1 什么是 Sentinel

- Sentinel 是 2018 年， 阿里开源的一个轻量级流量控制组件
- 它以流量为切入点，从**限流、熔断降级、系统负载**等多个维度来保证服务端的稳定性
    - 限流(flowslot)：指定某个接口最大 QPS 或 最大并发线程数，当超过指定值时，会采用几种限流策略
        - 通过 FlowRule.controlBehavior 字段指定
        - 直接拒绝(0)
        - 预热(1)：当突然有大流量来时，会逐渐提高流量限制的阈值，最终才达到指定值。用来应对服务端初始化资源时和初始化资源后，它的流量负载能力不一样的场景
        - 流量规整(2)：使得流量按规定的 QPS 进入资源，其余的等待(有最大等待时间)
        - 预热 + 流量规整(3)
    - 熔断降级(degradeslot)：当某个资源的平均响应时间或异常比例超过指定值时拒绝请求
    - 系统负载：当应用的 QPS、并发线程数、CPU 使用率、平均响应时间等达到指定值时拒绝请求

---

2 Sentinel 核心概念简介

2.1 resource

- 资源是对受保护的某一方法或代码块的唯一标识，用 resourceName 来表示

2.2 context

- 用 context 来构成资源调用的链路

2.3 slot

- 它是 Sentinel 功能实现的核心，通过不同的 slot 来做不同维度的保护，如 flowSlot 做限流保护、degradeSlot 做熔断降级保护、paramSlot 做热点参数限流保护、systemSlot 做系统负载保护、statisticSlot 收集 Metrics 数据等

2.4 slotChain

- 由多个 slot 串联成的一个调用链，这样使得所有定义的 slot 按一定的顺序执行

2.5 slotChainBuilder

- 通过不同的 slotChainBuilder 来创建不同的 slotChain，默认为 DefaultSlotChainBuilder，可以通过 SPI 接口扩展自定义的 slotChainBuilder

2.6 Node

- 通过不同的 Node 收集 Metrics 数据，各个 Slot 就是使用 Node 中的数据来做保护判断的，如 QPS、并发线程数都被实时的收集在不同的 Node 中

---

3 Sentinel 扩展点

3.1 InitFunc

- 通过实现 InitFunc 接口，并且在 META-INF/services 目录下的 com.alibaba.csp.sentinel.init.InitFunc 文件中指定该实现类全名
- 当第一次资源调用的时候，就会通过 Env 类中的 static 块加载到，并且运行 init 方法
- 所以我们可以在项目中创建 InitFunc 实例，并指定
    - 如通过 InitFunc 实例来初始化配置数据源
    - 如 sentinel-parameter-flow-control 包中就通过此扩展点注册了 ParamFlowStatisticEntryCallback 和 ParamFlowStatisticExitCallback

3.2 StatisticSlotCallbackRegistry

- StatisticSlot 中会调用此类来获取所有的 SlotEntryCallback、SlotExitCallback 实例 的 onPass、onBlocked 、onExit 来处理其他的一些成功或阻塞后的一些逻辑
- 如 ParamFlowStatisticCallback 来处理热点参数的线程数信息
- 注册动作可以通过 InitFunc 方式来添加 Callback 实例

3.3 SlotChainBuilder

- 通过实现 SlotChainBuilder 接口，并且在 META-INF/services 目录下的 com.alibaba.csp.sentinel.slotchain.SlotChainBuilder 文件中指定该实现类全名
- 当第一次资源调用的时候，就会通过 SlotChainProvider 类加载此文件中指定的类，如果有多个实例，就会选取一个最为唯一的 SlotChainBuilder 类

- 所以我们可以在项目中创建 SlotChainBuilder 实例，并指定
    - 如我们需要自定义某种 slot，那就需要通过自定义 SlotChainBuilder 来将我们定义的 slot 加入到 slotChain 中
    - 如 sentinel-parameter-flow-control 包中就通过此扩展点实现了 HotParamSlotChainBuilder，将 ParmaFlowSlot 加入到 slotChain 中

3.4 MetricExtension

- 通过实现 MetricExtension 接口，并且在 META-INF/services 目录下的 com.alibaba.csp.sentinel.metric.extension.MetricExtension 文件中指定该实现类全名
- 当第一次资源调用的时候，就会通过 MetricExtensionProvider 类加载此文件中指定的类，并放入 metricExtensions 列表中
- 目前 在 MetricEntryCallback、MetricExitCallback 中就会调用所有的 MetricExtension 实例
- 作用：可以通过它对 statistic 做扩展

---

4 Slot 简介

4.1 NodeSelectorSlot

- 生成 contextName 对应的 **resource** 属性，并存入 map 中
    - 此 map 为 实例属性，又因为 **一个 slot 实例对应一个 resourceName**
    - 所以这个 map 存储的是**一个 resource**(value) 下，对应的多个 contextName(key)
    - 这样的话，一个 **resource 就会对应 多个 DefaultNode** 了！因为一个 resource 下的每一个 contextName 都会对应不同的 DefaultNode
    - 那怎么通过 resource 下所有的 statistics 呢？用 ClusterNode！
- 设置 context 的 curNode 属性为 新建的 DefaultNode(**resourceWrapper**,null)
- 往后传的 node 就是 DefaultNode(**resourceWrapper**,null) 了！
- Sentinel 通过 `NodeSelectorSlot` **建立不同资源间的调用的关系**，并且通过 `ClusterNodeBuilderSlot` 记录每个资源的实时统计信息。

4.2 ClusterBuilderSlot

- 生成 resourceName(key) 对应的 clusterNode 实例属性，并存入 map 中
- 设置 DefaultNode(**resourceWrapper**,null) 的 clusterNode 属性
- 如果 origin != “”，则设置 这个 clusterNode 对应的 originNodeMap 实例属性
    - 即将同一个 resource 中的 clusterNode 按 origin 细分为 多个 originNode
    - 为了使统计针对 origin 粒度的信息
    - 现在的粒度有：contextName(这个粒度可大可小),origin,resource

4.3 LogSlot

- 记录日志

4.4 StatisticSlot

- 记录不同粒度的 matrix

```
//这里 EntryType 起作用了
if (resourceWrapper.getEntryType() == EntryType.IN) {
    // Add count for global inbound entry node for global statistics.
    Constants.ENTRY_NODE.increaseThreadNum();//只要是 IN 的都会进入这个全局节点
    Constants.ENTRY_NODE.addPassRequest(count);
}
```

4.5 SystemSlot

- 系统规则保护：thread、rt、load、cpu。可以自定义设置阈值
- 则通过系统的状态，例如 load1 等，来**控制总的入口流量**

```
//在此 Slot 中，对 statisticSlot 统计的 ENTRY_NODE 信息进行了规则保护
Constants.ENTRY_NODE
```

4.6 AuthoritySlot

- 权限过滤：黑白名单，判断 origin 是否是被拒绝的 origin

4.7 FlowSlot

- FlowSlot 通过使用预设的规则，来判断正在访问的请求是否应该被阻塞
- 如果任意预设的规则被触发了，就会抛 FlowException 异常，用户可以通过捕获这个异常来处理他们想要的逻辑
- 每一个 FlowRule 主要由：grade、strategy 、path 组成
    - grade：0 表示线程指标，1 表示 QPS 指标。这两个指标都会被实时收集
    - stage：主要用来处理不断堆积的线程问题，当一个资源访问耗时的时候
    - 流量规整：使得每段时间只有固定的线程通过

4.8 DegradeSlot

- 处理降级规则

---

5 Sentinel 使用案例

5.1 原始接口使用

```
//其中 contextName对应调用链路入口名称，通过 contextName 形成一条调用链
public void foo() {
    Entry entry = null;
    try {
        //ContextUtil.enter("contextName","originName");//可以不用
        entry = SphU.entry("abc");//"abc" 为被保护资源的唯一标识
        // resource that need protection
    } catch (BlockException blockException) {
        //因为遍历的时候会抛 BlockException，所以在这里处理 blocked handle logic
        //降级、限流、系统保护等拒绝后的逻辑
    } catch (Throwable bizException) {
        // business exception
```

```
    } finally {
        // ensure finally be executed
        if (entry != null){
            entry.exit();//必须是同一线程释放，否则抛 ErrorEntryFreeException 异常
        }
        //ContextUtil.exit();
    }
}
//---------------------- 各个方法分析
//对 resourceName 资源做保护
//1、
//获取 context，如果之前没有设置，则会使用 sentinel_default_context 来生成 context
//这里有个要注意的地方，一个 contextName 可能就对应多个资源了！
//context = new Context(node, name); node 为 context 中的 entranceNode 属性

//2、
//生成 resourceName 对应的 slotChain，并存入 map 中
//即，同一个 resourceName 肯定会走同一条 slotChain!
//这个 map 的最大值为 6000，也就是说一个项目最后定义 6000 个 resourceName

//3、
//生成 resource、slotChain、context 对应的 Entry，最为返回值，为了处理 context 等信息

//4、
//遍历 slotChain，执行相应的方法(功能)
entry = SphU.entry("resourceName");

entry.exit();//清除 context 信息
ContextUtil.exit();//清除线程信息
```

5.2 注解使用

- value： 资源名
- entryType：IN：表示进入系统的流量，OUT：表示出口流量，即调用其他资源。systemSlot 只对 IN 类型的流量生效，目前没有其它 slot 使用这个属性
- blockHandler：被拒绝之后走的处理方法，对应原始接口的 BlockException 后的逻辑
- blockHandlerClass：定义 blockHandler 所在的类，如果定义了这个，则 blockHandler 对应的类必须是 static 方法（因为这样 反射的话就不用指定 类的实例了）
- blockHandler 方法的返回类型必须和资源的返回类型一样，参数类型必须必资源的参数多一个 BlockException 参数
- fallback：处理 blockHandler 不能处理的异常
- fallbackClass：同理 blockHandlerClass
- blockHandler 方法的返回类型必须和资源的返回类型一样，参数类型可以加一个 Throwable 参数，也可以不加
- defaultFallback：如果之前的 fallback 流程走不通，则走这个 defaultFallback 流程，只是一个备选方案，也对应 fallbackClass。方法参数最多加个 Throwable 参数
- exceptionsToTrace：非 BlockException 异常，不通过 handleFallback 处理的 异常
- exceptionsToIgnore：非 BlockException 异常，通过 handleFallback 处理的 异常，且异常会被跟踪

```
@SentinelResource(value = "hello")
public String hello(String name) {
    return "hello：" + name;
}
```

- 如果要用注解，需要**添加依赖**，并且要将 SentinelResourceAspect 类注入容器中以使得注解生效

```
@Configuration
public class SentinelAspect {
    @Bean
    public SentinelResourceAspect sentinelResourceAspect() {
        System.out.println("init SentinelResourceAspect");
        return new SentinelResourceAspect();
    }
}
--- 添加依赖
<dependency>
    <groupId>com.alibaba.csp</groupId>
    <artifactId>sentinel-annotation-aspectj</artifactId>
    <version>1.6.3</version>
</dependency>
```

参考

sentinel-core-1.6.3