https://www.cnblogs.com/mushroom/p/4526912.html

[Redis系列(四)-低成本高可用方案设计](https://www.cnblogs.com/mushroom/p/4526912.html)

关于Redis高可用方案，看到较多的是keepalived、zookeeper方案。 keepalived是主备模式，意味着总有一台浪费着。zookeeper工作量成本偏高。 本文主要介绍下使用官方sentinel做redis高可用方案的设计。

**阅读目录：**

1. [Redis Sentinel](https://www.cnblogs.com/mushroom/p/4526912.html#one)
2. [故障转移消息接收的3种方式](https://www.cnblogs.com/mushroom/p/4526912.html#two)
3. [整体流程图](https://www.cnblogs.com/mushroom/p/4526912.html#three)
4. [总结](https://www.cnblogs.com/mushroom/p/4526912.html#four)

**Redis Sentinel**

**Sentinel介绍**

Sentinel是Redis官方为集群提供的高可用解决方案。 在实际项目中可以使用sentinel去做redis自动故障转移，减少人工介入的工作量。另外sentinel也给客户端提供了监控消息的通知，这样客户端就可根据消息类型去判断服务器的状态，去做对应的适配操作。

下面是Sentinel主要功能列表：

* Monitoring：Sentinel持续检查集群中的master、slave状态，判断是否存活。
* Notification：在发现某个redis实例死的情况下，Sentinel能通过API通知系统管理员或其他程序脚本。
* Automatic failover：如果一个master挂掉后，sentinel立马启动故障转移，把某个slave提升为master。其他的slave重新配置指向新master。
* Configuration provider：对于客户端来说sentinel通知是有效可信赖的。客户端会连接sentinel去请求当前master的地址，一旦发生故障sentinel会提供新地址给客户端。

**Sentinel配置**

Sentinel本质上只是一个运行在特殊模式下的redis服务器，通过不同配置来区分提供服务。 sentinel.conf配置：

// [监控名称] [ip] [port] [多少sentinel同意才发生故障转移]

sentinel monitor mymaster 127.0.0.1 6379 2

// [监控名称] [Master多少毫秒后不回应ping命令，就认为master是主观下线状态]

sentinel down-after-milliseconds mymaster 60000

// [故障转移超时时间]

sentinel failover-timeout mymaster 180000

//[在执行故障转移时,最多可以有多少个从服务器同时对新的主服务器进行同步]

sentinel parallel-syncs mymaster 1

sentinel需要使用redis2.8版本以上，启动如下：

redis-sentinel sentinel.conf

启动后Sentinel会：

* 以10秒一次的频率，向被监视的master发送info命令，根据回复获取master当前信息。
* 以1秒一次的频率，向所有redis服务器、包含sentinel在内发送PING命令，通过回复判断服务器是否在线。
* 以2秒一次的频率，通过向所有被监视的master，slave服务器发送包含当前sentinel，master信息的消息。

另外建议sentinel至少起3个实例以上，并配置2个实例同意即可发生转移。 5个实例，配置3个实例同意以此类推。

**故障转移消息接收的3种方式**

Redis服务器一旦发送故障后，sentinel通过raft算法投票选举新master。 故障转移过程可以通过sentinel的API获取/订阅接收事件消息。

**脚本接收**

//当故障转移期间，可以指定一个“通知”脚本用来告知系统管理员，当前集群的情况。  
//脚本被允许执行的最大时间为60秒，如果超时，脚本将会被终止(KILL)

sentinel notification-script mymaster /var/redis/notify.sh

//故障转移期之后，配置通知客户端的脚本.

sentinel client-reconfig-script mymaster /var/redis/notifyReconfig.sh

**客户端直接接收**

Sentinel的故障转移消息通知使用的是redis发布订阅(详解[Redis发布订阅及客户端编程](http://www.cnblogs.com/mushroom/p/4470006.html" \t "_blank))。就是说在故障转移期间所有产生的事件信息，都通过频道(channel)发布出去。比如我们加台slave服务器，sentinel监听到后会发布加slave的消息到"+slave"频道上，客户端只需要订阅"+slave"频道即可接收到对应消息。

其消息格式如下：  
[实例类型] [事件服务器名称] [服务器ip] [服务器端口] @[master名称] [ip] [端口]

<instance-type> <name> <ip> <port> @ <master-name> <master-ip> <master-port>

通知消息格式示例：

\* //订阅类型， \*即订阅所有事件消息。

-sdown //消息类型

slave 127.0.0.1:6379 127.0.0.1 6379 @ mymaster 127.0.0.1 6381

订阅消息示例：

[复制代码](javascript:void(0);)

using (RedisSentinel rs = new RedisSentinel(CurrentNode.Host, CurrentNode.Port))

{

var redisPubSub = new RedisPubSub(node.Host, node.Port);

redisPubSub.OnMessage += OnMessage;

redisPubSub.OnSuccess += (msg) =>{};

redisPubSub.OnUnSubscribe += (obj) =>{};

redisPubSub.OnError = (exception) =>{ };

redisPubSub.PSubscribe("\*");

}

[复制代码](javascript:void(0);)

**服务间接接收**

这种方式在第二种基础上扩展了一层，即应用端不直接订阅sentinel。 单独做服务去干这件事情，然后应用端提供API供这个服务回调通知。 这样做的好处在于：

* 减少应用端监听失败出错的可能性。
* 应用端由主动方变成被动方，降低耦合。
* 性能提高，轮询变回调。
* 独立成服务可扩展性更高。

比如：

1：以后换掉sentinel，我们只需要动服务即可，应用端无需更改。

2：可以在服务内多增加一层守护线程去主动拉取redis状态，这样可确保即使sentinel不生效，也能及时察觉redis状态，并通知到应用端。 当然这种情况很极端，因为sentinel配的也是多节点，同时挂的几率非常小。   
示例：  
应用端提供回调API，在这个API逻辑下去刷新内存中的Redis连接。

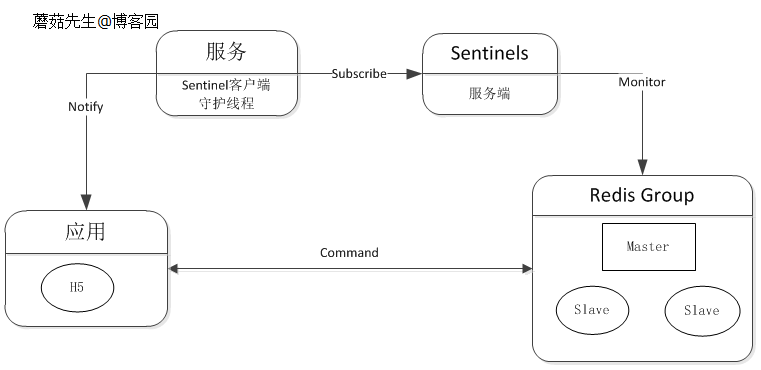
http://127.0.0.1/redis/notify.api

独立服务监控到状况后，调用API通知应用端：

httprequest.post("http://127.0.0/redis/notify.api");

**整体设计**

推荐使用第三种，其整体流程图如下：



**总结**

各种sentinel通知消息类型见官方文档，项目中使用的redis客户端在github上[[HRedis](https://github.com/mushroomsir/HRedis" \t "_blank)]。本文分享了楼主在项目中做Redis高可用的经验，希望对大家有所帮助。 在人力物力满足的情况下还是推荐使用zookeeper方案的。 只有三五杆枪的情况下也就退而求其次，利用最小成本满足需求并保留可扩展性。

**相信没有最好的架构，只有更合适的架构。**

[1] [Redis Sentinel Documentation](http://redis.io/topics/sentinel" \t "_blank)