主讲老师：Fox

> 有道笔记链接： https://note.youdao.com/s/VJyTfT7A

学习本课程前提：

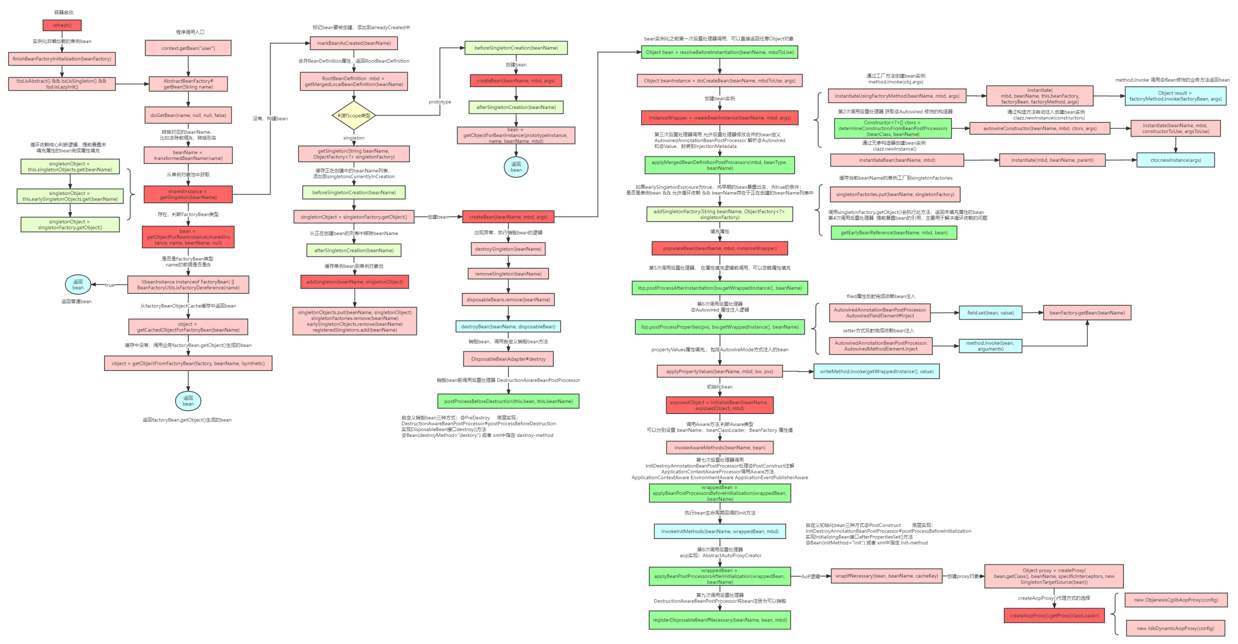
- 掌握Spring主线流程源码
- 掌握Spring Boot主线流程源码
- 熟悉Spring Cloud&Spring Cloud Alibaba中间件核心功能源码

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**

Bean生命周期主线流程：

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



2. Spring扩展点应用场景

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?

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

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

LoadBalancerInterceptor

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

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

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

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