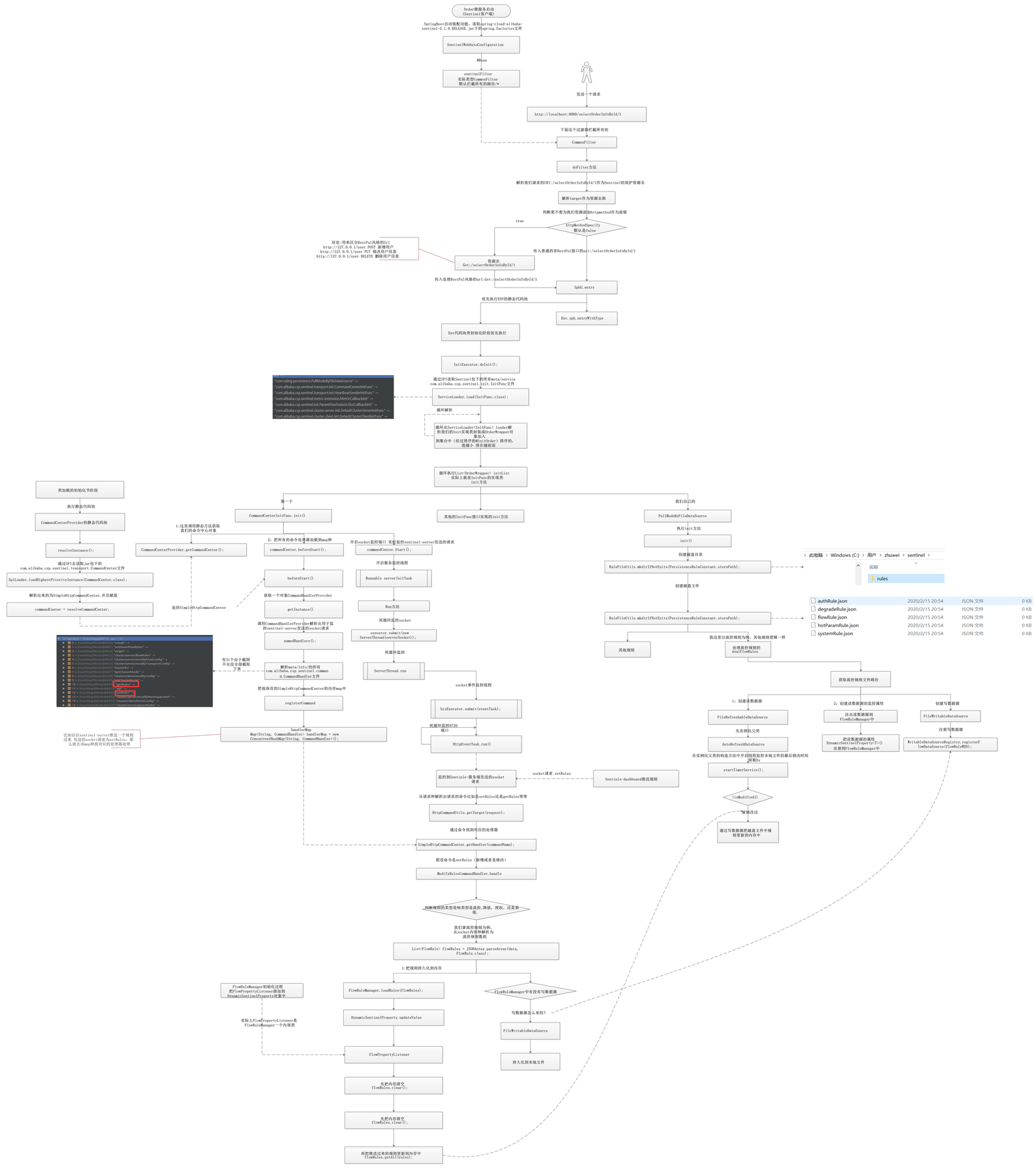
Order微服务启动
(Sentinel客户端)
SpringBoot自动装配功能，读取spring-cloud-alibaba-sentinel-2.1.0.RELEASE.jar下的spring.factories文件
SentinelWebAutoConfiguration
@Bean
sentinelFilter
实际类型CommonFilter
默认拦截所有的路径/*
发送一个请求
http://localhost:8080/selectOrderInfoById/1
下面这个过滤器拦截所有的
CommonFilter
doFilter方法
解析我们请求的URI:/selectOrderInfoById/1作为Sentinel的保护资源名
解析target作为资源名称
判断要不要为我们资源前加http method作为前缀
true
httpMethodSpecify
默认是false
好处:用来区分RestFul风格的Url
http://127.0.0.1/user POST 新增用户
http://127.0.0.1/user PUT 修改用户信息
http://127.0.0.1/user DELETE 删除用户信息
资源名
Get:/selectOrderInfoId/1
传入普通的非RestFul接口的url:/selectOrderInfoById/1
传入处理RestFul风格的url:Get:/selectOrderInfoId/1
SphU.entry
优先执行ENV的静态代码块
Env.sph.entryWithType
Env代码块类初始化阶段优先执行
InitExecutor.doInit();
通过SPI读取Sentinel包下的所有meta/service
com.alibaba.csp.sentinel.init.InitFunc文件
"com.tuling.persistence.PullModeByFileDataSource" ->
"com.alibaba.csp.sentinel.transport.init.CommandCenterInitFunc" ->
"com.alibaba.csp.sentinel.transport.init.HeartBeatSenderInitFunc" ->
"com.alibaba.csp.sentinel.metric.extension.MetricCallbackInit" ->
"com.alibaba.csp.sentinel.init.ParamFlowStatisticSlotCallbackInit" ->
"com.alibaba.csp.sentinel.cluster.server.init.DefaultClusterServerInitFunc" ->
"com.alibaba.csp.sentinel.cluster.client.init.DefaultClusterClientInitFunc" ->
ServiceLoader.load(InitFunc.class);
循环解析
循环从ServiceLoader<InitFunc> loader解析我们的Init实现类封装成OrderWrapper对象加入
到集合中（经过排序的和initOrder）排序的，数越小，排在最前面
循环执行List<OrderWrapper> initList
实际上就是InitFunc的实现类
init方法
第一个
CommonCenterInitFunc.init()
其他的InitFunc接口实现的init方法
我们自己的
PullModeByFileDataSource
执行init方法
init()
类加载的初始化节阶段
执行静态代码块
CommonCenterProvider的静态代码块
resolveInstance();
通过SPI去读取jar包下的
com.alibaba.csp.sentinel.transport.CommandCenter文件
SpiLoader.loadHighestPriorityInstance(CommandCenter.class);
解析出来的为SimpleHttpCommandCenter,并且赋值
commandCenter = resolveCommandCenter;
1:这里调用静态方法获取
我们的命令中心对象
CommandCenterProvider.getCommandCenter();
返回SimpleHttpCommandCenter
2、把所有的命令处理器加载到map种
commandCenter.beforeStart();
beforeStart()
获取一个对象CommonHandlerProvider
getInstance()
调用CommonHandlerProvider解析出用于监
控sentinel-server发送的socket请求
nameOHandlers();
解析meta/info/的所有
com.alibaba.csp.sentinel.comman
d.CommandHandler文件
有31个由于截图
不安投全部截取
了来
把他保存的SimpleHttpCommanCenter的内存map中
registerCommand
handlerMap
Map<String, CommandHandler> handlerMap = new
ConcurrentHashMap<String, CommandHandler>();
比如以后sentinel-server推送一个规则
过来 发送的socket请求为setRules，那
么就去该map种找对应的处理器处理
开启socket监控端口 实时监控sentinel-server发送的请求
commandCenter.Start();
开启服务端控线程
Runnable serverInitTask
Run方法
死循环监控socket
executor.submit(new
ServerThread(serverSocket));
死循环监控
ServerThread.run
socket事件监控线程
bizExecutor.submit(eventTask);
死循环监控8720
端口
HttpEventTask.run()
监控到Sentinle-服务端发送的socket
请求
socket请求 setRules
Sentinle-dashboard推送规则
从请求种解析出请求的命令比如是setRules还是getRules等等
HttpCommandUtils.getTarget(request);
通过命令找到对应的处理器
SimpleHttpCommandCenter.getHandler(commandName);
假设命令是setRules（新增或者是修改）
ModifyRulesCommandHandler.handle
判断规则的类型是哪类型是流控,降级，授权，还是熔
断.
我们拿流控规则为例，
从socket内容种解析为
流控规则数组
List<FlowRule> flowRules = JSONArray.parseArray(data,
FlowRule.class);
1:把规则持久化到内存
FlowRuleManager初始化过程
把FlowPropertyListener添加到
DynamicSentinelProperty对象中
实际上FlowPropertyListener是
FlowRuleManager一个内部类
FlowRuleManager.loadRules(flowRules);
DynamicSentinelProperty.updateValue
FlowPropertyListener
先把内存清空
flowRules.clear();
先把内存清空
flowRules.clear();
再把推送过来的规则更新到内存中
flowRules.putAll(rules);
FlowRuleManager中有没有写数据源
写数据源怎么来的?
FileWritableDataSource
持久化到本地文件
RuleFileUtils.mkdirIfNotExits(PersistenceRuleConstant.storePath);
创建磁盘目录
此电脑 › Windows (C:) › 用户 › zhuwei › sentinel ›
名称
rules
创建磁盘文件
RuleFileUtils.mkdirIfNotExits(PersistenceRuleConstant.storePath);
authRule.json 2020/2/15 20:54 JSON 文件 0 KB
degradeRule.json 2020/2/15 20:54 JSON 文件 0 KB
flowRule.json 2020/2/15 20:54 JSON 文件 0 KB
hotParamRule.json 2020/2/15 20:54 JSON 文件 0 KB
systemRule.json 2020/2/15 20:54 JSON 文件 0 KB
其他规则
我这里以流控规则为例，其他规则逻辑一样
处理流控规则的
dealFlowRules
获取流控规则文件路径
1、创建读数据源
2、创建读数据源的监控属性
创建写数据源
FileRefreshableDataSource
往出读数据源到
FlowRuleManager中
FileWritableDataSource
实实例化父类
AutoRefreshDataSource
在实例化父类的构造方法中开启线程监控本地文件的最后修改时间
更新ts
把读数据源的属性
DynamicSentinelProperty.T:t()
注册到FlowRuleManager中
注册写数据源
WritableDataSourceRegistry.registerF
lowDataSource(flowRuleWDS);
startTimerService();
lisModified()
"被修改过
通过写数据源把磁盘文件中规
则更新到内存中