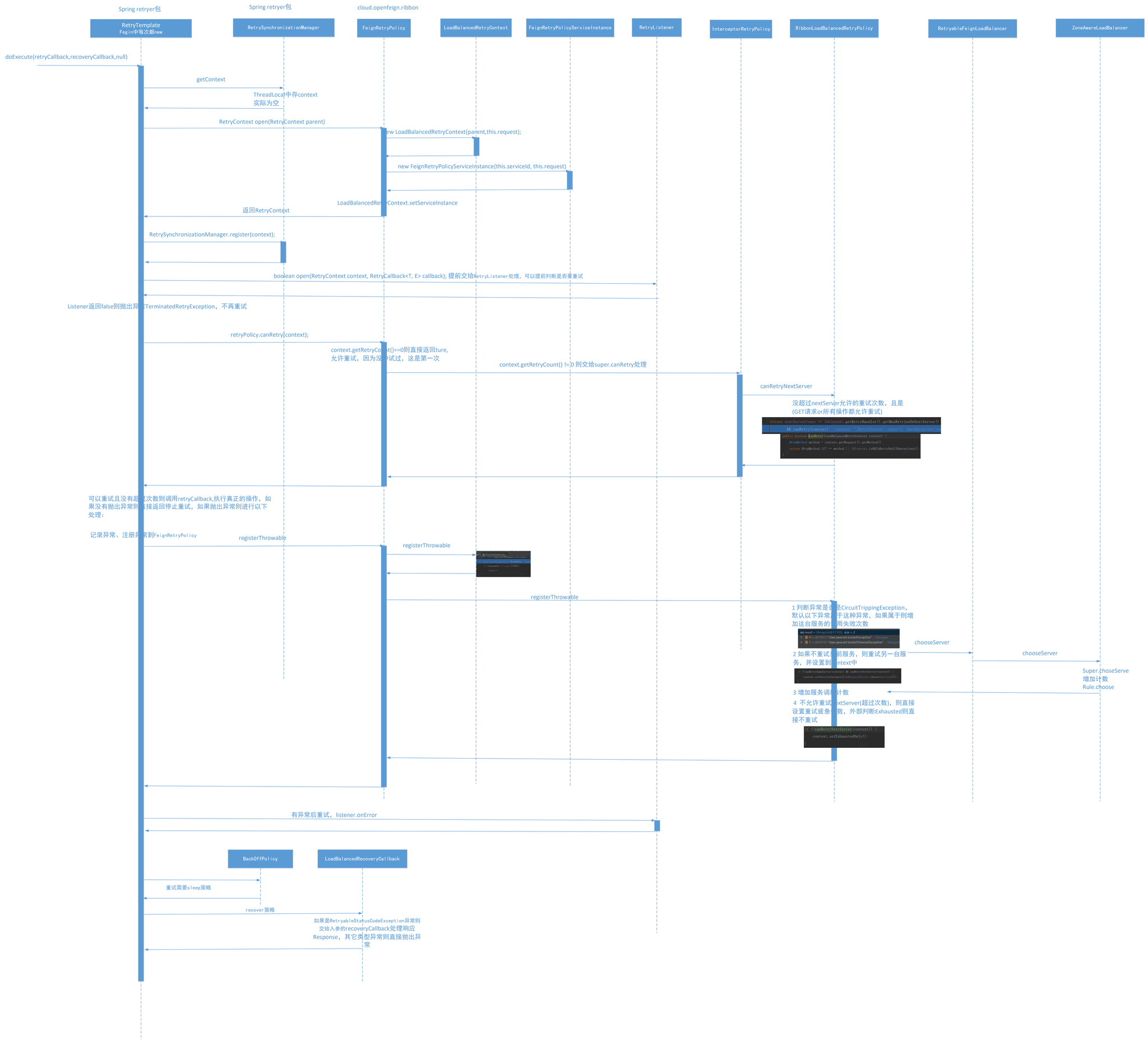
Spring retryer包
RetryTemplate
Fegin中每次都new
Spring retryer包
RetrySynchronizationManager
cloud.openfeign.ribbon
FeignRetryPolicy
LoadBalancedRetryContext
FeignRetryPolicyServiceInstance
RetryListener
InterceptorRetryPolicy
RibbonLoadBalancedRetryPolicy
RetryableFeignLoadBalancer
ZoneAwareLoadBalancer
doExecute(retryCallback,recoveryCallback,null)
getContext
ThreadLocal中存context
实际为空
RetryContext open(RetryContext parent)
new LoadBalancedRetryContext(parent,this.request);
new FeignRetryPolicyServiceInstance(this.serviceId, this.request)
LoadBalancedRetryContext.setServiceInstance
返回RetryContext
RetrySynchronizationManager.register(context);
boolean open(RetryContext context, RetryCallback<T, E> callback); 提前交给RetryListener处理，可以提前判断是否要重试
Listener返回false则抛出异常TerminatedRetryException，不再重试
retryPolicy.canRetry(context);
context.getRetryCount()==0则直接返回ture,
允许重试，因为没重试过，这是第一次
context.getRetryCount() !=0 则交给super.canRetry处理
canRetryNextServer
没超过nextServer允许的重试次数，且是
(GET请求or所有操作都允许重试)
return nextServerCount <= lbContext.getRetryHandler().getMaxRetriesOnNextServer()
&& canRetry(context) context: "AbstractRetryContext: count=0, lastBackOffMs=0"
public boolean canRetry(LoadBalancedRetryContext context) {
HttpMethod method = context.getRequest().getMethod();
return HttpMethod.GET == method || lbContext.isOkToRetryOnAllOperations();
}
可以重试且没有超过次数则调用retryCallback,执行真正的操作，如
果没有抛出异常则直接返回停止重试，如果抛出异常则进行以下
处理：
记录异常、注册异常到FeignRetryPolicy
registerTerminalThrowable(Throwable t);
if (t instanceof Error)
throw (Error) t;
if (Throwable.fatal(t))
return;
throw (RuntimeException) t;
registerThrowable
registerThrowable
registerThrowable
1 判断异常是否是CircuitTrippingException，
默认以下异常属于这种异常，如果属于则增
加这台服务的熔断失败次数
result = {ArrayList@11105} size = 2
0 = {@9920} "class java.net.SocketException" , NavigableSet
1 = {@9921} "class java.net.SocketTimeoutException" , NavigableSet
chooseServer
chooseServer
2 如果不重试当前服务，则重试另一台服
务，并设置到context中
if (loadBalancerSameServer(context) && canRetryNextServer(context)) {
context.setServiceInstance(fireRuleOnServer(server, chooseServiceId));
}
Super.choseServe
增加计数
Rule.choose
3 增加服务调用计数
4 不允许重试nextServer(超过次数)，则直接
设置重试疲惫标记，外部判断Exhausted则直
接不重试
if (!canRetryNextServer(context)) {
context.setExhaustedOnly();
}
有异常后重试，listener.onError
BackOffPolicy
LoadBalancedRecoveryCallback
重试需要sleep策略
recover策略
如果是RetryableStatusCodeException异常则
交给入参的recoveryCallback处理响应
Response，其它类型异常则直接抛出异
常