Sentinel中有很多比较重要的概念，我们要了解一个框架，首先要对框架中重要的概念实体进行分析，本文我将跟大家一起来分析一下sentinel中非常重要的几个概念。

Resource

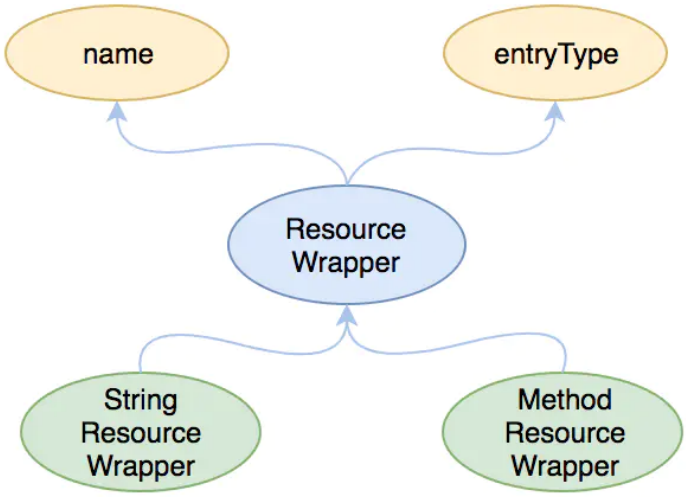
resource是sentinel中最重要的一个概念，sentinel通过资源来保护具体的业务代码或其他后方服务。sentinel把复杂的逻辑给屏蔽掉了，用户只需要为受保护的代码或服务定义一个资源，然后定义规则就可以了，剩下的通通交给sentinel来处理了。并且资源和规则是解耦的，规则甚至可以在运行时动态修改。定义完资源后，就可以通过在程序中埋点来保护你自己的服务了，埋点的方式有两种：

- try-catch 方式（通过 `SphU.entry(...)`），当 catch 到BlockException时执行异常处理(或fallback)
- if-else 方式（通过 `SphO.entry(...)`），当返回 false 时执行异常处理(或fallback)

以上这两种方式都是通过硬编码的形式定义资源然后进行资源埋点的，对业务代码的侵入太大，从0.1.1版本开始，sentinel加入了注解的支持，可以通过注解来定义资源，具体的注解为：`SentinelResource`。通过注解除了可以定义资源外，还可以指定 `blockHandler` 和 `fallback` 方法。

在sentinel中具体表示资源的类是：`ResourceWrapper`，他是一个抽象的包装类，包装了资源的**Name**和**EntryType**。他有两个实现类，分别是：`StringResourceWrapper` 和 `MethodResourceWrapper`

顾名思义，`StringResourceWrapper` 是通过对一串字符串进行包装，是一个通用的资源包装类，`MethodResourceWrapper` 是对方法调用的包装。



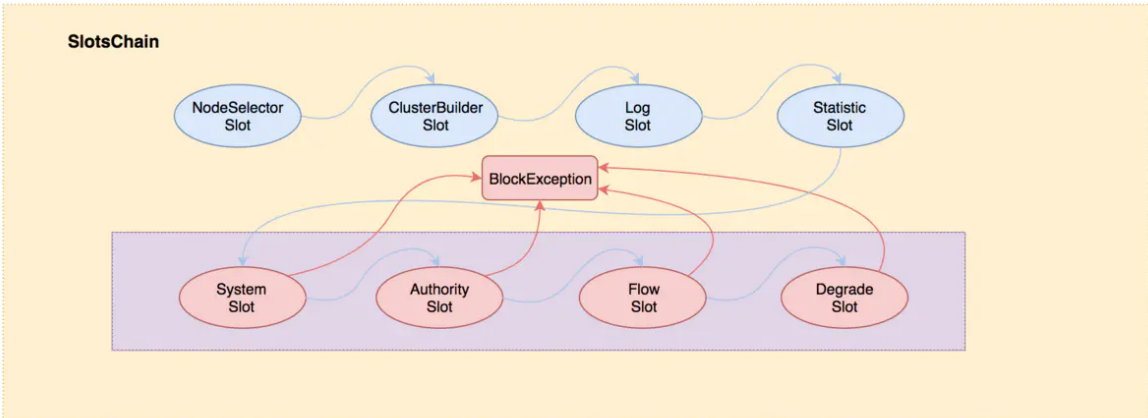
resource.png

Slot

slot是另一个sentinel中非常重要的概念，sentinel的工作流程就是围绕着一个个插槽所组成的插槽链来展开的。需要注意的是每个插槽都有自己的职责，他们各司其职完好的配合，通过一定的编排顺序，来达到最终的限流降级的目的。默认的各个插槽之间的顺序是固定的，因为有的插槽需要依赖其他的插槽计算出来的结果才能进行工作。

但是这并不意味着我们只能按照框架的定义来，sentinel 通过 `SlotChainBuilder` 作为 SPI 接口，使得 Slot Chain 具备了扩展的能力。我们可以通过实现 `SlotsChainBuilder` 接口加入自定义的 slot 并自定义编排各个 slot 之间的顺序，从而可以给 sentinel 添加自定义的功能。

那SlotChain是在哪创建的呢？是在 `CtSph.lookProcessChain()` 方法中创建的，并且该方法会根据当前请求的资源先去一个静态的HashMap中获取，如果获取不到才会创建，创建后会保存到HashMap中。这就意味着，同一个资源会全局共享一个SlotChain



slot-chain.png

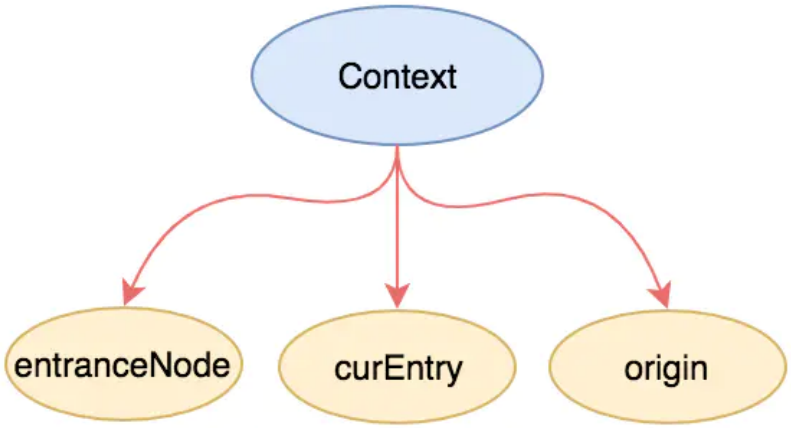
Context

context上下文是sentinel中一个比较难懂的概念。源码中是这样描述context类的：

> This class holds metadata of current invocation

就是说在context中维护着当前调用链的元数据，那元数据有哪些呢，从context类的源码中可以看到有：

- entranceNode：当前调用链的入口节点
- curEntry：当前调用链的当前entry
- node：与当前entry所对应的curNode
- origin：当前调用链的调用源



context.png

每次调用 `SphU.entry()` 或 `SphO.entry()` 都需要在一个 context 中执行，如果没有当前执行时还没有 context，那么框架会使用默认的 context，默认的 context 是通过 `MyContextUtil.myEnter()` 创建的。

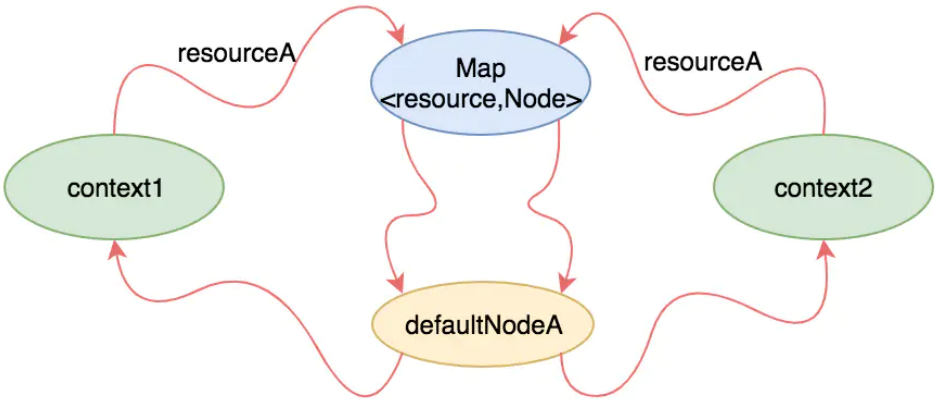
那如果我想自己在调用 `SphU.entry()` 或 `SphO.entry()` 前，自己创建一个context该怎么操作呢？那可以通过调用 `ContextUtil.enter()` 方法来创建。

另外context是保存在ThreadLocal中的，每次执行的时候会优先到ThreadLocal中获取。如果context为null时才会再次去创建一个context。

那什么时候context会被置为null并从ThreadLocal中清空呢？当Entry执行exit方法时，当当前entry的parent为null时，也就说明当前entry是最上层的节点了，此时要把保存在ThreadLocal中的context也清空掉。

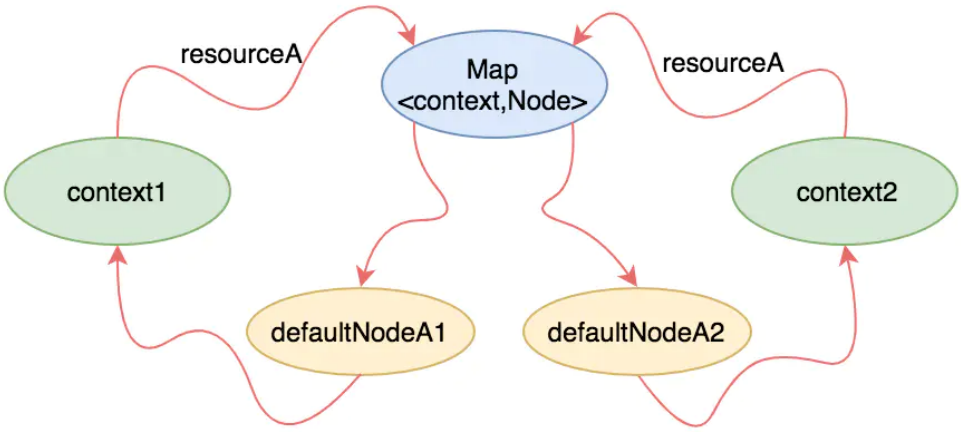
在NodeSelectorSlot类中有一个Map保存了DefaultNode，但是key是用的contextName，而不是resourceName，这是为什么呢?

试想一下，如果用resourceName来做map的key，那对于同一个资源resourceA来说，在context1中获取到的defaultNodeA和在context2中获取到的defaultNodeA是同一个，那么怎么在这两个context中对defaultNodeA进行更改呢，修改了一个必定会对另一个产生影响。



resource-node.png

而如果用contextName来作为key，那对于同一个资源resourceA来说，在context1中获取到的是defaultNodeA1，在context2中获取到是defaultNodeA2，那在不同的context中对同一个资源可以使用不同的DefaultNode进行分别统计和计算，最后再通过ClusterNode进行合并就可以了。



content-node.png

所以在NodeSelectorSlot这个类里面，map里面保存的是contextName和DefaultNode的映射关系，目的是为了可以在不同的context对相同的资源进行分开统计。

同一个context中对同一个resource进行多次entry()调用时，会形式一颗调用树，这个树是通过CtEntry之间的parent/child关系维护的。

> 具体的调用链的原理分析可以参考笔者的另一篇文章：限流降级神器-哨兵(sentinel)的资源调用链原理分析

Entry

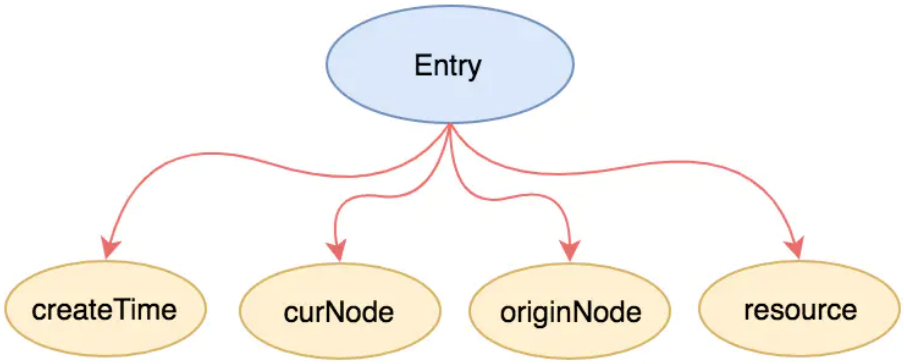
entry是sentinel中用来表示是否通过限流的一个凭证，就像一个token一样。每次执行 `SphU.entry()` 或 `SphO.entry()` 都会返回一个 `Entry` 给调用者，意思就是告诉调用者，如果正确返回了 `Entry` 给你，那表示你可以正常访问被sentinel保护的后方服务了，否则sentinel会抛出一个BlockException(如果是 `SphO.entry()` 会返回false)，这就表示调用者想要访问的服务被保护了，也就是说调用者本身被限流了。

entry中保存了本次执行 entry() 方法的一些基本信息，包括：

- createTime：当前Entry的创建时间，主要用来后期计算rt
- node：当前Entry所关联的node，该node主要是记录了当前context下该资源的统计信息
- origin：当前Entry的调用来源，通常是调用方的应用名称，在 `ClusterBuilderSlot.entry()` 方法中设置的
- resourceWrapper：当前Entry所关联的资源

当在一个上下文中多次调用了 `SphU.entry()` 方法时，就会创建一个调用树，这个树的节点之间是通过parent和child关系维持的。

需要注意的是：parent和child是在 `CtSph` 类的一个私有内部类 `CtEntry` 中定义的，`CtEntry` 是 `Entry` 的一个子类。 由于context中总是保存着调用链树中的当前入口，所以当当前entry执行exit退出时，需要将parent设置为当前入口。



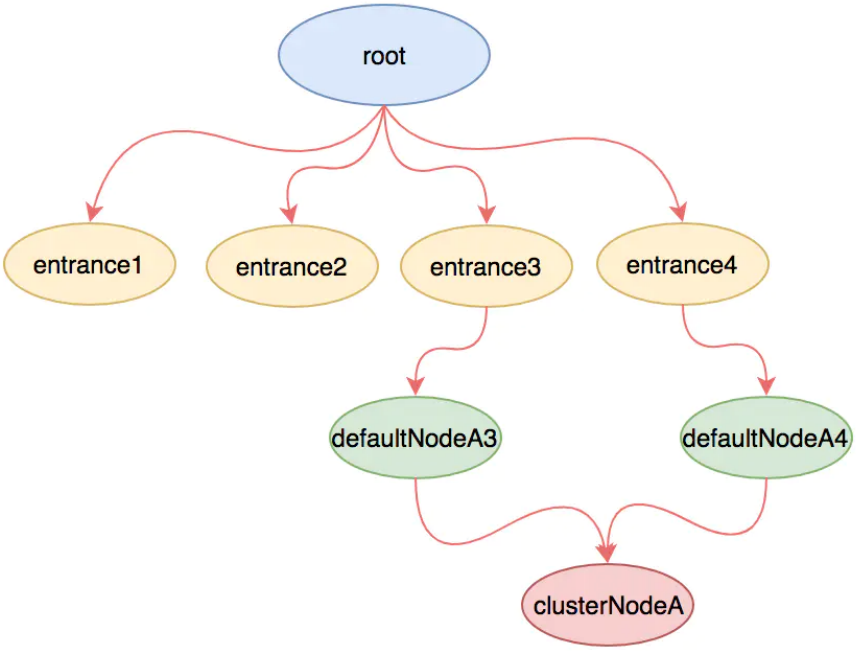
entry.png

Node

node中保存了资源的实时统计数据，例如：passQps，blockQps，rt等实时数据。正是有了这些统计数据后，sentinel才能进行限流、降级等一系列的操作。

node是一个接口，他有一个实现类：StatisticNode，但是StatisticNode本身也有两个子类，一个是DefaultNode，另一个是ClusterNode，DefaultNode又有一个子类叫EntranceNode。

其中entranceNode是每个上下文的入口，该节点是直接挂在root下的，是全局唯一的，每一个context都会对应一个entranceNode。另外defaultNode是记录当前调用的实时数据的，每个defaultNode都关联着一个资源和clusterNode，有着相同资源的defaultNode，他们关联着同一个clusterNode。



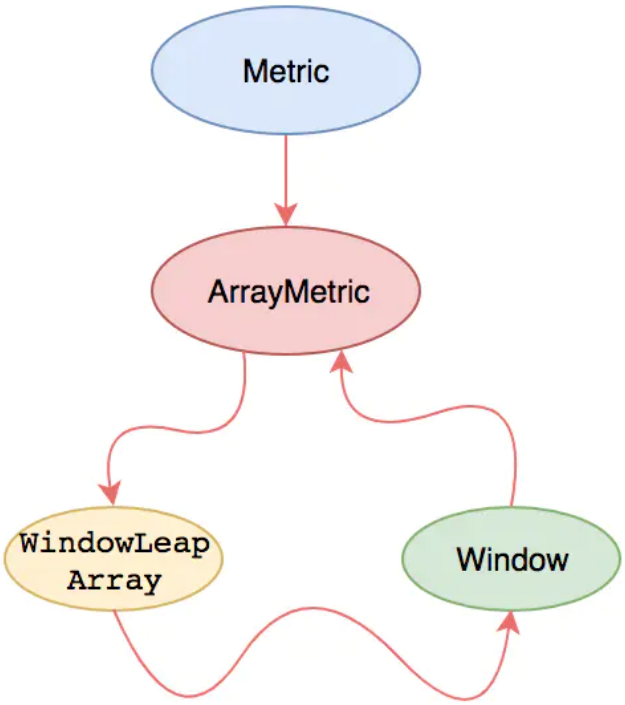
node.png

Metric

metric是sentinel中用来进行实时数据统计的度量接口，node就是通过metric来进行数据统计的。而metric本身也并没有统计的能力，他也是通过Window来进行统计的。

> 具体的统计原理，可以参考笔者另一篇文章：sentinel基于滑动时间窗口的实时指标统计原理分析

Metric有一个实现类：ArrayMetric，在ArrayMetric中主要是通过一个叫WindowLeapArray的对象进行窗口统计的。



metric.png