LifecycleProcessor**
- **HandlerInterceptor**
- **MethodInterceptor**

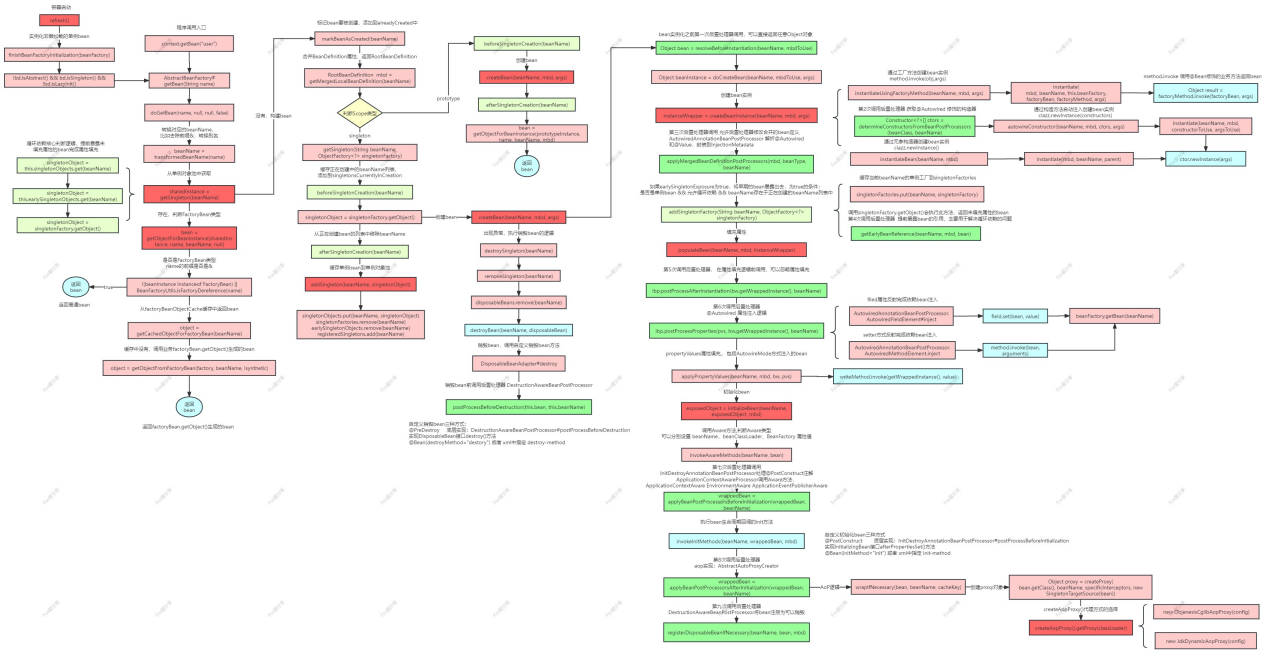
IoC工作原理

https://www.processon.com/view/link/5cd10507e4b085d010929d02



Bean生命周期主线流程：

https://www.processon.com/view/link/5eafa609f346fb177ba8091f



2. Spring扩展点在微服务组件中的应用场景

2.1 整合Nacos

ApplicationListener扩展场景——监听容器中发布的事件

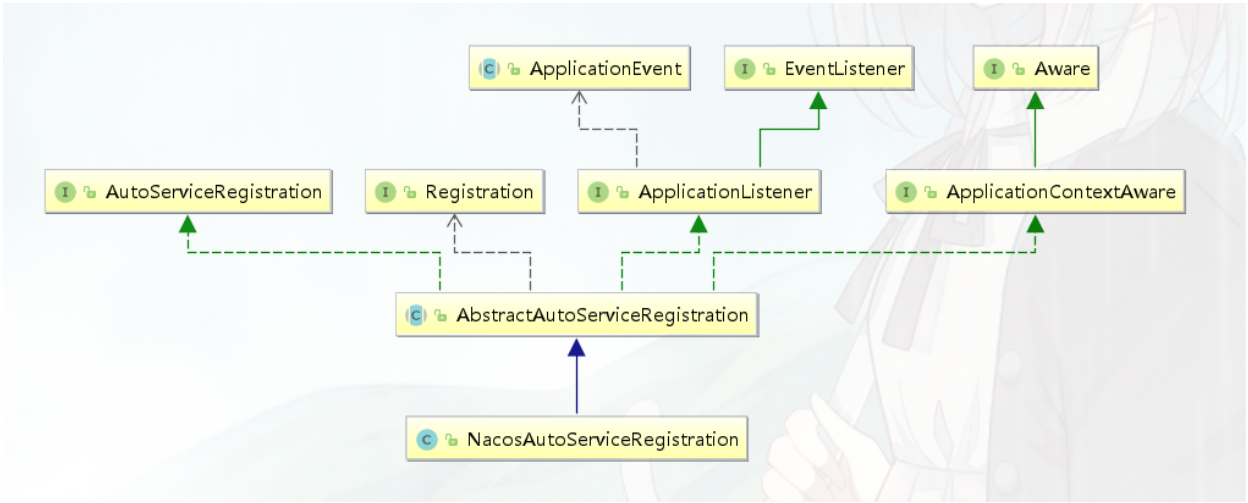
思考： 为什么整合Nacos注册中心后，服务启动就会自动注册，Nacos是如何实现自动服务注册的？

```
s.b.a.e.web.EndpointLinksResolver      : Exposing 19 endpoint(s) beneath base path '/actuator'
s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8021 (http) with context path ''
a.c.n.registry.NacosServiceRegistry    : nacos registry, DEFAULT_GROUP mall-order 192.168.65.103:8021 register fi
t.mall.order.MallOrderApplication      : Started MallOrderApplication in 15.852 seconds (JVM running for 27.986)
```

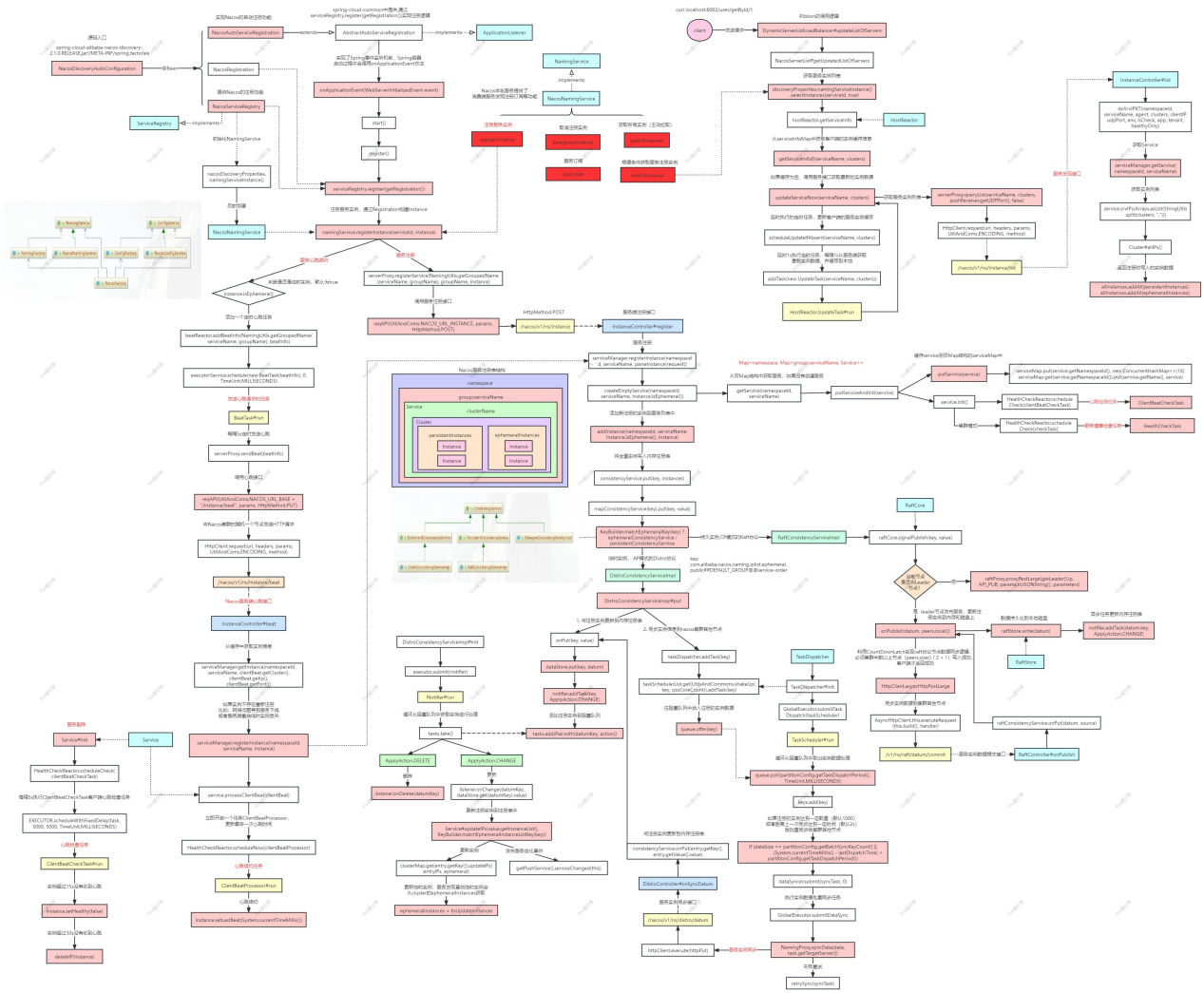
服务启动是自动注册到nacos注册中心

NacosAutoServiceRegistration

```
# 对ApplicationListener的扩展
AbstractAutoServiceRegistration#onApplicationEvent
# 服务注册
》NacosServiceRegistry#register
```



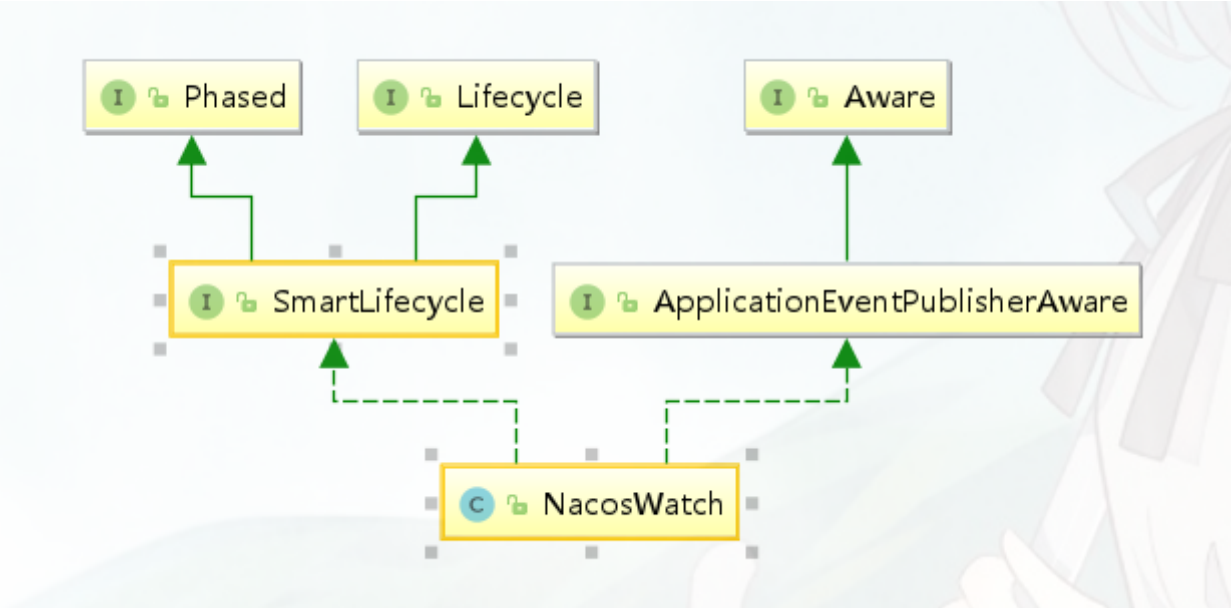
Nacos注册中心源码分析https://www.processon.com/view/link/5ea27ca15653bb6efc68eb8c



Lifecycle扩展场景——管理具有启动、停止生命周期需求的对象

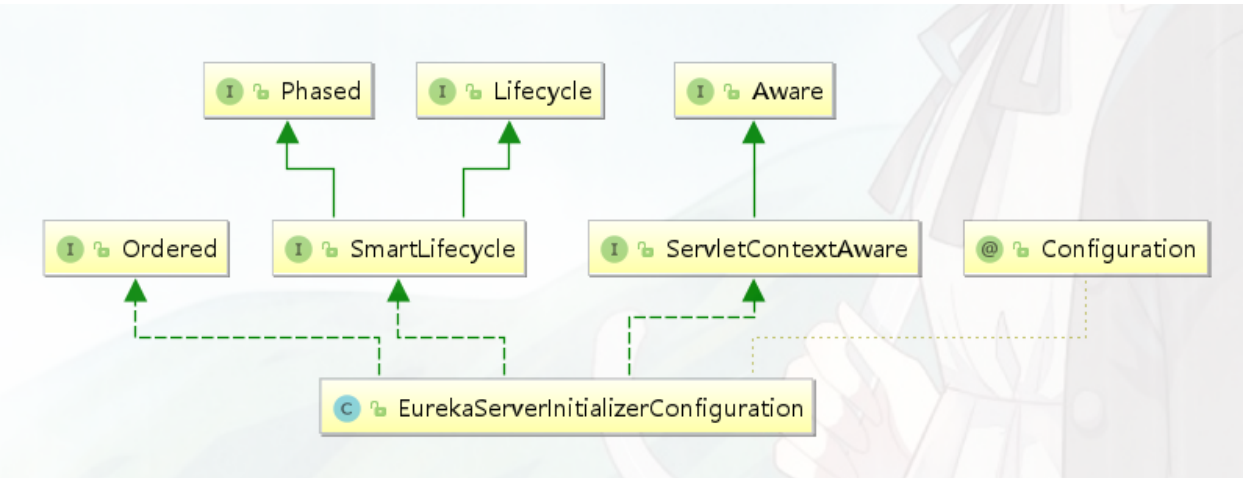
NacosWatch

```
#对SmartLifecycle的扩展
NacosWatch#start
#订阅服务接收实例更改的事件
》NamingService#subscribe
```



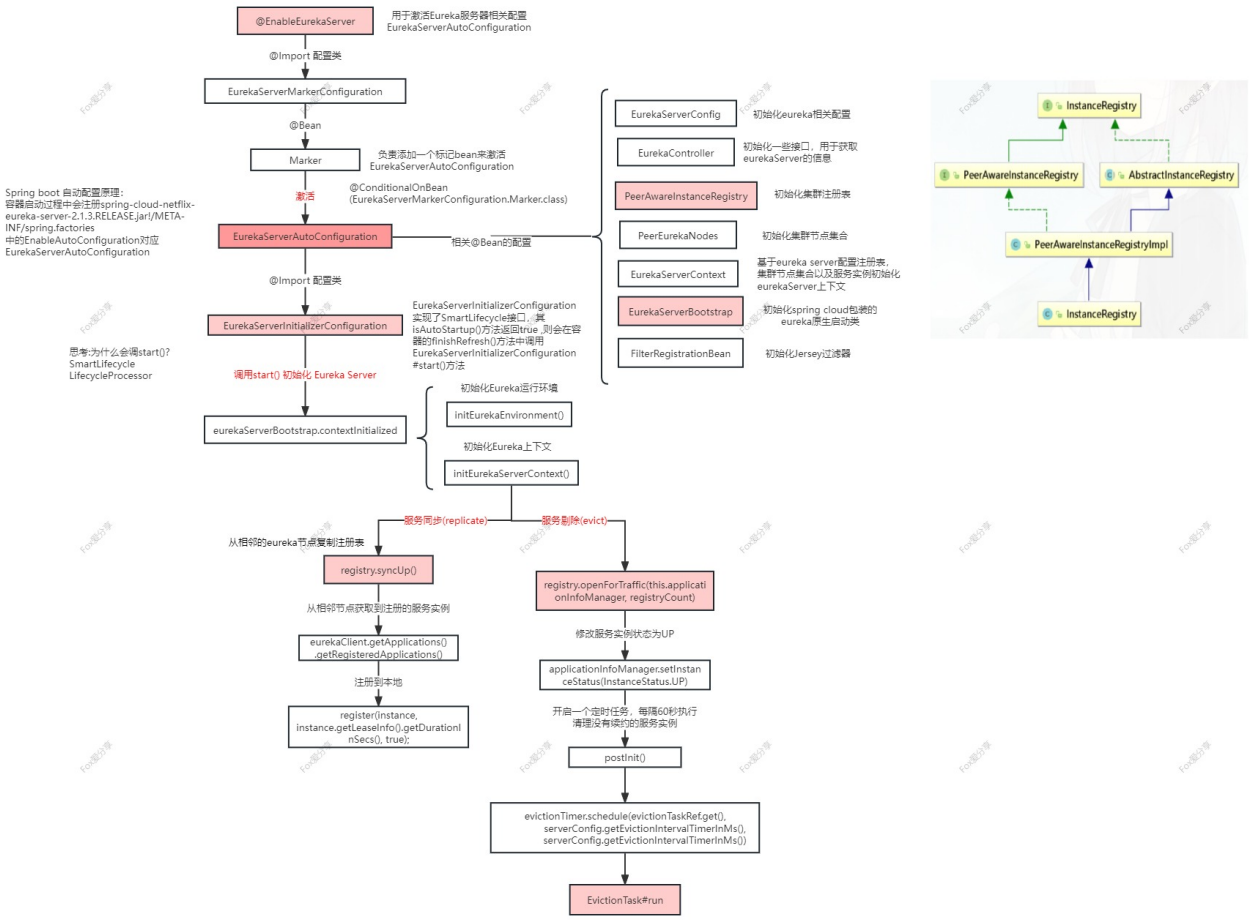
扩展： Eureka Server端上下文的初始化是在SmartLifecycle#start中实现的

EurekaServerInitializerConfiguration



Eureka Server源码分析：

https://www.processon.com/view/link/5e5fa095e4b0a967bb35b667



2.2 整合Ribbon、LoadBalancer

SmartInitializingSingleton扩展场景—— 对容器中的Bean对象进行定制处理

思考：为什么@Bean修饰的RestTemplate加上@LoadBalanced就能实现负载均衡功能?

```
@Bean
@LoadBalanced
public RestTemplate restTemplate() {
    return new RestTemplate();
}
```

LoadBalancerAutoConfiguration

对SmartInitializingSingleton的扩展，为所有用@LoadBalanced修饰的restTemplate（利用了@Qualifier）绑定实现了负载均衡逻辑的拦截器LoadBalancerInterceptor

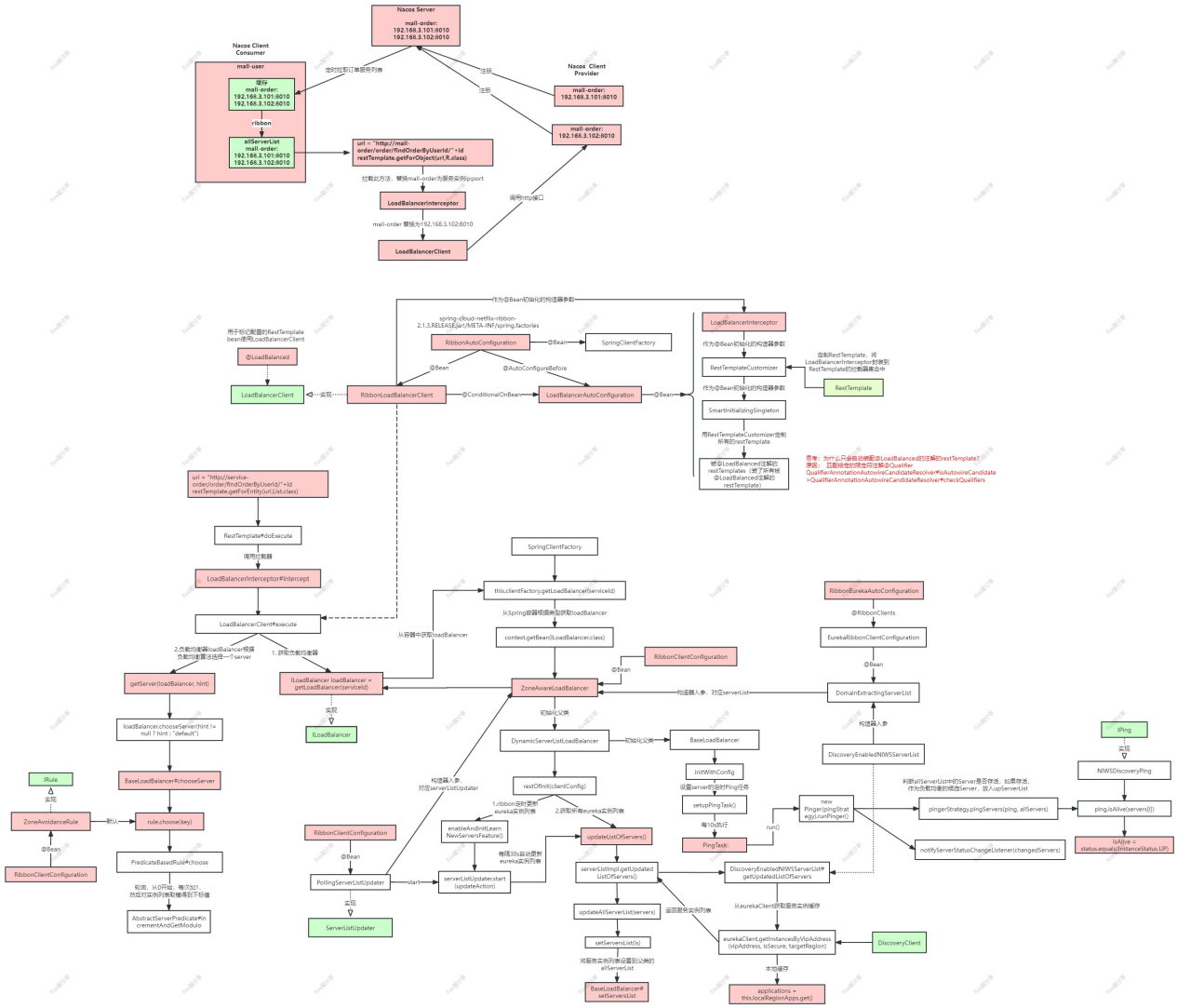
```java
public class LoadBalancerAutoConfiguration {

    @LoadBalanced
    @Autowired(required = false)
    private List<RestTemplate> restTemplates = Collections.emptyList();

    @Autowired(required = false)
    private List<LoadBalancerRequestTransformer> transformers = Collections.emptyList();

    @Bean
    public SmartInitializingSingleton loadBalancedRestTemplateInitializerDeprecated(
            final ObjectProvider<List<RestTemplateCustomizer>> restTemplateCustomizers) {
        return () -> restTemplateCustomizers.ifAvailable(customizers -> {
            for (RestTemplate restTemplate : LoadBalancerAutoConfiguration.this.restTemplates) {
                for (RestTemplateCustomizer customizer : customizers) {
                    customizer.customize(restTemplate);
                }
            }
        });
    }
```

LoadBalancerInterceptor

```java
@Bean
@ConditionalOnMissingBean
public RestTemplateCustomizer restTemplateCustomizer(
        final LoadBalancerInterceptor loadBalancerInterceptor) {
    return restTemplate -> {
        List<ClientHttpRequestInterceptor> list = new ArrayList<>(
                restTemplate.getInterceptors());
        list.add(loadBalancerInterceptor);
        restTemplate.setInterceptors(list);
    };
}
```

Ribbon源码分析：

https://www.processon.com/view/link/5e7466dce4b027d999bdaddb



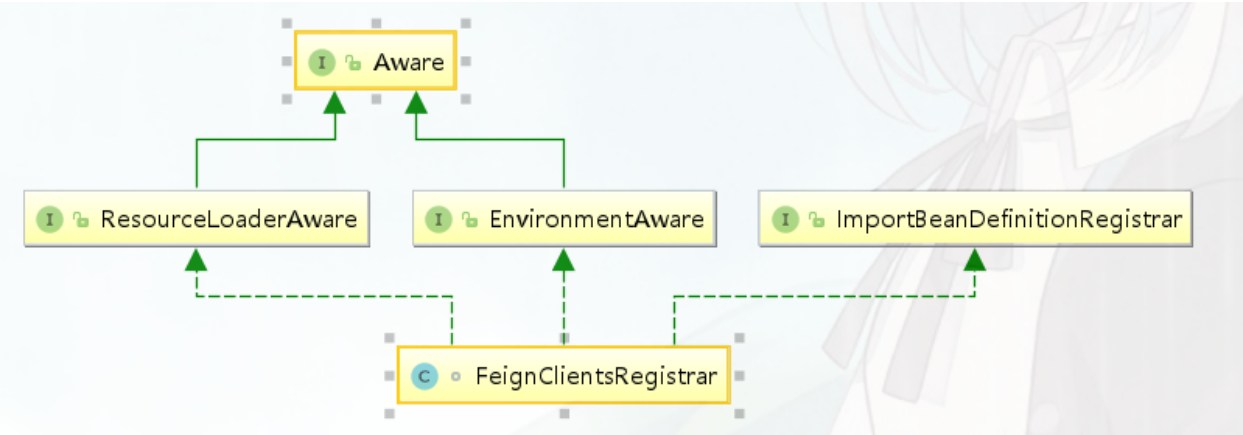
2.3 整合Feign

FactoryBean的扩展场景——将接口生成的代理对象交给Spring管理

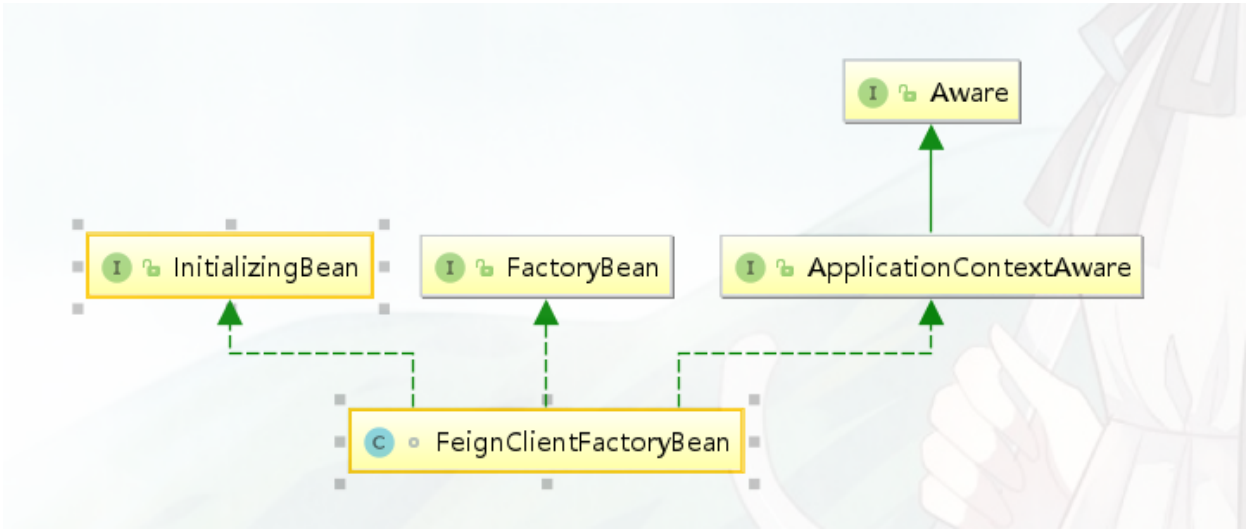
思考：为什么Feign接口可以通过@Autowired直接注入使用？Feign接口是如何交给Spring管理的?

```java
@FeignClient(value = "mall-order",path = "/order")
public interface OrderFeignService {

    @RequestMapping("/findOrderByUserId/{userId}")
    R findOrderByUserId(@PathVariable("userId") Integer userId);
}

@RestController
@RequestMapping("/user")
public class UserController {

    @Autowired
    OrderFeignService orderFeignService;

    @RequestMapping(value = "/findOrderByUserId/{id}")
    public R  findOrderByUserId(@PathVariable("id") Integer id) {
        //feign调用
        R result = orderFeignService.findOrderByUserId(id);
        return result;
    }
}
```

FeignClientsRegistrar

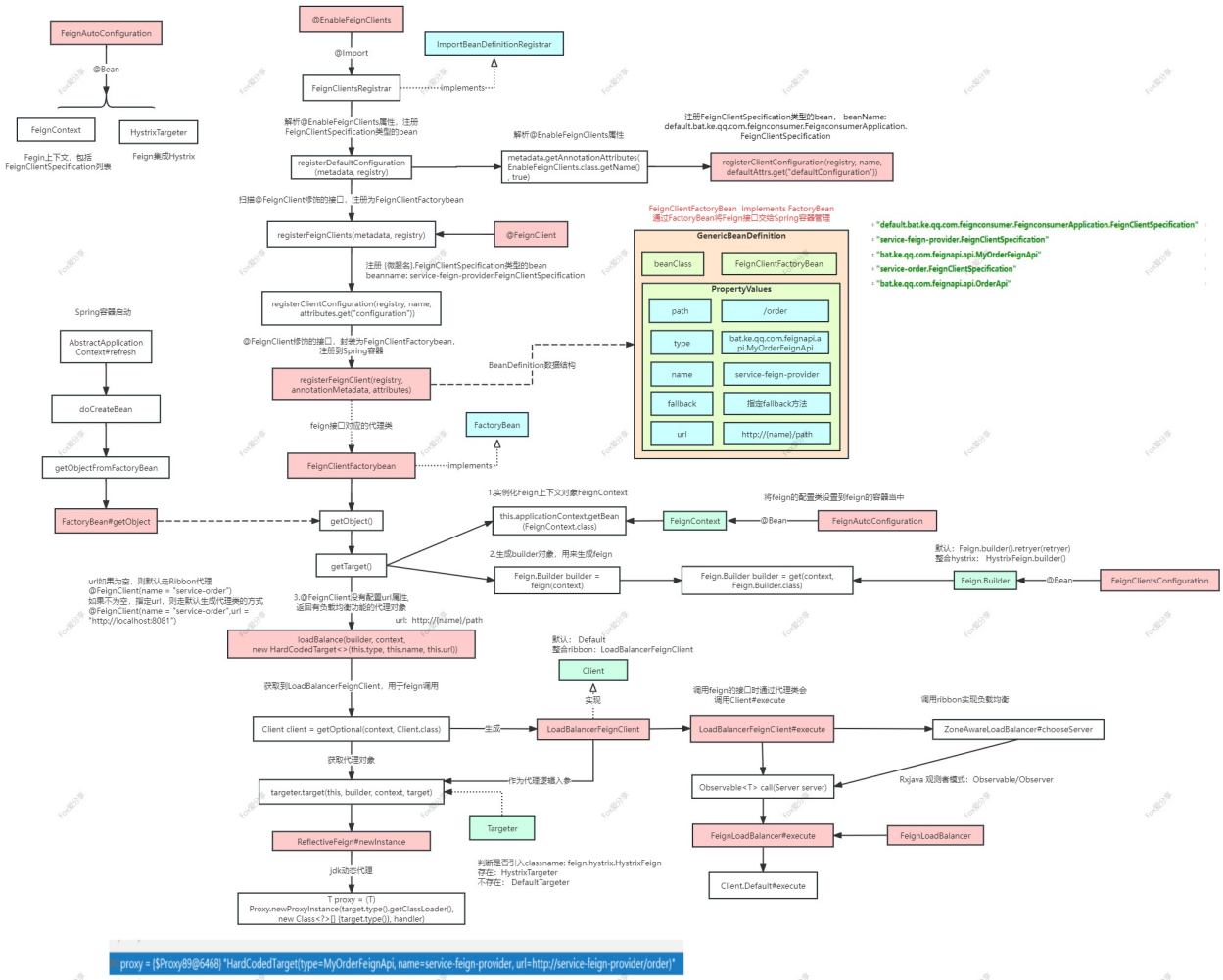


FeignClientFactorybean



Feign源码分析：

https://www.processon.com/view/link/5e80ae79e4b03b99653fe42f

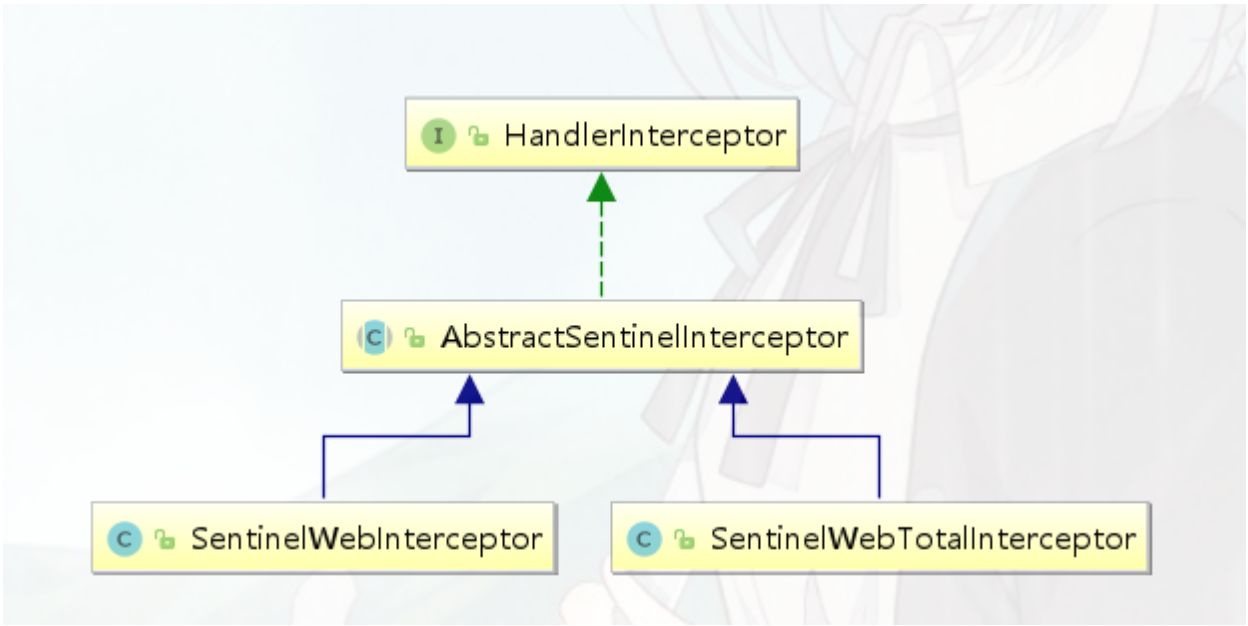


2.4 整合sentinel

HandlerInterceptor扩展场景——对mvc请求增强

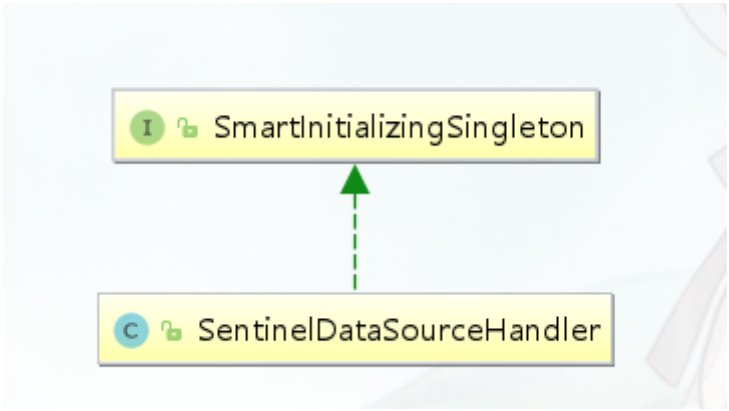
AbstractSentinelInterceptor

```
# Webmvc接口资源保护入口
AbstractSentinelInterceptor#preHandle
```

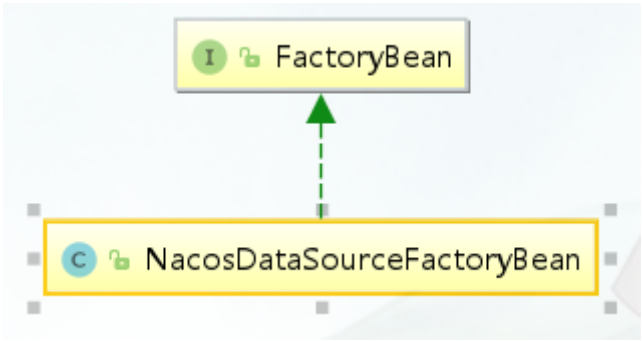


SmartInitializingSingleton&FactoryBean结合场景——根据类型动态装配对象 SentinelDataSourceHandler

```
#Sentinel持久化读数据源设计，利用了SmartInitializingSingleton扩展点
SentinelDataSourceHandler#afterSingletonsInstantiated
# 注册一个FactoryBean类型的数据源
》SentinelDataSourceHandler#registerBean
》》NacosDataSourceFactoryBean#getObject
# 利用FactoryBean获取到读数据源
》》new NacosDataSource(properties, groupId, dataId, converter)
```

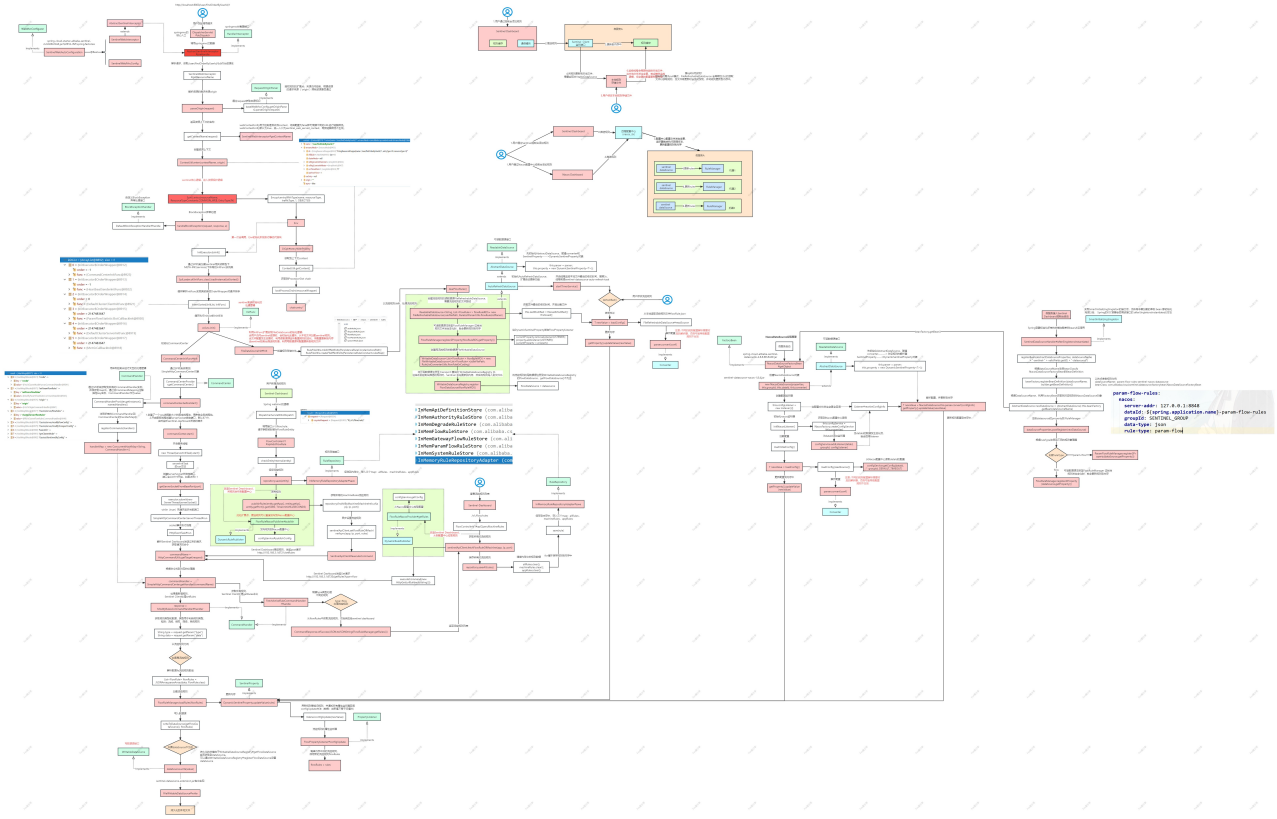


NacosDataSourceFactoryBean



sentinel规则持久化源码分析：

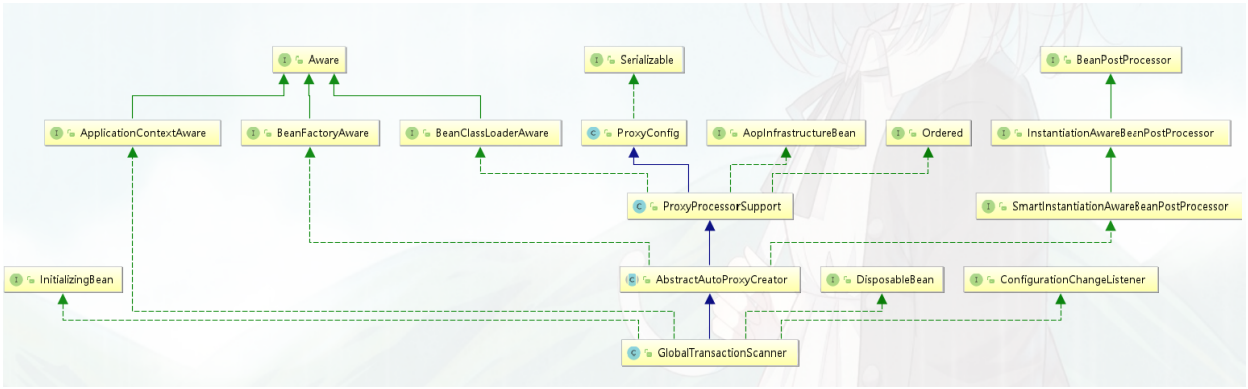
https://www.processon.com/view/link/607fef267d9c08283ddc2f8d



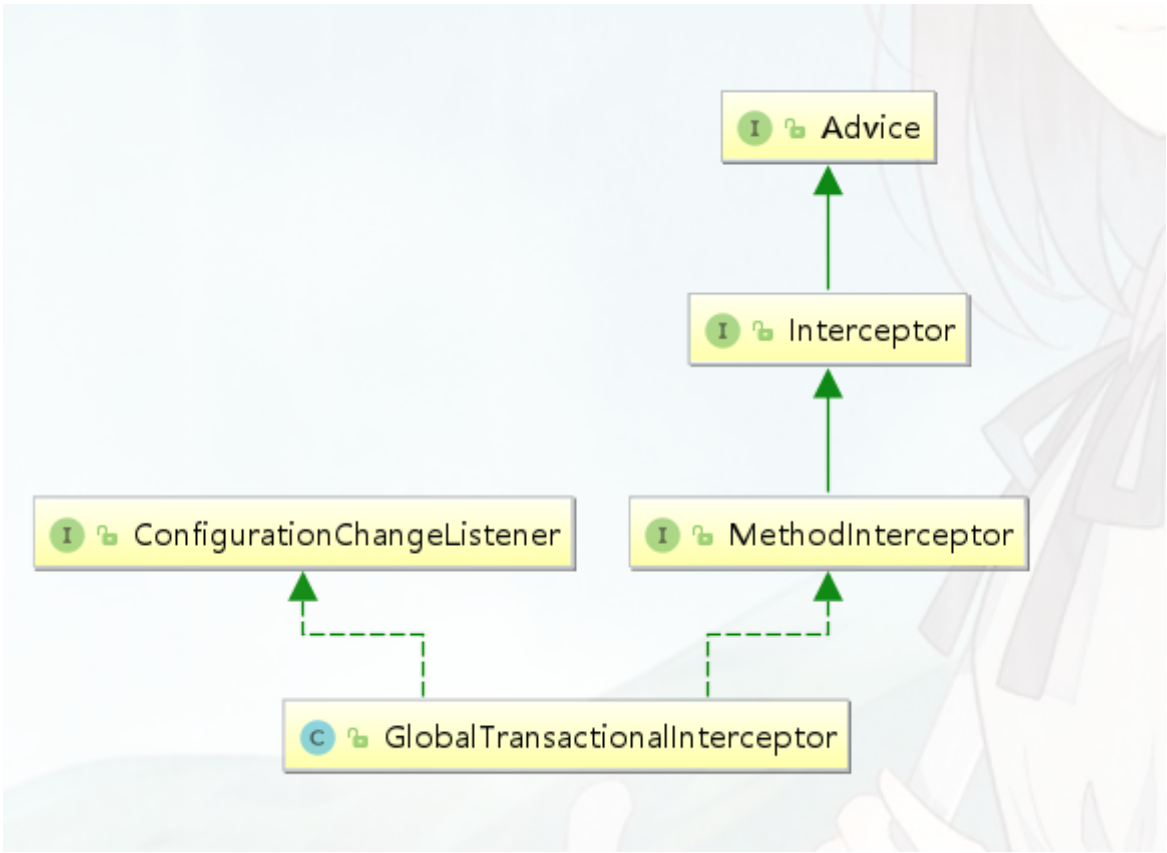
2.5 整合seata

AbstractAutoProxyCreator&MethodInterceptor结合场景——实现方法增强

GlobalTransactionScanner

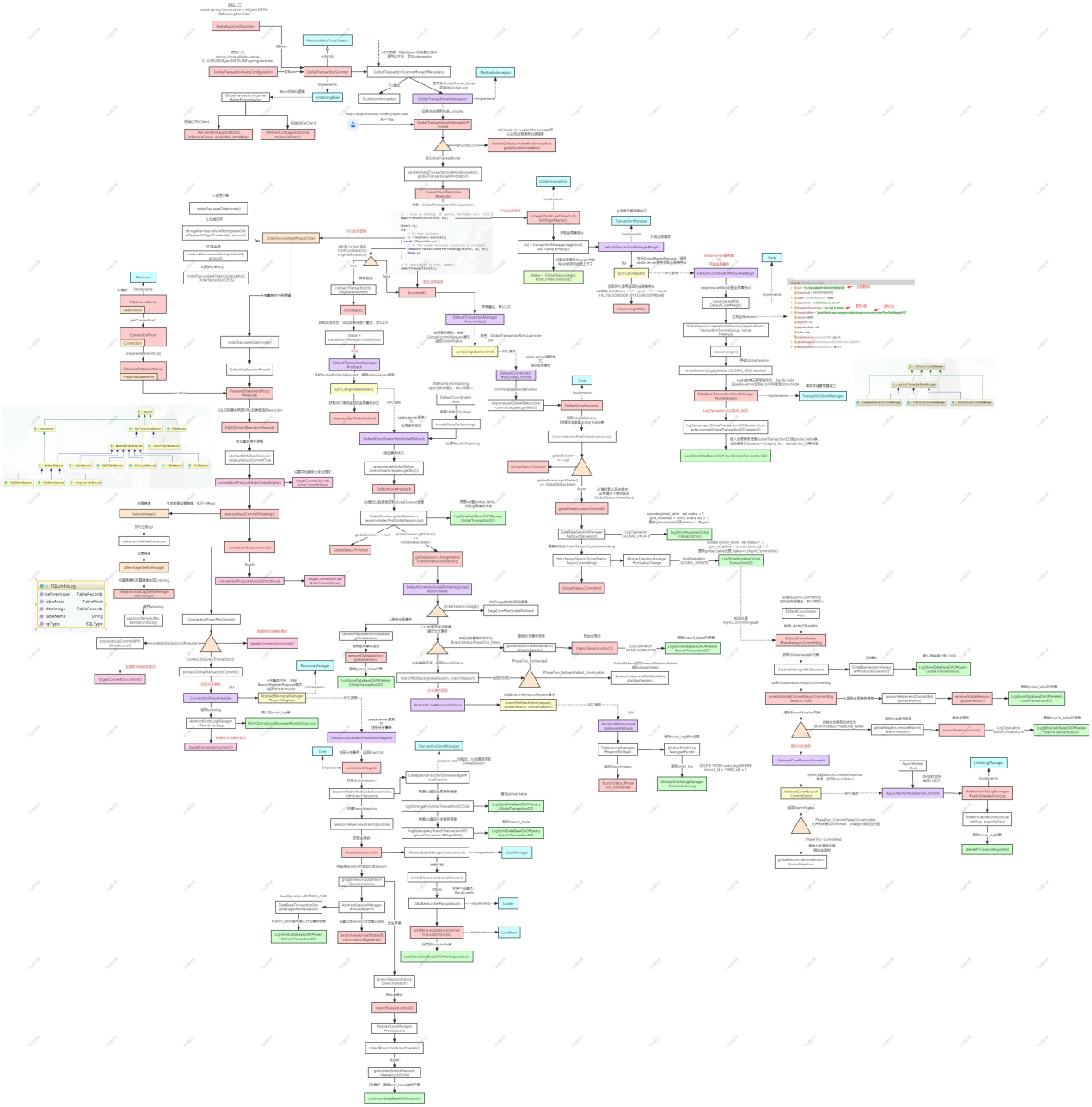


GlobalTransactionalInterceptor



Seata源码分析:

https://www.processon.com/view/link/5f743063e0b34d0711f001d2



3.使用AI高效学习微服务组件源码

idea插件推荐

- **TONGYI Lingma**

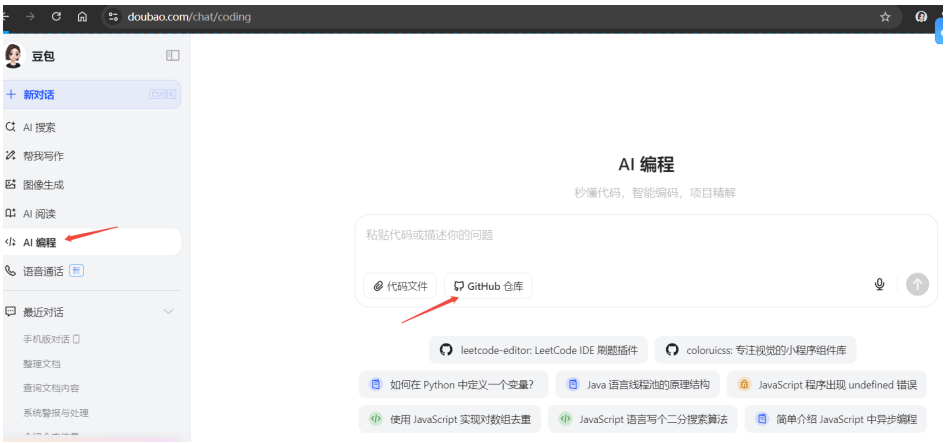
通义灵码，是一款基于通义大模型的智能编码辅助工具，提供行级/函数级实时续写、自然语言生成代码、单元测试生成、代码注释生成、代码解释、研发智能问答、异常报错排查等能力，并针对阿里云SDK/API 的使用场景调优，为开发者带来高效、流畅的编码体验。

- **Baidu Comate**

文心快码 - 百度智能编码助手，您的人工智能编程伙伴。这款基于人工智能的智能代码生成工具，让您的编码更快、更好、更简单！文心快码由 ERNIE-Code 驱动，该模型基于百度多年积累的非敏感代码数据以及来自 GitHub 的顶级公开代码数据进行训练。它能自动生成完整且更贴合特定场景的代码行或代码块，助力每一位开发者轻松完成开发任务。

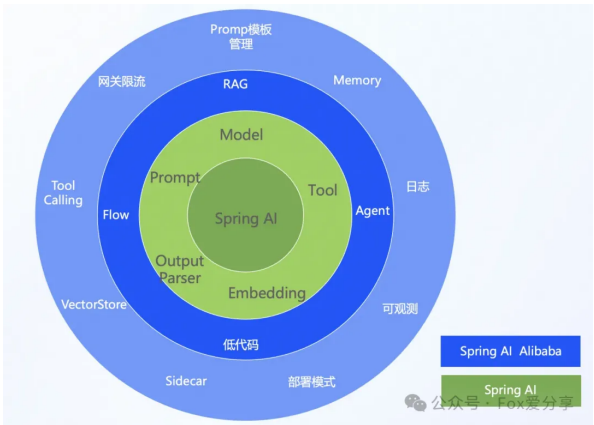
- **MarsCode AI**

MarsCode AI 是豆包旗下的智能编程助手，提供以智能代码补全为代表的核心能力，支持主流编程语言及 IDE，能在编码过程中提供单行或整个函数的建议，同时支持在用户编码过程中提供代码解释、单测生成、问题修复、技术问答等辅助功能，提升编码效率与质量。



SpringBoot + Spring AI Alibaba实现AI智能体应用开发

Spring 官方开源了 Spring AI 框架，用来简化 Spring 开发者开发智能体应用的过程。随后阿里巴巴开源了 Spring AI Alibaba，它基于 Spring AI，同时与阿里云百炼大模型服务、通义系列大模型做了深度集成与最佳实践。基于 Spring AI Alibaba，Java 开发者可以非常方便地开发 AI 智能体应用。



官网地址：https://java2ai.com/

接入DeepSeek：

SpringBoot + Spring AI Alibaba 整合阿里云百炼DeepSeek大模型，小白也能轻松上手