# ▶ **spring boot启动流程**

1、新建module,在主程序类加入断点，启动springboot

2、首先进入SpringAplication类run方法

3、run方法新建SpringApplication对象

4、SpringApplication对象的run方法，首先创建并启动计时监控类

5、接着通过configureHeadlessProperty设置java.awt.headless的值

6、接着调用getRunListeners创建所有spring监听器

7、接着DefaultApplicationArguments初始化应用应用参数

8、接着prepareEnvironment根据运行监听器和参数准备spring环境

9、接着调用createApplicationContext方法创建应用上下文

10、通过prepareContext准备应用上下文

11、refreshContext方法刷新上下文

12、调用stop方法停止计时监控器类

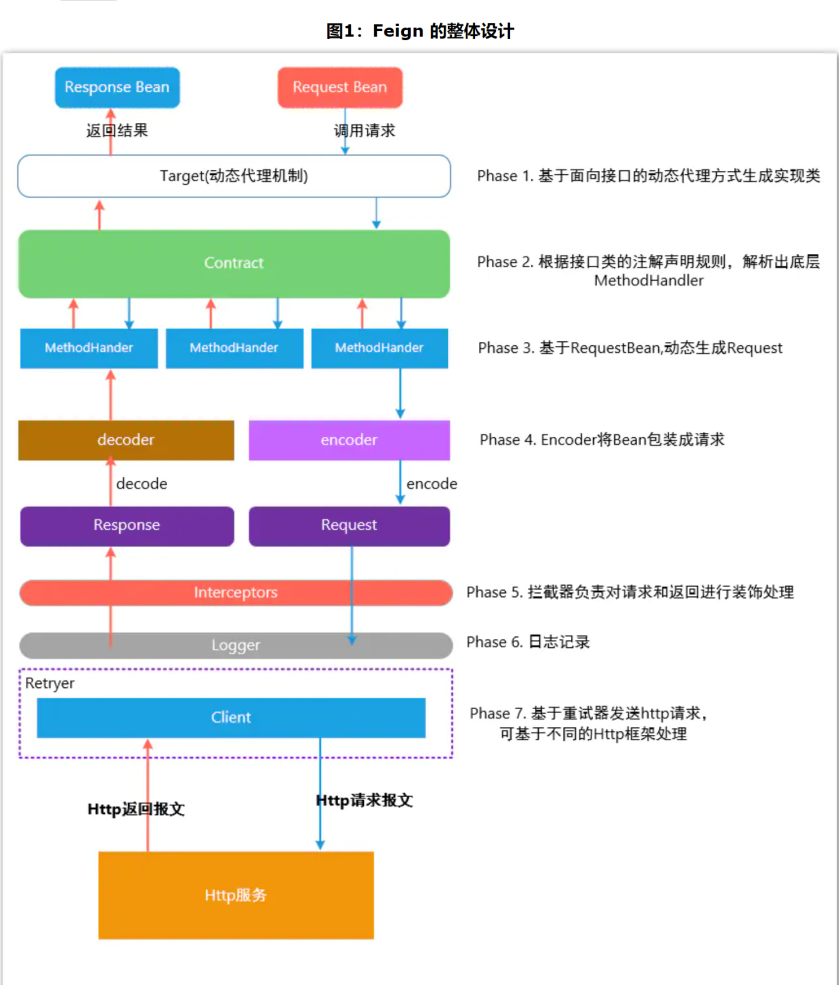
13、调用started发布应用上下文启动完成事件

14、callRunners方法执行所有runner运行器

15、调用running发布应用上下文就绪事件

16、最后返回应用上下文

▶**feign调用原理**



1.@FeignClient远程调用接口开始调用。

2.由Target动态代理进行MethodHandler方法处理器分发。

3.构造request实例。

4.由Encoder包装Request，执行相应的装饰器，记录日志。

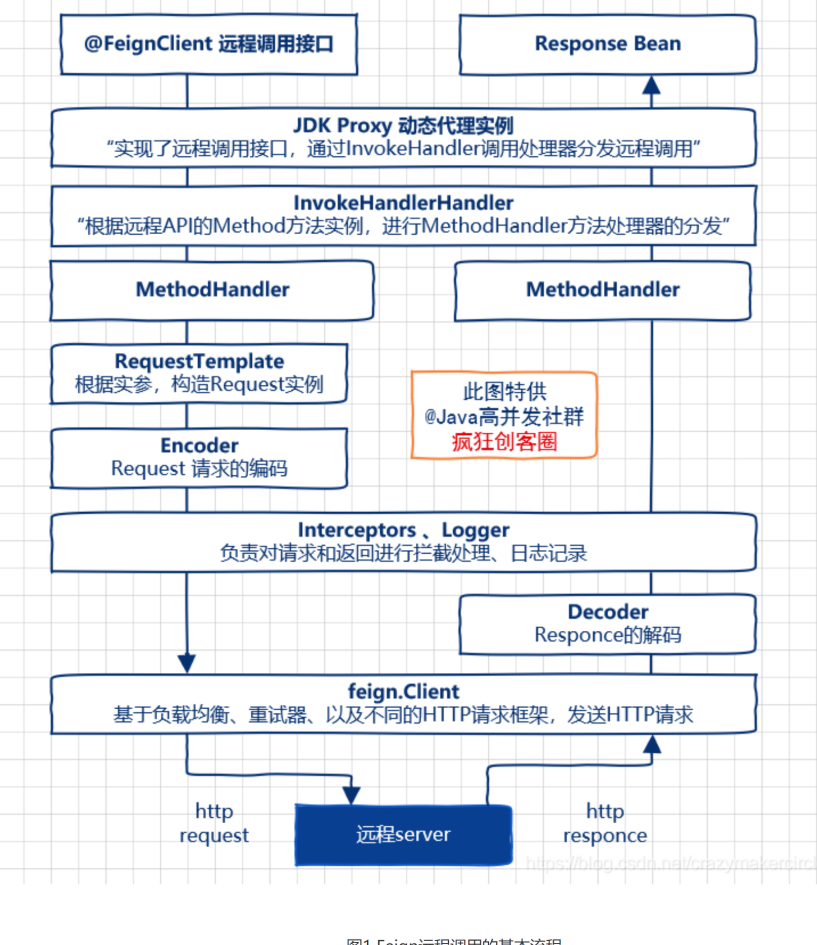
5.基于feign.client发起请求。

6.取得请求Response，由Decoder解码。

7.执行Response相关的MethodHandler。

8.经由代理类返回最终结果。

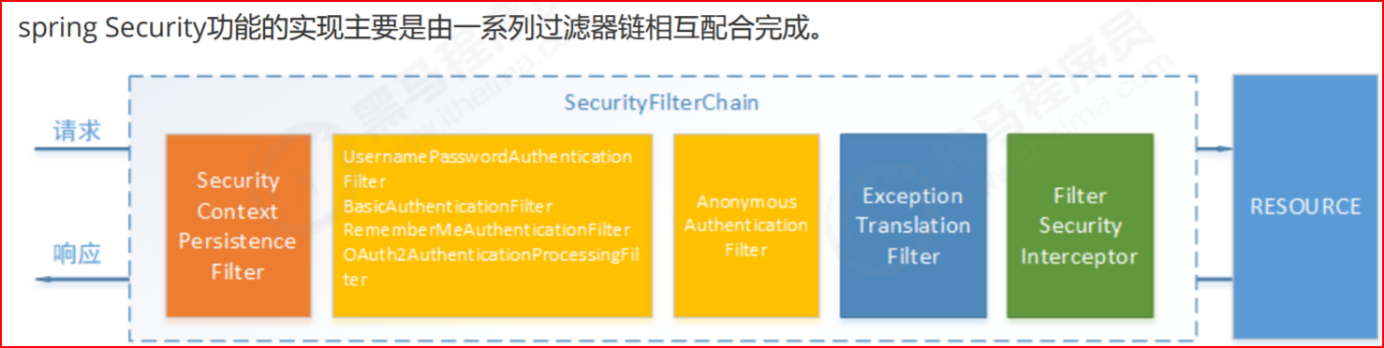
或者



▶**两个版本的区别与联系**



▶**Spring Security 登陆认证流程**



◇**SecurityContextPersistenceFilter**

这个Filter是整个拦截过程的入口和出口

◇**UsernamePasswordAuthenticationFilter**

(先获取用户名和密码，封装成UsernamePasswordToken，调用AuthenticationManager进行验证)

◇**AuthenticationManager**

AuthenticationManager 它自己不直接处理认证请求，而是委托给其所配置的 AuthenticationProvider列表， 然后会依次使用每一个 AuthenticationProvider 进行认证，如果 有一个AuthenticationProvider 认证后的结果 不为 null，则表示该AuthenticationProvider已经 认证成功，之后的AuthenticationProvider 不再继续认证。如果所有的 AuthenticationProvider 的 认证结果都为null，则表示认证失败，将抛出一个 ProviderNotFoundException。

◇**AuthenticationProvider**

(请求认证处理) [默认实现类：DaoAuthenticationProvider]

DaoAuthenticationProvider认证过程：DaoAuthenticationProvider先调用UserDetailsService 的 loadUserByUsername()方法获取UserDetails， 获取后再与UsernamePasswordAuthenticationFilter 获取的username和password进行比较；如果认证通过后会将该 UserDetails 赋给认证通过的 Authentication的principal，然后再把该 Authentication 存入到 SecurityContext 中。 默认情 况下，在认证成功后ProviderManager也将清除返回的Authentication中的凭证信息。

▶**Spring Cloud**

◇**为什么用Feign**

如果没有Feign，我们需要实现如下内容：用负载平衡client 获取服务实例，然后获取基本URL， 利用REST模板来调用服务



负载均衡

远程调用

▶**Eureka**

◇**服务提供者**

服务提供者启动，会注册到注册中心，并同步其他注册中心，服务提供者会按照心跳时间往注 册中心发请求，如果超过服务的过期失效时间，注册中心没有收到心跳请求，注册中心默认其 下线。

◇**服务消费者**

消费者服务启动时，会发送一个请求给服务注册中心，来获取上面注册的服务清单，该缓存清 单默认会每隔30秒更新一次。通过服务名可以获取具体提供服务的实例名和该实例的元数据信 息，客户端可以根据自己的需要决定具体调用哪个实例

◇**自保机制**

最后 1 分钟收到客户端实例续约的总次数 小于 期望每分钟收到客户端实例续约的总次数 则 进行自我保护机制

Eureka 的自我保护模式是有意义的，该模式被激活后，它不会从注册列表中剔除因长时间 没收到心跳导致租期过期的服务，而是等待修复，直到心跳恢复正常之后，它自动退出自 我保护模式

◇**缺点**

某个服务不可用时，各个Eureka Client不能及时的知道，需要1~3个⼼心跳周期才能感知， 但是，由于服务调⽤用端都会使⽤用Hystrix来容错和降级，弥补了了基于客户端服务发现的 时效性的缺点。

▶**Hystrix**

◇**熔断器原理**

当Hystrix Command请求后端服务时，在一定时间内（metrics.rollingStats.timeInMilliseconds，默认10s），请求次数超过了最低要求（circuitBreaker.requestVolumeThreshold，默认20次），并且其失败数量超过一定比例(circuitBreaker.errorThresholdPercentage，默认50%)，断路器会切换到开路状态(Open)。这时所有请求会直接失败而不会发送到后端服务. 断路器保持在开路状态一段时间后(circuitBreaker.sleepWindowInMilliseconds，默认5秒), 自动切换到半开路状态(HALF-OPEN)。这时会判断下一次请求的返回情况, 如果请求成功, 断路器切回闭路状态(CLOSED), 否则重新切换到开路状态(OPEN)

◇**隔离策略**

信号量隔离对每个业务接口（command）创建一个自己的计数器，限制总的并发数，服务还是主线程进行同步调用

线程池隔离对每个业务接口（command）创建一个自己的线程池。就算业务接口出现了问题，也不会影响其他业务

◇**配置**

hystrix:

command:

default:

execution:

isolation:

semaphore:

maxConcurrentRequests: 300 # 如果是信号量隔离，可以设置最大并发请求数

thread:

timeoutInMilliseconds: 30000 # 调用方配置，被该调用方的所有方法的超时时间都是该值，优先级低于下边的指定配置

threadpool:

default:

coreSize: 8

maxQueueSize: 200

queueSizeRejectionThreshold: 1

◇**Zuul**

•**功能**

过滤

动态路由

负载分配

•**配置**

略

•**原理**

Zuul提供了一个框架，可以对过滤器进行动态的加载，编译，运行。

Zuul的过滤器之间通过一个RequestContext的静态类来进行数据传递的。RequestContext类 中有ThreadLocal变量来记录每个Request所需要传递的数据。

•**方法**

filterType

filterOrder

shouldFilter

run

•**四种标准过滤器类型**

PRE：这种过滤器在请求被路由之前调用。我们可利用这种过滤器实现身份验证、在集群中选择 请求的微服务、记录调试信息等。

ROUTING 务：这种过滤器用于构建发送给微服务的请求，并使用Apache HttpClient或Netfilx Ribbon请求微服务。

POST：这种过滤器在路由到微服务以后执行。这种过滤器可用来为响应添加标准的HTTP Header、 收集统计信息和指标、将响应从微服务发送给客户端等。

ERROR：在其他阶段发生错误时执行该过滤器。

◇**Gateway**

•**配置**

略

•**Gateway如何做服务限流**

使用redis的RateLimter限流算法来实现

•**GlobalFilter**

Filter

◇**Ribbon**

•**几种负载均衡算法**

RoundRobinRule：轮询策略

RandomRule: 随机策略

RetryRule: 重试策略

•**配置**

略

◇**多实例的情况，重启其中一个服务的正确步骤**

服务进行执行指令pause操作（相反操作resume），然后等待数秒，目的是为了请求进行平滑的过度其他服务，最后重启服务

◇**一个http请求，如果feign和ribbon都配置了重试机制，异常情况下一共会请求多少次**

请求总次数 n 为feignClient和ribbon配置参数的笛卡尔积：

n(请求总次数)=feign(默认5次) \* (MaxAutoRetries+1) \* (MaxAutoRetriesNextServer+1) 注意： +1是代表ribbon本身默认的请求。

# ▶**Eureka和Zookeeper的区别**

C - Consistent-一致性           A - Availability-可用性      P - Partition tolerance -分区容错性

**Eureka --> AP**

**Zookeeper --> CP**

Zookeeper当master挂了，会在30-120s进行leader选举，这点类似于redis的哨兵机制，在选举期间Zookeeper是不可用的，这么长时间不能进行服务注册，是无法忍受的，别说30s，5s都不能忍受。这时Zookeeper集群会瘫痪，这也是Zookeeper的CP，保持节点的一致性，牺牲了A/高可用。而Eureka不会，即使Eureka有部分挂掉，还有其他节点可以使用的，他们保持平级的关系，只不过信息有可能不一致，这就是AP，牺牲了C/一致性。

▶**Spring Boot 的核心注解是哪个？它主要由哪几个注解组成的？**

启动类上面的注解是@SpringBootApplication，它也是 Spring Boot 的核心注解，主要 组合包含了以下 3 个注解：

• @SpringBootConfiguration：组合了 @Configuration 注解，实现配置文件的功能。

• @EnableAutoConfiguration：打开自动配置的功能，也可以关闭某个自动配置的选项， 如关闭数据源自动配置功能： @SpringBootApplication(exclude = { DataSourceAutoConfiguration.class })。

• @ComponentScan：Spring 组件扫描。

▶**运行 Spring Boot 有哪几种方式？**

1）打包用命令或者放到容器中运行

2）用 Maven/ Gradle 插件运行

3）直接执行 main 方法运行