Redis 的 Sentinel 文档

Redis 的 Sentinel 系统用于管理多个 Redis 服务器（instance）， 该系统执行以下三个任务：

- **监控（Monitoring）**： Sentinel 会不断地检查你的主服务器和从服务器是否运作正常。
- **提醒（Notification）**： 当被监控的某个 Redis 服务器出现问题时， Sentinel 可以通过 API 向管理员或者其他应用程序发送通知。
- **自动故障迁移（Automatic failover）**： 当一个主服务器不能正常工作时， Sentinel 会开始一次自动故障迁移操作，它会将失效主服务器的其中一个从服务器升级为新的主服务器， 并让失效主服务器的其他从服务器改为复制新的主服务器； 当客户端试图连接失效的主服务器时， 集群也会向客户端返回新主服务器的地址， 使得集群可以使用新主服务器代替失效服务器。

Redis Sentinel 是一个分布式系统， 你可以在一个架构中运行多个 Sentinel 进程（progress）， 这些进程使用流言协议（gossip protocols)来接收关于主服务器是否下线的信息， 并使用投票协议（agreement protocols）来决定是否执行自动故障迁移， 以及选择哪个从服务器作为新的主服务器。

虽然 Redis Sentinel 释出为一个单独的可执行文件 redis-sentinel ， 但实际上它只是一个运行在特殊模式下的 Redis 服务器， 你可以在启动一个普通 Redis 服务器时通过给定 –sentinel 选项来启动 Redis Sentinel 。

获取 Sentinel

目前 Sentinel 系统是 Redis 的 unstable 分支的一部分， 你必须到 Redis 项目的 Github 页面 克隆一份 unstable 分值， 然后通过编译来获得 Sentinel 系统。

Sentinel 程序可以在编译后的 src 文档中发现， 它是一个命名为 redis-sentinel 的程序。

你也可以通过下一节介绍的方法， 让 redis-server 程序运行在 Sentinel 模式之下。

另外， 一个新版本的 Sentinel 已经包含在了 Redis 2.8.0 版本的释出文件中。

启动 Sentinel

对于 redis-sentinel 程序， 你可以用以下命令来启动 Sentinel 系统：

对于 redis-server 程序， 你可以用以下命令来启动一个运行在 Sentinel 模式下的 Redis 服务器：

```
redis-server /path/to/sentinel.conf --sentinel
```

两种方法都可以启动一个 Sentinel 实例。

启动 Sentinel 实例必须指定相应的配置文件， 系统会使用配置文件来保存 Sentinel 的当前状态， 并在 Sentinel 重启时通过载入配置文件来进行状态还原。

如果启动 Sentinel 时没有指定相应的配置文件， 或者指定的配置文件不可写（not writable）， 那么 Sentinel 会拒绝启动。

配置 Sentinel

Redis 源码中包含了一个名为 sentinel.conf 的文件， 这个文件是一个带有详细注释的 Sentinel 配置文件示例。

运行一个 Sentinel 所需的最少配置如下所示：

```
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
```

第一行配置指示 Sentinel 去监视一个名为 mymaster 的主服务器， 这个主服务器的 IP 地址为 127.0.0.1 ， 端口号为 6379 ， 而将这个主服务器判断为失效至少需要 2 个 Sentinel 同意 （只要同意 Sentinel 的数量不达标，自动故障迁移就不会执行）。

不过要注意， 无论你设置要多少个 Sentinel 同意才能判断一个服务器失效， 一个 Sentinel 都需要获得系统中多数（majority） Sentinel 的支持， 才能发起一次自动故障迁移， 并预留一个给定的配置纪元 （configuration Epoch ，一个配置纪元就是一个新主服务器配置的版本号）。

换句话说， 在只有少数（minority） Sentinel 进程正常运作的情况下， Sentinel 是不能执行自动故障迁移的。

其他选项的基本格式如下：

```
sentinel <选项的名字> <主服务器的名字> <选项的值>
```

各个选项的功能如下：

- down-after-milliseconds 选项指定了 Sentinel 认为服务器已经断线所需的毫秒数。

如果服务器在给定的毫秒数之内， 没有返回 Sentinel 发送的 PING 命令的回复， 或者返回一个错误， 那么 Sentinel 将这个服务器标记为主观下线（subjectively down，简称 SDOWN ）。

不过只有一个 Sentinel 将服务器标记为主观下线并不一定会引起服务器的自动故障迁移： 只有在足够数量的 Sentinel 都将一个服务器标记为主观下线之后， 服务器才会被标记为客观下线（objectively down， 简称 ODOWN ）， 这时自动故障迁移才会执行。

将服务器标记为客观下线所需的 Sentinel 数量由对主服务器的配置决定。

- parallel-syncs 选项指定了在执行故障转移时， 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， 这个数字越小， 完成故障转移所需的时间就越长。

如果从服务器被设置为允许使用过期数据集（参见对 redis.conf 文件中对 slave-serve-stale-data 选项的说明）， 那么你可能不希望所有从服务器都在同一时间向新的主服务器发送同步请求， 因为尽管复制过程的绝大部分步骤都不会阻塞从服务器， 但从服务器在载入主服务器发来的 RDB 文件时， 仍然会造成从服务器在一段时间内不能处理命令请求： 如果全部从服务器一起对新的主服务器进行同步， 那么就可能会造成所有从服务器在短时间内全部不可用的情况出现。

你可以通过将这个值设为 1 来保证每次只有一个从服务器处于不能处理命令请求的状态。

本文档剩余的内容将对 Sentinel 系统的其他选项进行介绍， 示例配置文件 sentinel.conf 也对相关的选项进行了完整的注释。

主观下线和客观下线

前面说过， Redis 的 Sentinel 中关于下线（down）有两个不同的概念：

- 主观下线（Subjectively Down， 简称 SDOWN）指的是单个 Sentinel 实例对服务器做出的下线判断。
- 客观下线（Objectively Down， 简称 ODOWN）指的是多个 Sentinel 实例在对同一个服务器做出 SDOWN 判断， 并且通过 SENTINEL is-master-down-by-addr 命令互相交流之后， 得出的服务器下线判断。 （一个 Sentinel 可以通过向另一个 Sentinel 发送 SENTINEL is-master-down-by-addr 命令来询问对方是否认为给定的服务器已下线。）

如果一个服务器没有在 master-down-after-milliseconds 选项所指定的时间内， 对向它发送 PING 命令的 Sentinel 返回一个有效回复（valid reply）， 那么 Sentinel 就会将这个服务器标记为主观下线。

服务器对 PING 命令的有效回复可以是以下三种回复的其中一种：

- 返回 +PONG 。
- 返回 -LOADING 错误。
- 返回 -MASTERDOWN 错误。

如果服务器返回除以上三种回复之外的其他回复， 又或者在指定时间内没有回复 PING 命令， 那么 Sentinel 认为服务器返回的回复无效（non-valid）。

注意， 一个服务器必须在 master-down-after-milliseconds 毫秒内， 一直返回无效回复才会被 Sentinel 标记为主观下线。

举个例子， 如果 master-down-after-milliseconds 选项的值为 30000 毫秒（30 秒）， 那么只要服务器能在每 29 秒之内返回至少一次有效回复， 这个服务器就仍然会被认为是处于正常状态的。

从主观下线状态切换到客观下线状态并没有使用严格的法定人数算法（strong quorum algorithm）， 而是使用了流言协议： 如果 Sentinel 在给定的时间范围内， 从其他 Sentinel 那里接收到了足够数量的主服务器下线报告， 那么 Sentinel 就会将主服务器的状态从主观下线改变为客观下线。 如果之后其他 Sentinel 不再报告主服务器已下线， 那么客观下线状态就会被移除。

客观下线条件**只适用于主服务器**： 对于任何其他类型的 Redis 实例， Sentinel 在将它们判断为下线前不需要进行协商， 所以从服务器或者其他 Sentinel 永远不会达到客观下线条件。

只要一个 Sentinel 发现某个主服务器进入了客观下线状态， 这个 Sentinel 就可能会被其他 Sentinel 推选出， 并对失效的主服务器执行自动故障迁移操作。

每个 Sentinel 都需要定期执行的任务

- 每个 Sentinel 以每秒钟一次的频率向它所知的主服务器、从服务器以及其他 Sentinel 实例发送一个 PING 命令。
- 如果一个实例（instance）距离最后一次有效回复 PING 命令的时间超过 down-after-milliseconds 选项所指定的值， 那么这个实例会被 Sentinel 标记为主观下线。 一个有效回复可以是： +PONG 、 -LOADING 或者 -MASTERDOWN 。
- 如果一个主服务器被标记为主观下线， 那么正在监视这个主服务器的所有 Sentinel 要以每秒一次的频率确认主服务器的确进入了主观下线状态。
- 如果一个主服务器被标记为主观下线， 并且有足够数量的 Sentinel （至少要达到配置文件指定的数量）在指定的时间范围内同意这一判断， 那么这个主服务器被标记为客观下线。
- 在一般情况下， 每个 Sentinel 会以每 10 秒一次的频率向它已知的所有主服务器和从服务器发送 INFO 命令。 当一个主服务器被 Sentinel 标记为客观下线时， Sentinel 向下线主服务器的所有从服务器发送 INFO 命令的频率会从 10 秒一次改为每秒一次。
- 当没有足够数量的 Sentinel 同意主服务器已经下线， 主服务器的客观下线状态就会被移除。 当主服务器重新向 Sentinel 的 PING 命令返回有效回复时， 主服务器的主观下线状态就会被移除。

自动发现 Sentinel 和从服务器

一个 Sentinel 可以与其他多个 Sentinel 进行连接， 各个 Sentinel 之间可以互相检查对方的可用性， 并进行信息交换。

你无须为运行的每个 Sentinel 分别设置其他 Sentinel 的地址， 因为 Sentinel 可以通过发布与订阅功能来自动发现正在监视相同主服务器的其他 Sentinel ， 这一功能是通过向频道 **sentinel**:hello 发送信息来实现的。

与此类似， 你也不必手动列出主服务器属下的所有从服务器， 因为 Sentinel 可以通过询问主服务器来获得所有从服务器的信息。

- 每个 Sentinel 会以每两秒一次的频率， 通过发布与订阅功能， 向被它监视的所有主服务器和从服务器的 **sentinel**:hello 频道发送一条信息， 信息中包含了 Sentinel 的 IP 地址、端口号和运行 ID （runid）。
- 每个 Sentinel 都订阅了被它监视的所有主服务器和从服务器的 **sentinel**:hello 频道， 查找之前未出现过的 sentinel （looking for unknown sentinels）。 当一个 Sentinel 发现一个新的 Sentinel 时， 它会将新的 Sentinel 添加到一个列表中， 这个列表保存了 Sentinel 已知的， 监视同一个主服务器的所有其他 Sentinel 。
- Sentinel 发送的信息中还包括完整的主服务器当前配置（configuration）。 如果一个 Sentinel 包含的主服务器配置比另一个 Sentinel 发送的配置要旧， 那么这个 Sentinel 会立即升级到新配置上。
- 在将一个新 Sentinel 添加到监视主服务器的列表上面之前， Sentinel 会先检查列表中是否已经包含了和要添加的 Sentinel 拥有相同运行 ID 或者相同地址（包括 IP 地址和端口号）的 Sentinel ， 如果是的话， Sentinel 会先移除列表

中已有的那些拥有相同运行 ID 或者相同地址的 Sentinel ， 然后再添加新 Sentinel 。

Sentinel API

在默认情况下， Sentinel 使用 TCP 端口 26379 （普通 Redis 服务器使用的是 6379 ）。

Sentinel 接受 Redis 协议格式的命令请求， 所以你可以使用 redis-cli 或者任何其他 Redis 客户端来与 Sentinel 进行通讯。

有两种方式可以和 Sentinel 进行通讯：

- 第一种方法是通过直接发送命令来查询被监视 Redis 服务器的当前状态， 以及 Sentinel 所知道的关于其他 Sentinel 的信息， 诸如此类。
- 另一种方法是使用发布与订阅功能， 通过接收 Sentinel 发送的通知： 当执行故障转移操作， 或者某个被监视的服务器被判断为主观下线或者客观下线时， Sentinel 就会发送相应的信息。

Sentinel 命令

以下列出的是 Sentinel 接受的命令：

- PING： 返回 PONG 。
- SENTINEL masters： 列出所有被监视的主服务器，以及这些主服务器的当前状态。
- SENTINEL slaves： 列出给定主服务器的所有从服务器，以及这些从服务器的当前状态。
- SENTINEL get-master-addr-by-name： 返回给定名字的主服务器的 IP 地址和端口号。 如果这个主服务器正在执行故障转移操作， 或者针对这个主服务器的故障转移操作已经完成， 那么这个命令返回新的主服务器的 IP 地址和端口号。
- SENTINEL reset： 重置所有名字和给定模式 pattern 相匹配的主服务器。 pattern 参数是一个 Glob 风格的模式。 重置操作清楚主服务器目前的所有状态， 包括正在执行中的故障转移， 并移除目前已经发现和关联的， 主服务器的所有从服务器和 Sentinel 。
- SENTINEL failover： 当主服务器失效时， 在不询问其他 Sentinel 意见的情况下， 强制开始一次自动故障迁移 （不过发起故障转移的 Sentinel 会向其他 Sentinel 发送一个新的配置，其他 Sentinel 会根据这个配置进行相应的更新）。

发布与订阅信息

客户端可以将 Sentinel 看作是一个只提供了订阅功能的 Redis 服务器： 你不可以使用 PUBLISH 命令向这个服务器发送信息， 但你可以用 SUBSCRIBE 命令或者 PSUBSCRIBE 命令， 通过订阅给定的频道来获取相应的事件提醒。

一个频道能够接收和这个频道的名字相同的事件。 比如说， 名为 +sdown 的频道就可以接收所有实例进入主观下线（SDOWN）状态的事件。

通过执行 PSUBSCRIBE * 命令可以接收所有事件信息。

以下列出的是客户端可以通过订阅来获得的频道和信息的格式： 第一个英文单词是频道/事件的名字， 其余的是数据的格式。

注意， 当格式中包含 instance details 字样时， 表示频道所返回的信息中包含了以下用于识别目标实例的内容：

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

@ 字符之后的内容用于指定主服务器， 这些内容是可选的， 它们仅在 @ 字符之前的内容指定的实例不是主服务器时使用。

- +reset-master： 主服务器已被重置。
- +slave： 一个新的从服务器已经被 Sentinel 识别并关联。
- +failover-state-reconf-slaves： 故障转移状态切换到了 reconf-slaves 状态。
- +failover-detected： 另一个 Sentinel 开始了一次故障转移操作，或者一个从服务器转换成了主服务器。
- +slave-reconf-sent： 领头（leader）的 Sentinel 向实例发送了 [SLAVEOF](/commands/slaveof.html) 命令，为实例设置新的主服务器。
- +slave-reconf-inprog： 实例正在将自己设置为指定主服务器的从服务器，但相应的同步过程仍未完成。
- +slave-reconf-done： 从服务器已经成功完成对新主服务器的同步。
- -dup-sentinel： 对给定主服务器进行监视的一个或多个 Sentinel 已经因为重复出现而被移除 —— 当 Sentinel 实例重启的时候，就会出现这种情况。

- +sentinel： 一个监视给定主服务器的新 Sentinel 已经被识别并添加。
- +sdown： 给定的实例现在处于主观下线状态。
- -sdown： 给定的实例已经不再处于主观下线状态。
- +odown： 给定的实例现在处于客观下线状态。
- -odown： 给定的实例已经不再处于客观下线状态。
- +new-epoch： 当前的纪元（epoch）已经被更新。
- +try-failover： 一个新的故障迁移操作正在执行中，等待被大多数 Sentinel 选中（waiting to be elected by the majority）。
- +elected-leader： 赢得指定纪元的选举，可以进行故障迁移操作了。
- +failover-state-select-slave： 故障转移操作现在处于 select-slave 状态 —— Sentinel 正在寻找可以升级为主服务器的从服务器。
- no-good-slave： Sentinel 操作未能找到适合进行升级的从服务器。Sentinel 会在一段时间之后再次尝试寻找合适的从服务器来进行升级，又或者直接放弃执行故障转移操作。
- selected-slave： Sentinel 顺利找到适合进行升级的从服务器。
- failover-state-send-slaveof-noone： Sentinel 正在将指定的从服务器升级为主服务器，等待升级功能完成。
- failover-end-for-timeout： 故障转移因为超时而中止，不过最终所有从服务器都会开始复制新的主服务器（slaves will eventually be configured to replicate with the new master anyway）。
- failover-end： 故障转移操作顺利完成。所有从服务器都开始复制新的主服务器了。
- +switch-master： 配置变更，主服务器的 IP 和地址已经改变。 这是绝大多数外部用户都关心的信息。
- +tilt： 进入 tilt 模式。
- -tilt： 退出 tilt 模式。

故障转移

一次故障转移操作由以下步骤组成：

- 发现主服务器已经进入客观下线状态。
- 对我们的当前纪元进行自增（详情请参考 Raft leader election ）， 并尝试在这个纪元中当选。
- 如果当选失败， 那么在设定的故障迁移超时时间的两倍之后， 重新尝试当选。 如果当选成功， 那么执行以下步骤。
- 选出一个从服务器，并将它升级为主服务器。
- 向被选中的从服务器发送 `SLAVEOF NO ONE` 命令，让它转变为主服务器。
- 通过发布与订阅功能， 将更新后的配置传播给所有其他 Sentinel， 其他 Sentinel 对它们自己的配置进行更新。
- 向已下线主服务器的从服务器发送 SLAVEOF 命令， 让它们去复制新的主服务器。
- 当所有从服务器都已经开始复制新的主服务器时， 领头 Sentinel 终止这次故障迁移操作。

每当一个 Redis 实例被重新配置（reconfigured） —— 无论是被设置成主服务器、从服务器、又或者被设置成其他主服务器的从服务器 —— Sentinel 都会向被重新配置的实例发送一个 CONFIG REWRITE 命令， 从而确保这些配置会持久化在硬盘里。

Sentinel 使用以下规则来选择新的主服务器：

- 在失效主服务器属下的从服务器当中， 那些被标记为主观下线、已断线、或者最后一次回复 PING 命令的时间大于五秒钟的从服务器都会被淘汰。
- 在失效主服务器属下的从服务器当中， 那些与失效主服务器连接断开的时长超过 down-after 选项指定的时长十倍的从服务器都会被淘汰。
- 在经历了以上两轮淘汰之后剩下来的从服务器中， 我们选出复制偏移量（replication offset）最大的那个从服务器作为新的主服务器； 如果复制偏移量不可用， 或者从服务器的复制偏移量相同， 那么带有最小运行 ID 的那个从服务器成为新的主服务器。

Sentinel 自动故障迁移的一致性特质

Sentinel 自动故障迁移使用 Raft 算法来选举领头（leader） Sentinel， 从而确保在一个给定的纪元（epoch）里， 只有一个领头产生。

这表示在同一个纪元中， 不会有两个 Sentinel 同时被选中为领头， 并且各个 Sentinel 在同一个纪元中只会对一个领头进行投票。

更高的配置纪元总是优于较低的纪元， 因此每个 Sentinel 都会主动使用更新的纪元来代替自己的配置。

简单来说， 我们可以将 Sentinel 配置看作是一个带有版本号的状态。 一个状态会以最后写入者胜出（last-write-wins）的方式（也即是，最新的配置总是胜出）传播至所有其他 Sentinel 。

举个例子， 当出现网络分割（network partitions）时， 一个 Sentinel 可能会包含了较旧的配置， 而当这个 Sentinel 接到其他 Sentinel 发来的版本更新的配置时， Sentinel 就会对自己的配置进行更新。

如果要在网络分割出现的情况下仍然保持一致性， 那么应该使用 min-slaves-to-write 选项， 让主服务器在连接的从实例少于给定数量时停止执行写操作， 与此同时， 应该在每个运行 Redis 主服务器或从服务器的机器上运行 Redis Sentinel 进程。

Sentinel 状态的持久化

Sentinel 的状态会被持久化在 Sentinel 配置文件里面。

每当 Sentinel 接收到一个新的配置， 或者当领头 Sentinel 为主服务器创建一个新的配置时， 这个配置会与配置纪元一起被保存到磁盘里面。

这意味着停止和重启 Sentinel 进程都是安全的。

Sentinel 在非故障迁移的情况下对实例进行重新配置

即使没有自动故障迁移操作在进行， Sentinel 总会尝试将当前的配置设置到被监视的实例上面。 特别是：

- 根据当前的配置， 如果一个从服务器被宣告为主服务器， 那么它会代替原有的主服务器， 成* 为新的主服务器， 并且成为原有主服务器的所有从服务器的复制对象。 那些连接了错误主服务器的从服务器会被重新配置， 使得这些从服务器会去复制正确的主服务器。

不过， 在以上这些条件满足之后， Sentinel 在对实例进行重新配置之前仍然会等待一段足够长的时间， 确保可以接收到其他 Sentinel 发来的配置更新， 从而避免自身因为保存了过期的配置而对实例进行了不必要的重新配置。

TILT 模式

Redis Sentinel 严重依赖计算机的时间功能： 比如说， 为了判断一个实例是否可用， Sentinel 会记录这个实例最后一次相应 PING 命令的时间， 并将这个时间和当前时间进行对比， 从而知道这个实例有多长时间没有和 Sentinel 进行任何成功通讯。

不过， 一旦计算机的时间功能出现故障， 或者计算机非常忙碌， 又或者进程因为某些原因而被阻塞时， Sentinel 可能也会跟着出现故障。

TILT 模式是一种特殊的保护模式： 当 Sentinel 发现系统有些不对劲时， Sentinel 就会进入 TILT 模式。

因为 Sentinel 的时间中断器默认每秒执行 10 次， 所以我们预期时间中断器的两次执行之间的间隔为 100 毫秒左右。 Sentinel 的做法是， 记录上一次时间中断器执行时的时间， 并将它和这一次时间中断器执行的时间进行对比：

- 如果两次调用时间之间的差距为负值， 或者非常大（超过 2 秒钟）， 那么 Sentinel 进入 TILT 模式。
- 如果 Sentinel 已经进入 TILT 模式， 那么 Sentinel 延迟退出 TILT 模式的时间。

当 Sentinel 进入 TILT 模式时， 它仍然会继续监视所有目标， 但是：

- 它不再执行任何操作，比如故障转移。
- 当有实例向这个 Sentinel 发送 SENTINEL is-master-down-by-addr 命令时， Sentinel 返回负值： 因为这个 Sentinel 所进行的下线判断已经不再准确。

如果 TILT 可以正常维持 30 秒钟， 那么 Sentinel 退出 TILT 模式。

处理 -BUSY 状态

当 Lua 脚本的运行时间超过指定时限时， Redis 就会返回 -BUSY 错误。

当出现这种情况时， Sentinel 在尝试执行故障转移操作之前， 会先向服务器发送一个 SCRIPT KILL 命令， 如果服务器正在执行的是一个只读脚本的话， 那么这个脚本就会被杀死， 服务器就会回到正常状态。

关于topics/sentinel互动的最新评论

geelou 发布于 2016-8-31 17:59:49

topics/sentinel互动

发表评论