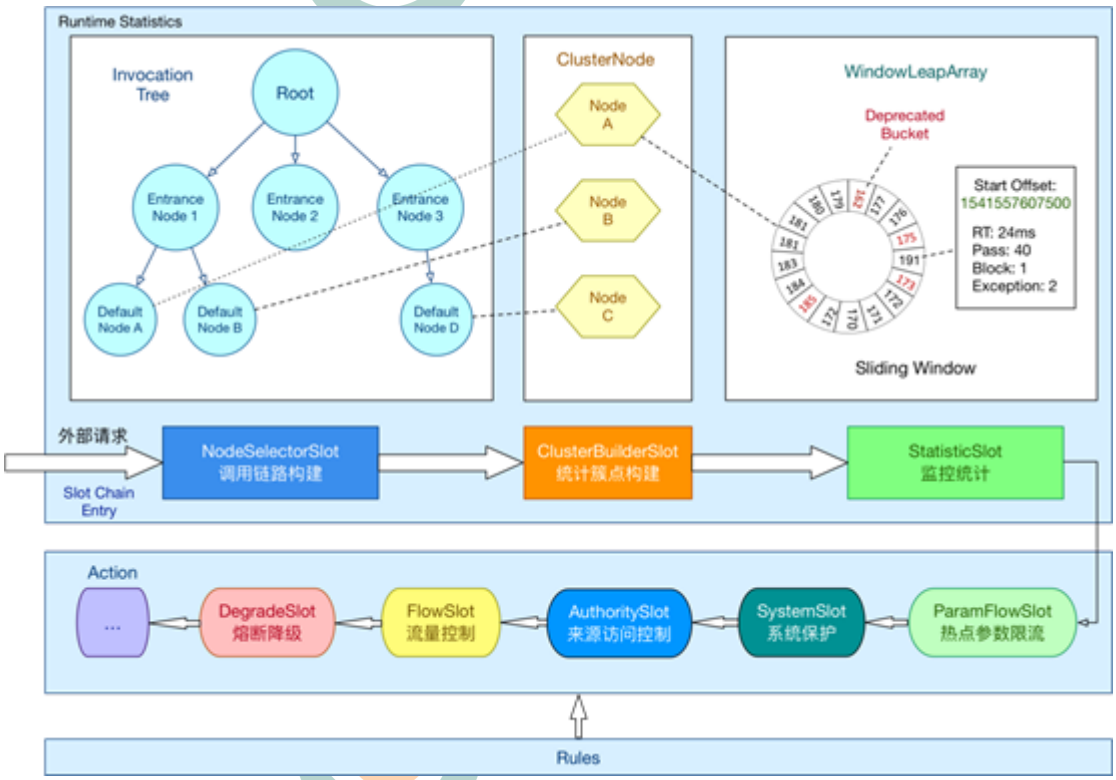
Sentinel核心源码解析

Sentinel是分布式系统的防御系统。以流量为切入点，通过动态设置的流量控制、服务熔断等手段达到保护系统的目的，通过服务降级增强服务被拒后用户的体验。

一、Sentinel工作原理

1 架构图解析

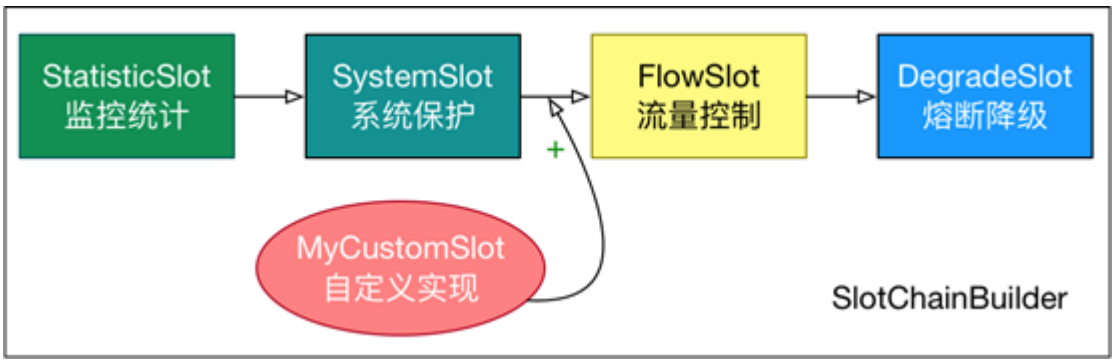
若要读懂Sentinel源码，则必须要搞明白官方给出的Sentinel的架构图。



Sentinel的核心骨架是ProcessorSlotChain。其将不同的 Slot 按照顺序串在一起（责任链模式），从而将不同的功能组合在一起（限流、降级、系统保护）。系统会为每个资源创建一套SlotChain。

2 SPI机制

Sentinel槽链中各Slot的执行顺序是固定好的。但并不是绝对不能改变的。Sentinel将ProcessorSlot 作为 SPI 接口进行扩展，使得 SlotChain 具备了扩展能力。用户可以自定义Slot并编排Slot 间的顺序。



3 Slot简介

NodeSelectorSlot

负责收集资源的路径，并将这些资源的调用路径，以树状结构存储起来，用于根据调用路径来限流降。

ClusterBuilderSlot

用于存储资源的统计信息以及调用者信息，例如该资源的 RT, QPS, thread count，Block count，Exception count 等等，这些信息将用作为多维度限流，降级的依据。简单来说，就是用于构建 ClusterNode。

StatisticSlot

用于记录、统计不同纬度的 runtime 指标监控信息。

ParamFlowSlot

对应 热点流控 。

FlowSlot

用于根据预设的限流规则以及前面 slot 统计的状态，来进行流量控制。对应 流控规则 。

AuthoritySlot

根据配置的黑白名单和调用来源信息，来做黑白名单控制。对应 授权规则 。

DegradeSlot

通过统计信息以及预设的规则，来做熔断降级。对应 降级规则 。

SystemSlot

通过系统的状态，例如 load1 等，来控制总的入口流量。对应 系统规则 。

4 Context简介

Context是对资源操作的上下文，每个资源操作必须属于一个Context。如果代码中没有指定Context，则会创建一个name为sentinel_default_context的默认Context。一个Context生命周期中可以包含多个资源操作。Context生命周期中的最后一个资源在exit()时会清理该Conetxt，这也就意味着这个Context生命周期结束了。

5 Context代码举例

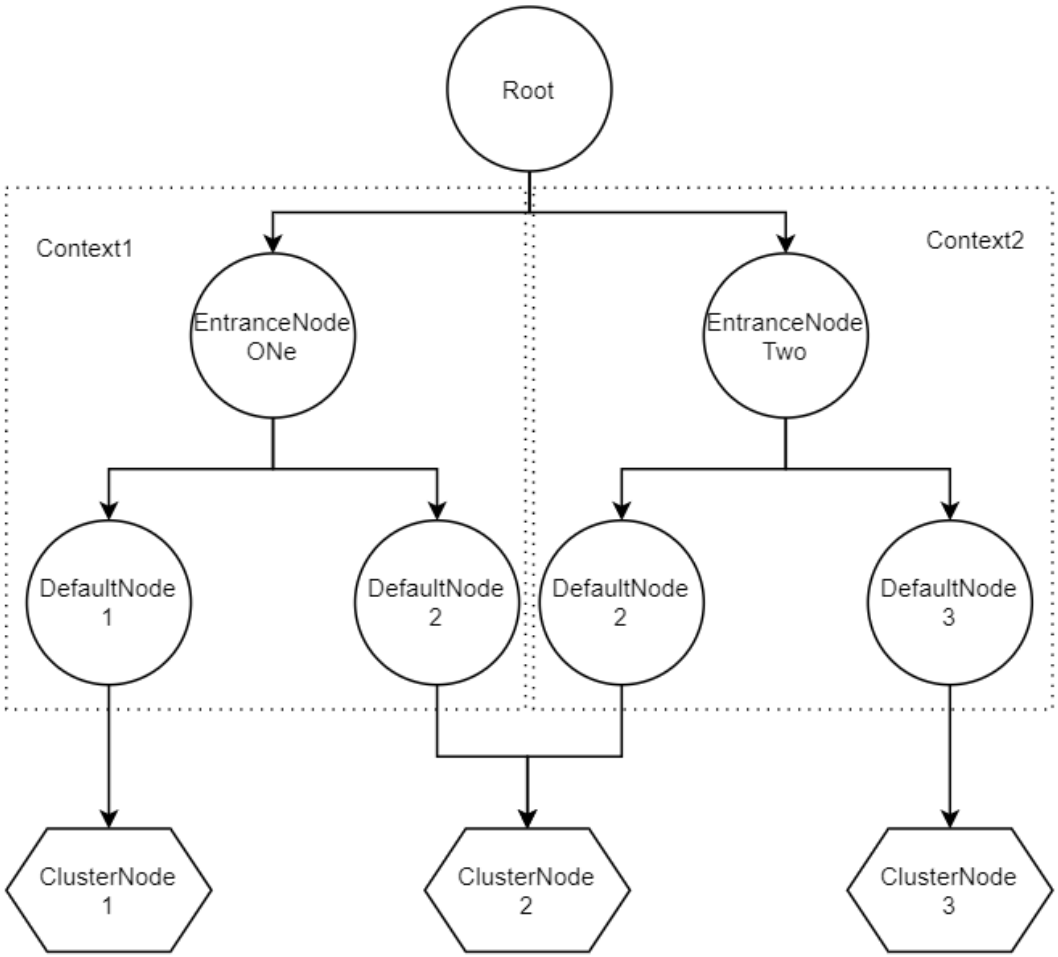
```
// 创建一个来自于appA访问的Context,
        // entranceOne为Context的name
        ContextUtil.enter("entranceOne", "appA");
        // Entry就是一个资源操作对象
        Entry resource1 = null;
        Entry resource2 = null;
        try {
            // 获取资源resource1的entry
            resource1 = SphU.entry("resource1");
            // 代码能走到这里，说明当前对资源resource1的请求通过了流控
            // 对资源resource1的相关业务处理。。。

            // 获取资源resource2的entry
            resource2 = SphU.entry("resource2");
            // 代码能走到这里，说明当前对资源resource2的请求通过了流控
            // 对资源resource2的相关业务处理。。。
        } catch (BlockException e) {
            // 代码能走到这里，说明请求被限流，
            // 这里执行降级处理
        } finally {
            if (resource1 != null) {
                resource1.exit();
            }
            if (resource2 != null) {
                resource2.exit();
            }
        }
        // 释放Context
        ContextUtil.exit();

        // ------------------------------------------------------
```

```
32
33          // 创建另一个来自于appA访问的Context,
34          // entranceTwo为Context的name
35          ContextUtil.enter("entranceTwo", "appA");
36          // Entry就是一个资源操作对象
37          Entry resource3 = null;
38          try {
39              // 获取资源resource2的entry
40              resource2 = SphU.entry("resource2");
41              // 代码能走到这里，说明当前对资源resource2的请求通过了流控
42              // 对资源resource2的相关业务处理。。。
43
44
45              // 获取资源resource3的entry
46              resource3 = SphU.entry("resource3");
47              // 代码能走到这里，说明当前对资源resource3的请求通过了流控
48              // 对资源resource3的相关业务处理。。。
49
50          } catch (BlockException e) {
51              // 代码能走到这里，说明请求被限流，
52              // 这里执行降级处理
53          } finally {
54              if (resource2 != null) {
55                  resource2.exit();
56              }
57              if (resource3 != null) {
58                  resource3.exit();
59              }
60          }
61          // 释放Context
62          ContextUtil.exit();
```

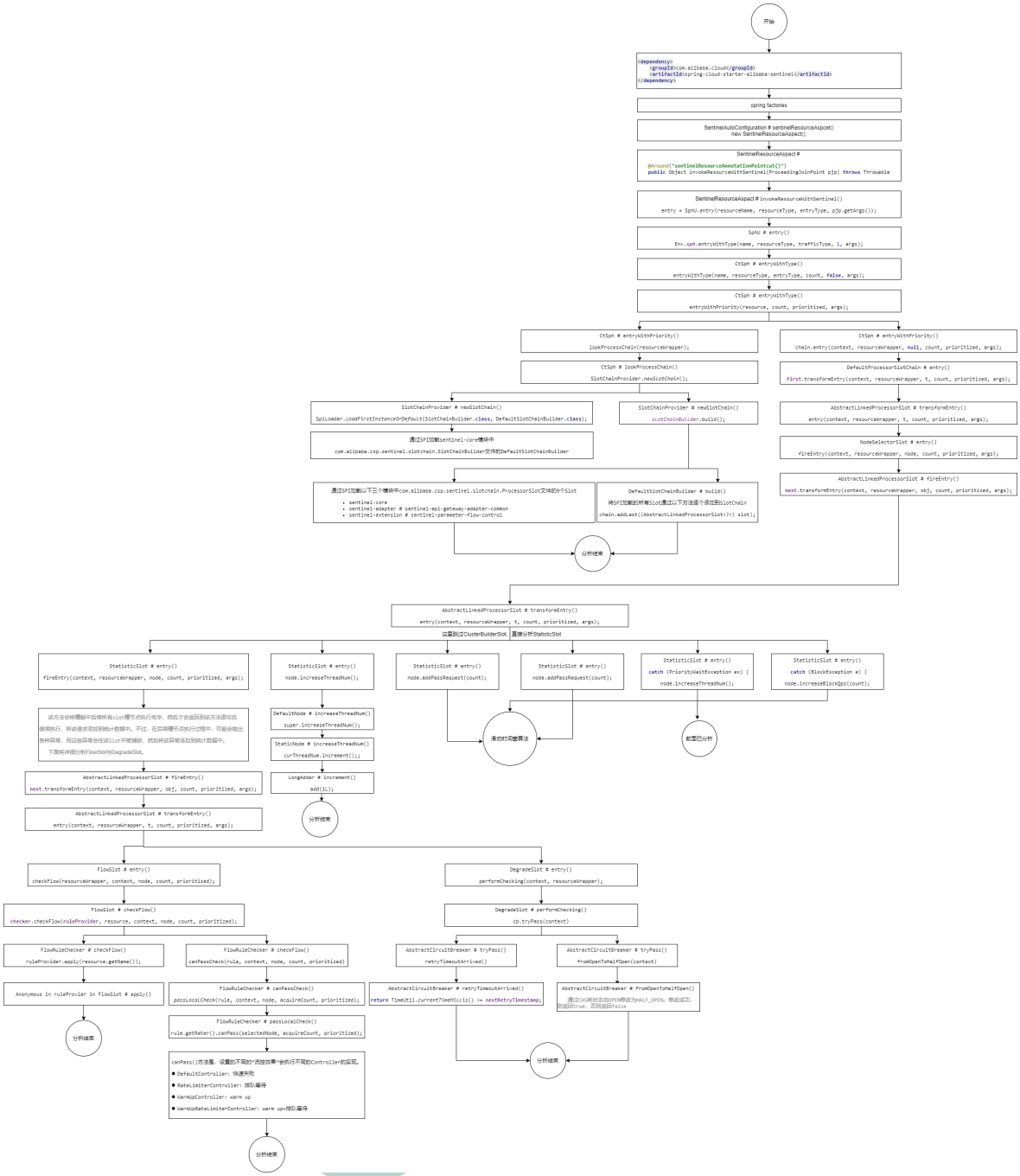
6 Node间的关系



Node间的关系

- Node：用于完成数据统计的接口
- StatisticNode：统计节点，是Node接口的实现类，用于完成数据统计
- EntranceNode：入口节点，一个Context会有一个入口节点，用于统计当前Context的总体流量数据
- DefaultNode：默认节点，用于统计一个资源在当前Context中的流量数据
- ClusterNode：集群节点，用于统计一个资源在所有Context中的总体流量数据

二、Sentinel核心源码解析



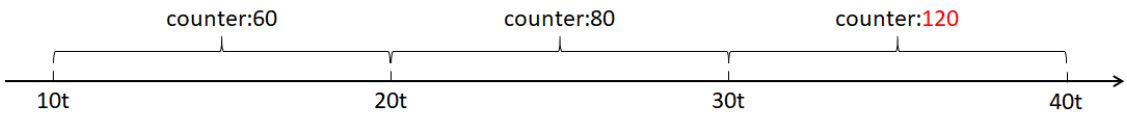
三、滑动时间窗算法

对于滑动时间窗算法的源码解析分为两部分：对数据的统计，与对统计数据的使用。不过，在分析源码之前，需要先理解该算法原理。

1 时间窗限流算法

算法原理

阈　值：100
时间窗：10t



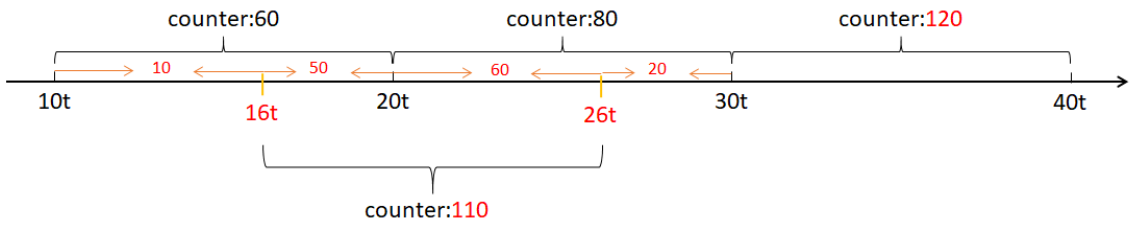
时间窗限流算法

该算法原理是，系统会自动选定一个时间窗口的起始零点，然后按照固定长度将时间轴划分为若干定长的时间窗口。所以该算法也称为“固定时间窗算法”。

当请求到达时，系统会查看该请求到达的时间点所在的时间窗口当前统计的数据是否超出了预先设定好的阈值。未超出，则请求通过，否则被限流。

存在的问题

阈　值：100
时间窗：10t

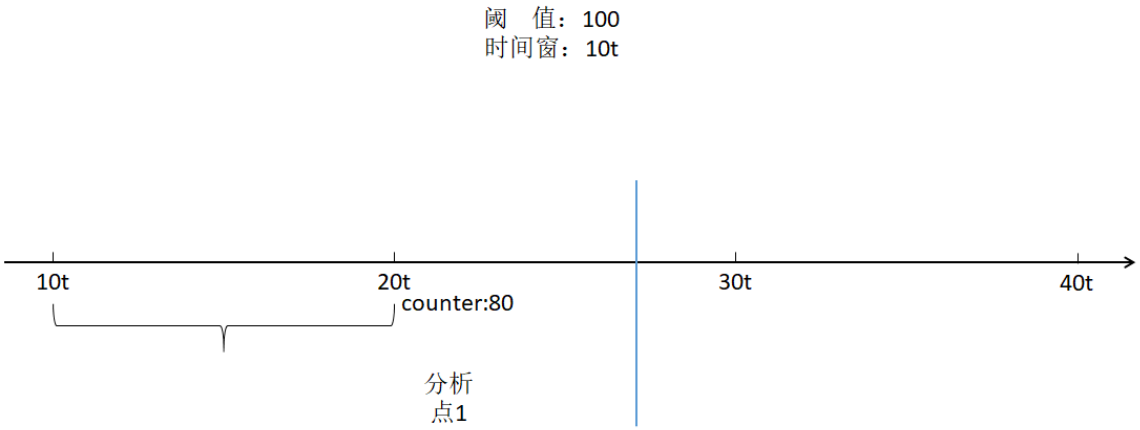


时间窗限流算法存在的问题

该算法存在这样的问题：连续两个时间窗口中的统计数据都没有超出阈值，但在跨窗口的时间窗长度范围内的统计数据却超出了阈值。

2 滑动时间窗限流算法

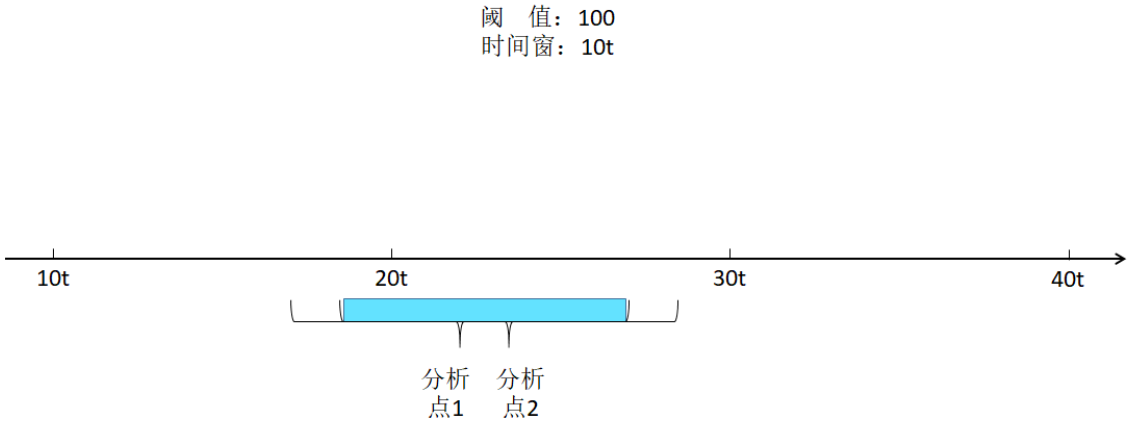
算法原理



滑动时间窗限流算法

滑动时间窗限流算法解决了固定时间窗限流算法的问题。其没有划分固定的时间窗起点与终点，而是将每一次请求的到来时间点作为统计时间窗的终点，起点则是终点向前推时间窗长度的时间点。这种时间窗称为“滑动时间窗”。

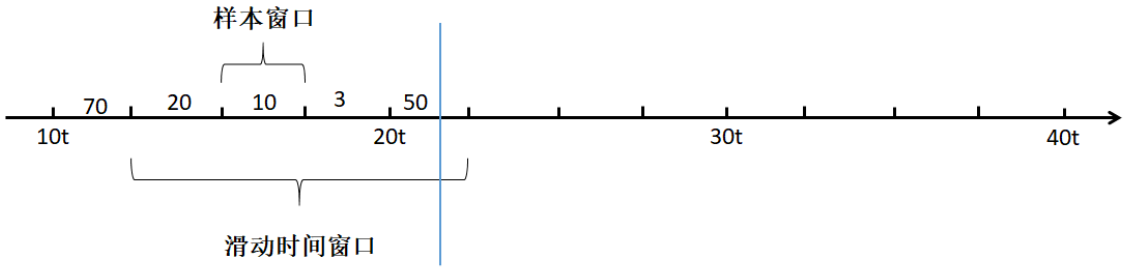
存在的问题



滑动时间窗限流算法

算法改进

阈　值：100
时间窗：10t
单位滑动窗口样本数：4
样本窗口长度：2.5t

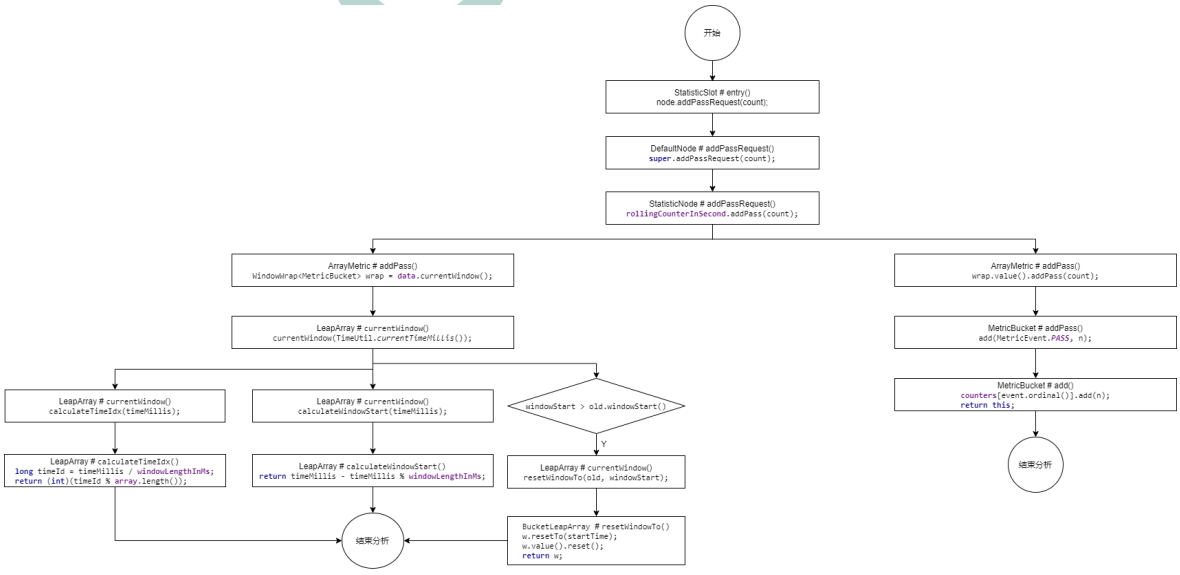


滑动时间窗限流算法

针对以上问题，系统采用了一种“折中”的改进措施：将整个时间轴拆分为若干“样本窗口”，样本窗口的长度是小于滑动时间窗口长度的。当等于滑动时间窗口长度时，就变为了“固定时间窗口算法”。 一般时间窗口长度会是样本窗口长度的整数倍。

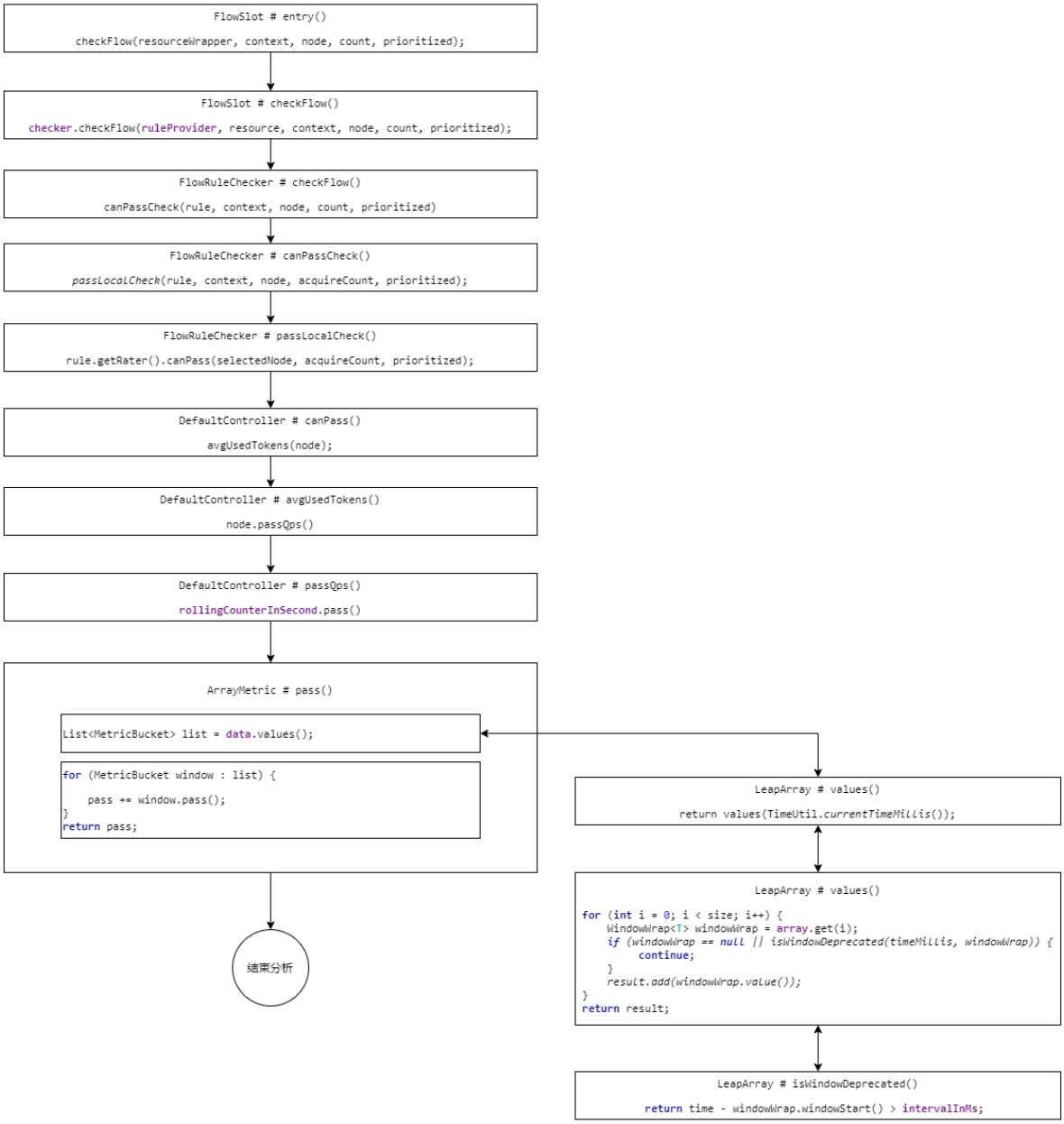
那么是如何判断一个请求是否能够通过呢？当到达样本窗口终点时间时，每个样本窗口会统计一次本样本窗口中的流量数据并记录下来。当一个请求到达时，会统计出当前请求时间点所在样本窗口中的流量数据，然后再获取到当前请求时间点所在时间窗中其它样本窗口的统计数据，求和后，如果没有超出阈值，则通过，否则被限流。

3 数据统计源码解析



Sentinel滑动时间窗算法源码解析—数据统计

4 使用统计数据



Sentinel滑动时间窗算法源码解析—使用统计数据