Redis 的这些拓展方案

Java后端 今天

收录于话题
#Redis

9个

Redis大家都不陌生，就算是没用过，也都听说过了。作为最广泛使用的**KV**内存数据库之一，在当今的**大流量时代**，单机模式略显单薄，免不了要有一些**拓展**的方案。

笔者下文会对各种方案进行介绍，并且给出场景，实现 等等概述，还会提到一些新手常见的误区。

正文

先从基础的拓展方式开始，这样更便于理解较高级的模式。

ps: 本文背景是以笔者落笔时~~官网最新稳定版~~***5.0.8***为准，虽然还没写完就变成了*6.0.1*。

分区

> 概述

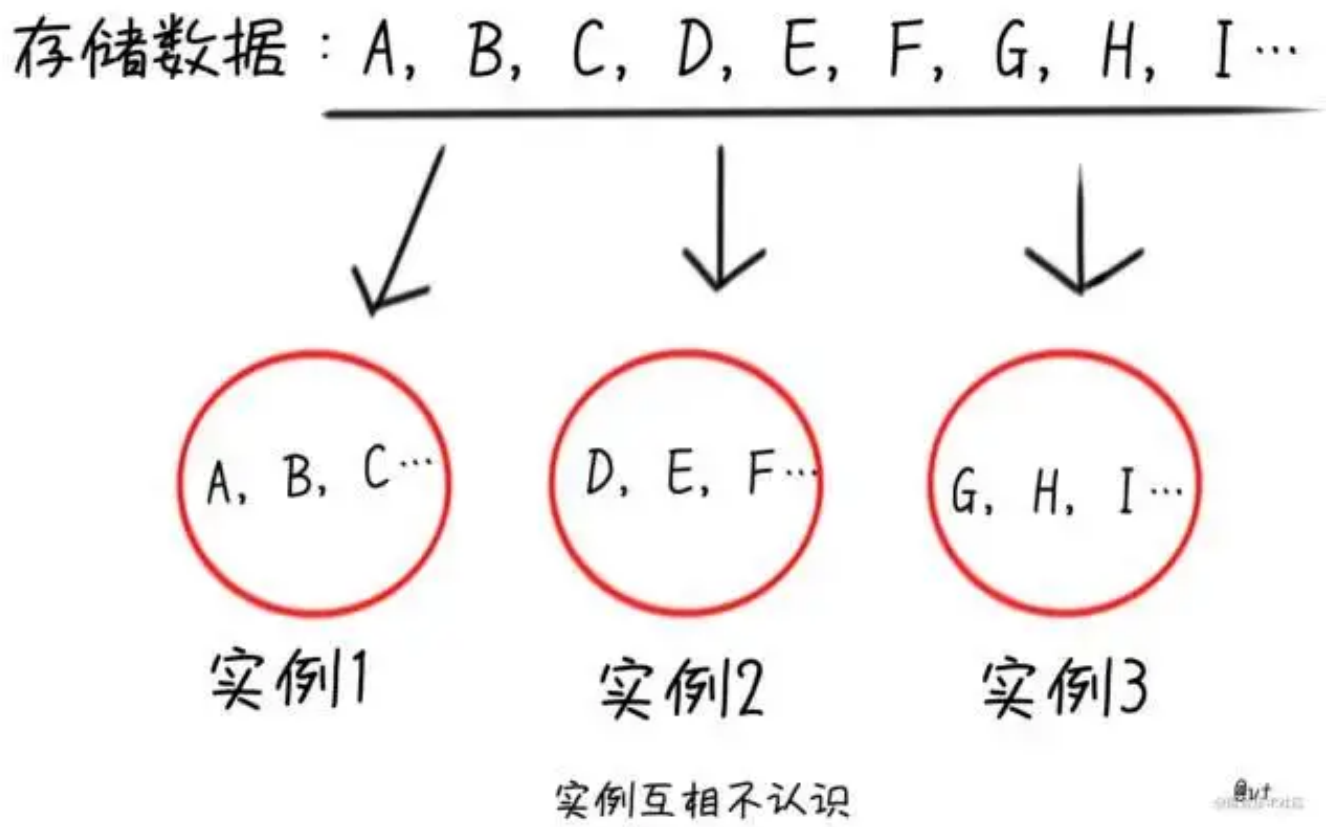
分区(Partitioning)是一种最为简单的拓展方式。

在我们面临**单机**的**存储空间**瓶颈时，第一点就能想到像传统的关系型数据库一样，进行数据分区。

或者假设手中有**N**台机器可以作为Redis服务器，所有机器内存总和有256G, 而客户端正好也需要一个大内存的存储空间。

我们除了可以把内存条都拆下来焊到一个机器上，也可以选择分区使用，这样又拓展了计算能力。

单指**分区**来讲，即将全部数据分散在多个Redis实例中，每个实例不需要关联，可以是完全独立的。



> 使用方式

客户端处理

和传统的数据库分库分表一样，可以从**key**入手，先进行计算，找到对应数据存储的实例在进行操作。**范围角度**，比如orderId:1~orderId:1000放入实例1，orderId:1001~orderId:2000放入实例2。

哈希计算，就像我们的**hashmap**一样，用hash函数加上位运算或者取模，高级玩法还有一致性Hash等操作，找到对应的实例进行操作

使用代理中间件

我们可以开发独立的代理中间件，屏蔽掉处理数据分片的逻辑，独立运行。当然也有他人已经造好的轮子，Redis也有优秀的代理中间件，譬如Twemproxy，或者codis，可以结合场景选择是否使用。

> 缺点

无缘多key操作，key都不一定在一个实例上，那么多key操作或者多key事务自然是不支持。

维护成本，由于每个实例在物理和逻辑上，都属于单独的一个节点，缺乏统一管理。

灵活性有限，范围分片还好，比如hash+MOD这种方式，如果想**动态**调整Redis实例的数量，就要考虑大量数据迁移，这就非常麻烦了。

同为开发者，深知我们虽然总能“曲线救国”的完成一些当前环境不支持的功能，但是总归要麻烦一些。

主从

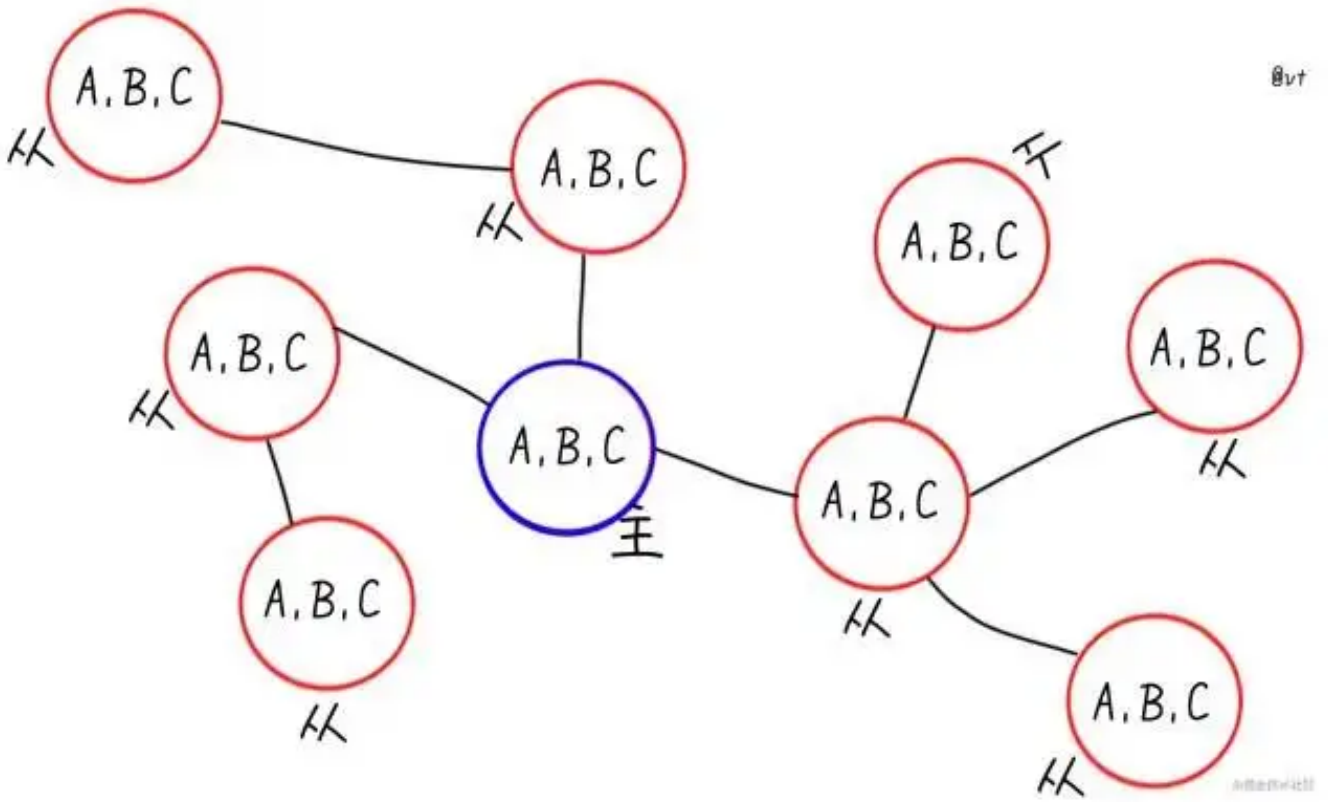
> 概述数据迁移

常说的**主从**(Master-Slave)，也就是复制(Replication)方式，怎么称呼都可以。

同上面的**分区**一样，也是Redis高可用架构的基础，新手可能会误以为这类基础模式即是“高可用”，这并不是十分正确的。

分区暂时能解决**单点**无法容纳的**数据量问题**，但是一个Key还是只在一个实例上，在大流量时代显得不那么可靠。

主从就是另一个纬度的拓展，节点将数据同步到**从**节点，就像将实例“**分身**”了一样，可靠性又提高了不少。



图画的有些夸张了，主要还是想体现**结构灵活**，是一主一从，还是一主多从，还是一主多从多从... 看你心情

有了“实例分身”，自然就可以做**读写分离**，将读流量均摊在各个从节点。

> 使用方式

高手云集的时代，聊天软件难免要备上这么一张表情包。

这表情包和**使用方式**有什么关系呢？首先看看使用方式：

1. 作为主节点的Redis实例，并不要求配置任何参数，只需要正常启动
2. 作为从节点的实例，使用配置文件或命令方式REPLICAOF 主节点Host 主节点port即可完成主从配置

是不是和表情包一样，“dalao”没动，我去“**抱大腿**”。这样一个主从最小配置就完成了，主从实例即可对外提供服务。命令里的“主节点”是相对的，slave也可以抱slave大腿，也就是上文提到的**结构灵活**。

> 缺点

slave节点都是**只读**的，如果**写流量**大的场景，就有些力不从心了。
那我把slave节点**只读**关掉不就行了？当然不行，数据复制是由主到从，从节点独有数据同步不到主节点，数据就不一致了。

故障转移不友好，主节点挂掉后，写处理就无处安放，需要**手工**的设定新的主节点，如使用REPLICAOF no one(谁大腿我都不抱了) 晋升为主节点，再梳理其他slave节点的新主配置，相对来说比较麻烦。

哨兵

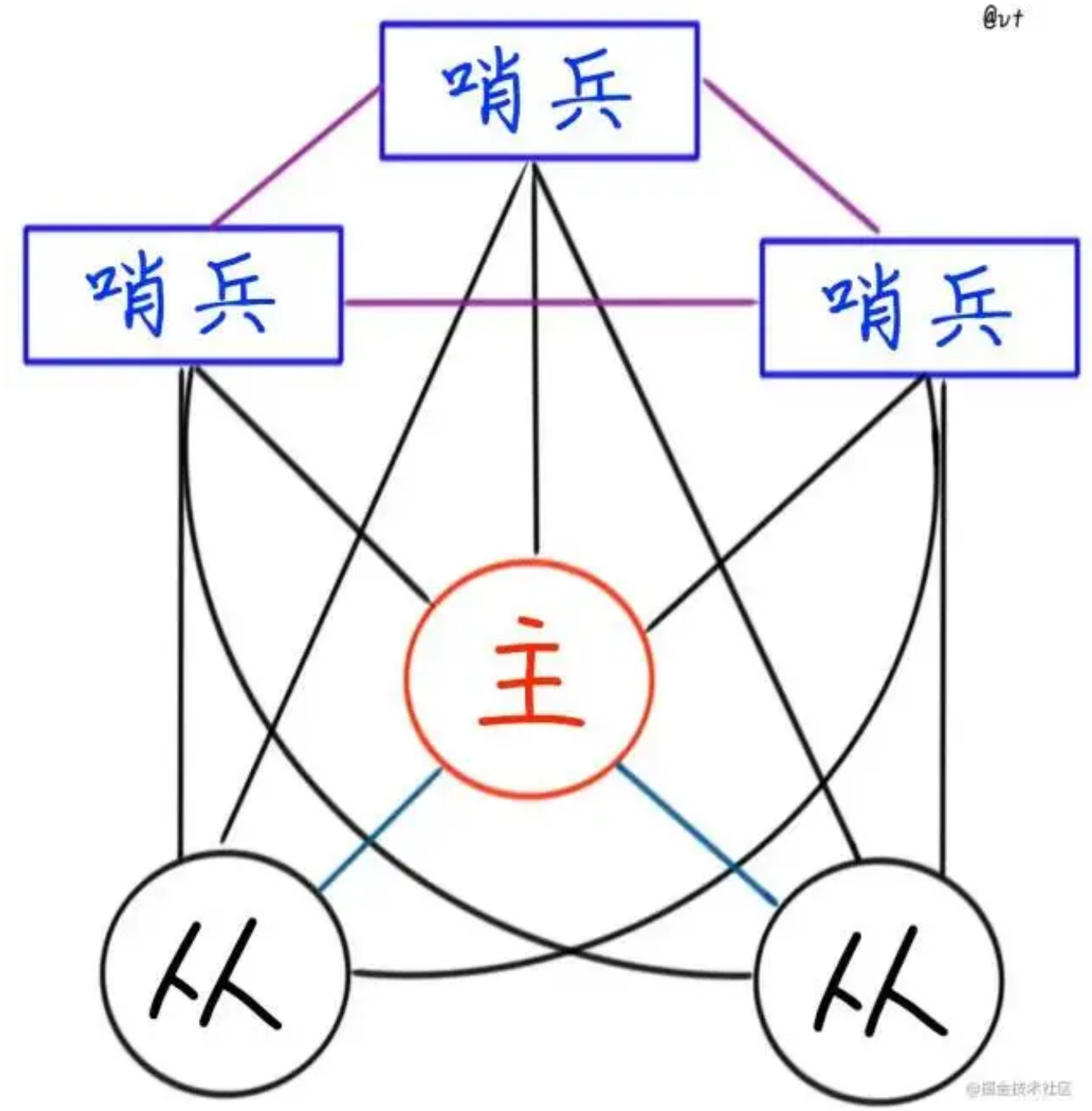
> 概述

主从的手工故障转移，肯定让人很难接受，自然就出现了高可用方案-**哨兵**（Sentinel）。

我们可以在主从架构不变的场景，直接加入**Redis Sentinel**，对节点进行**监控**，来完成自动的**故障发现**与**转移**。

并且还能够充当**配置提供者**，提供主节点的信息，就算发生了故障转移，也能提供正确的地址。

哨兵本身也是Redis实例的一种，但不作为数据存储方使用，启动命令也是不一样的。



虽然图有些复杂，看起来像要召唤光能使者。

其实实际使用起来是很便捷的。

> 使用方式

Sentinel的最小配置，一行即可：

```
sentinel monitor <主节点别名> <主节点host> <主节点端口> <票数>
```

只需要配置master即可，然后用redis-sentinel <配置文件> 命令即可启用。

Redis官网提到的“**最小配置**”是如下所示，除了上面提到的一行，还有其它的一些配置：

```
sentinel monitor mymaster 127.0.0.1 6379 2
sentinel down-after-milliseconds mymaster 60000
sentinel failover-timeout mymaster 180000
sentinel parallel-syncs mymaster 1

sentinel monitor resque 192.168.1.3 6380 4
sentinel down-after-milliseconds resque 10000
sentinel failover-timeout resque 180000
sentinel parallel-syncs resque 5
```

这是因为官网加了一个修饰词，是“**典型的最小配置**”，把重要参数和**多主**的例子都写出来了

，照顾大家CV大法的时候，不要忘记重要参数，其实都是有默认值的。

正如该例所示，设置**主节点别名**就是为了监控**多主**的时候，与其额外配置项能够与其对应，以及sentinel一些命令，如SENTINEL get-master-addr-by-name就要用到别名了。

哨兵数量建议在三个以上且为奇数，在Redis官网也提到了各种情况的“布阵”方式，非常值得参考。

> 更多

既然是高可用方案，并非有严格意义上的“缺点”，还需配合使用场景进行考量。

故障转移期间短暂的不可用，但其实官网的例子也给出了parallel-syncs参数来指定并行的同步实例数量，以免全部实例都在同步出现整体不可用的情况，相对来说要比手工的故障转移更加方便。

分区逻辑需要自定义处理，虽然解决了主从下的高可用问题，但是Sentinel并没有提供分区解决方案，还需开发者考虑如何建设。

既然是还是主从，如果异常的写流量搞垮了主节点，那么自动的“故障转移”会不会变成自动“灾难传递”，即slave提升为Master之后挂掉，又进行提升又被挂掉。

不过最后这点也是笔者猜测，并没有听说过出现这种案例，可不必深究。

集群

> 概述

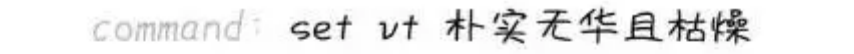
Redis Cluster是官方在**3.0**版本后推出的分布式方案。

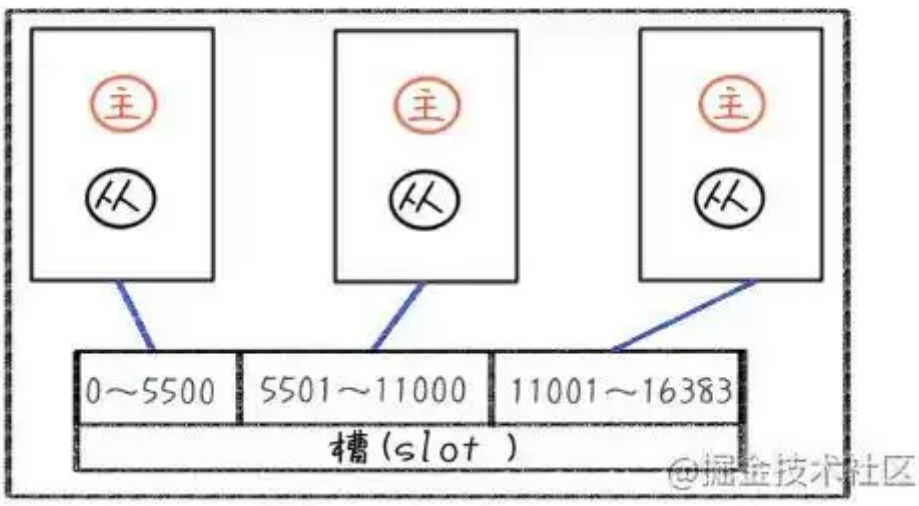
对开发者而言，“官方支持”一词是大概率非常美好的，小到issue，大到feature。自定义去解决问题，成本总是要高一些。

有了官方的正式集群方案，从**请求路由、故障转移、弹性伸缩**几个纬度的使用上，将更为容易。

Cluster不同于哨兵，是支持分区的。有说法Cluster是哨兵的升级，这是不严谨的。二者纬度不一样，如果因为Cluster也有故障转移的功能，就说它是哨兵的升级款，略显牵强。

Cluster在分区管理上，使用了“**哈希槽**”(hash slot)这么一个概念，一共有**16384**个槽位，每个实例负责一部分**槽**，通过CRC16（key）&16383这样的公式，计算出来key所对应的槽位。





虽然在**节点**和**key**二者中又引入了**槽**的概念，看起来不易理解，实际上因为颗粒度更细了，减少了节点的扩容和收缩难度，相比传统策略还是很有优势。

当然，“槽”是虚拟的概念，节点自身去维护“槽”的关系，并不是要真正下载启动个“槽服务”在跑。

> 使用方式

Redis的各种玩法，都是从配置文件着手，集群也不例外。

```
1  cluster-enabled yes
2  cluster-config-file "redis-node.conf"
```

关键配置简洁明了，有两步

- 开启集群
- 指定集群配置文件

集群配置文件(cluster-config-file)为内部使用，可以不去指定，Redis会帮助创建一个。启动还是普通的方式redis-server redis.conf

首先以集群方式启动了N台Redis实例，这当然还没完事。接下来的步骤笔者称为“**牵线搭桥分配槽**”，听起来还算顺口。

“牵线搭桥分配槽”的方式也在不断升级，从直接用原始命令来处理，到使用脚本，以及现在的Redis-cli官方支持，使用哪种方式都可以。

```
1  redis-cli --cluster create 127.0.0.1:7000 127.0.0.1:7001 \
2  127.0.0.1:7002 127.0.0.1:7003 127.0.0.1:7004 127.0.0.1:7005 \
3  --cluster-replicas 1
```

上方的命令即是Redis官网给出的redis-cli的方式用法，一行命令完成“三主三从”以及自动

分配槽的操作。

这样集群就搭建完成了，当然，使用官方提供的check命令检查一下，也是有必要的。

```
1  redis-cli --cluster check 127.0.0.1:7001
```

> 更多

虽然是对分区良好支持，但也有一些分区的老问题，譬如：如果不在同一个“槽”的数据，是没法使用类似mset的**多键操作**。

在select命令页有提到，集群模式下只能使用一个库，虽然平时一般也是这么用的，但是要了解一下。

运维上也要谨慎，俗话说得好，“**使用越简单底层越复杂**”，启动搭建是很方便，使用时面对带宽消耗，数据倾斜等等具体问题时，还需人工介入，或者研究合适的配置参数。

结尾

趣谈

在写“主从”方案的时候，发现有一个有趣的事情：

笔者开始是记得主从的关键命令是SLAVEOF，后来查阅官方的时候，发现命令已经更改为REPLICAOF，虽然SLAVEOF还能用。

官网的一些描述词汇，有的地方还是Slave，也有些是用Replication。好奇的笔者查了一下相关的资料，并看了些Redis作者antirez的有关此时博客，发现已经是两年前的事情了。

其实就是“Slave”这个变量名给了一些人机会，借此“喷”了一波作者，作者也做出了一部分妥协。有兴趣的盆友可以自己搜搜看，技术外的东西就不做评价了，看个乐呵就行。

笔者的主要目的还是：看官方文档的时候，别让不同的“词汇”迷惑了。

END

本文对Redis这些拓展方案都作出了大致描述。具体使用上，还需留意**详细配置**，以及**客户端支持**等综合情况来考量。

```
作者：Vt
来源：https://juejin.cn/post/6844904147943161869
```