**redis的Sentinel**

**sentinel功能**

redis的sentinel系统用于管理多个redis服务器，该系统主要执行三个任务：监控、提醒、自动故障转移。

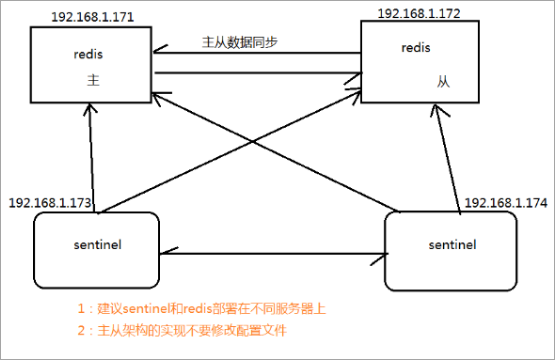
1、监控（Monitoring）： Redis Sentinel实时监控主服务器和从服务器运行状态，并且实现自动切换。

2、提醒（Notification）：当被监控的某个 Redis 服务器出现问题时， Redis Sentinel 可以向系统管理员发送通知， 也可以通过 API 向其他程序发送通知。

3、自动故障转移（Automatic failover）： 当一个主服务器不能正常工作时，Redis Sentinel 可以将一个从服务器升级为主服务器， 并对其他从服务器进行配置，让它们使用新的主服务器。当应用程序连接Redis 服务器时， Redis Sentinel会告之新的主服务器地址和端口。

注意：在使用sentinel监控主从节点的时候，从节点需要是使用动态方式配置的，如果直接修改配置文件，后期sentinel实现故障转移的时候会出问题。

**图示sentinel**



**主观下线和客观下线：**

1、主观下线状态：当一个sentinel认为一个redis服务连接不上的时候，会给这个服务打个标记为下线状态。

2、客观下线状态：当多个sentinel认为一个redids连接不上的时候，则认为这个redis服务确实下线了。这里的多个sentinel的个数可以在配置文件中设置。

主节点：主观下线和客观下线

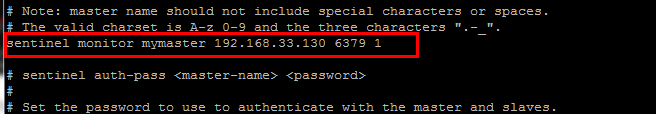
从节点：主观下线状态

### ****sentinel配置****

修改sentinel.conf文件

**[python]** [view plain](http://blog.csdn.net/u011204847/article/details/51307044) [copy](http://blog.csdn.net/u011204847/article/details/51307044)

1. sentinel monitor mymaster 192.168.33.130 6379 2     #最后一个参数视情况决定



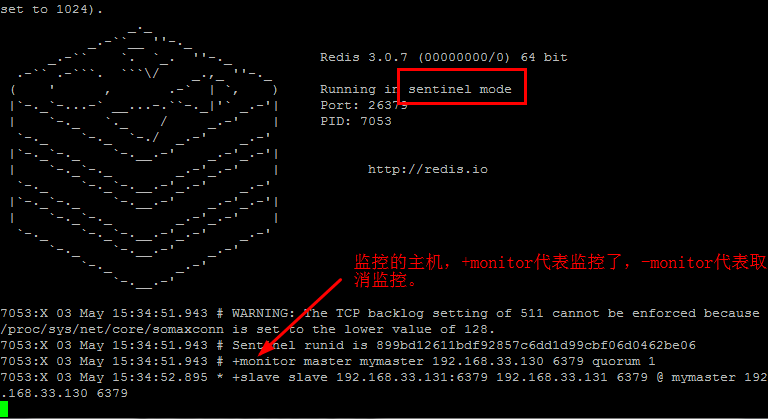
最后一个参数为需要判定客观下线所需的主观下线sentinel个数，这个参数不可以大于sentinel个数。

启动sentinel

**[python]** [view plain](http://blog.csdn.net/u011204847/article/details/51307044) [copy](http://blog.csdn.net/u011204847/article/details/51307044)

1. redis-sentinel sentinel.conf

启动后结果图示：



sentinel日志明细说明

<http://redisdoc.com/topic/sentinel.html>

通过订阅指定的频道信息，当服务器出现故障得时候通知管理员

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。

**sentinel的一些命令**

**[python]** [view plain](http://blog.csdn.net/u011204847/article/details/51307044) [copy](http://blog.csdn.net/u011204847/article/details/51307044)

1. INFO

sentinel的基本状态信息

**[python]** [view plain](http://blog.csdn.net/u011204847/article/details/51307044) [copy](http://blog.csdn.net/u011204847/article/details/51307044)

1. 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

**[python]** [view plain](http://blog.csdn.net/u011204847/article/details/51307044) [copy](http://blog.csdn.net/u011204847/article/details/51307044)

1. SENTINEL slaves <master name>

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

**[python]** [view plain](http://blog.csdn.net/u011204847/article/details/51307044) [copy](http://blog.csdn.net/u011204847/article/details/51307044)

1. SENTINEL get-master-addr-by-name <master name>

返回给定名字的主服务器的 IP 地址和端口号

**[python]** [view plain](http://blog.csdn.net/u011204847/article/details/51307044) [copy](http://blog.csdn.net/u011204847/article/details/51307044)

1. SENTINEL reset <pattern>

重置所有名字和给定模式 pattern 相匹配的主服务器。重置操作清除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 。

**[python]** [view plain](http://blog.csdn.net/u011204847/article/details/51307044) [copy](http://blog.csdn.net/u011204847/article/details/51307044)

1. SENTINEL failover <master name>

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，但是它会给其他sentinel发送一个最新的配置，其他sentinel会根据这个配置进行更新

### ****java操作sentinel****

代码示例：

sentinel日志明细说明

<http://redisdoc.com/topic/sentinel.html>

通过订阅指定的频道信息，当服务器出现故障得时候通知管理员

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。

**sentinel的一些命令**

**[python]** [view plain](http://blog.csdn.net/u011204847/article/details/51307044) [copy](http://blog.csdn.net/u011204847/article/details/51307044)

1. INFO

sentinel的基本状态信息

**[python]** [view plain](http://blog.csdn.net/u011204847/article/details/51307044) [copy](http://blog.csdn.net/u011204847/article/details/51307044)

1. 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

**[python]** [view plain](http://blog.csdn.net/u011204847/article/details/51307044) [copy](http://blog.csdn.net/u011204847/article/details/51307044)

1. SENTINEL slaves <master name>

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

**[python]** [view plain](http://blog.csdn.net/u011204847/article/details/51307044) [copy](http://blog.csdn.net/u011204847/article/details/51307044)

1. SENTINEL get-master-addr-by-name <master name>

返回给定名字的主服务器的 IP 地址和端口号

**[python]** [view plain](http://blog.csdn.net/u011204847/article/details/51307044) [copy](http://blog.csdn.net/u011204847/article/details/51307044)

1. SENTINEL reset <pattern>

重置所有名字和给定模式 pattern 相匹配的主服务器。重置操作清除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 。

**[python]** [view plain](http://blog.csdn.net/u011204847/article/details/51307044) [copy](http://blog.csdn.net/u011204847/article/details/51307044)

1. SENTINEL failover <master name>

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，但是它会给其他sentinel发送一个最新的配置，其他sentinel会根据这个配置进行更新

**java操作sentinel**

代码示例：

**[java]** [view plain](http://blog.csdn.net/u011204847/article/details/51307044) [copy](http://blog.csdn.net/u011204847/article/details/51307044)

1. **import** java.util.HashSet;
2. //需要在pom.xml文件中引入jedis依赖
3. **import** redis.clients.jedis.HostAndPort;
4. **import** redis.clients.jedis.Jedis;
5. **import** redis.clients.jedis.JedisPoolConfig;
6. **import** redis.clients.jedis.JedisSentinelPool;
8. **public** **class** SentinelTest {
10. **public** **static** **void** main(String[] args) {
11. // 使用HashSet添加多个sentinel
12. HashSet<String> sentinels = **new** HashSet<String>();
13. // 添加sentinel主机和端口
14. sentinels.add("192.168.33.131:26379");
16. // 创建config
17. JedisPoolConfig poolConfig = **new** JedisPoolConfig();
18. // 控制一个pool最多有多少个状态为idle(空闲的)的jedis实例。
19. poolConfig.setMaxIdle(10);
20. // 控制一个pool最多有多少个jedis实例。
21. poolConfig.setMaxTotal(100);
22. // 表示当borrow(引入)一个jedis实例时，最大的等待时间，如果超过等待时间，则直接抛出JedisConnectionException；
23. poolConfig.setMaxWaitMillis(2000);
24. // 在borrow一个jedis实例时，是否提前进行validate操作；如果为true，则得到的jedis实例均是可用的；
25. poolConfig.setTestOnBorrow(**true**);
27. // 通过Jedis连接池创建一个Sentinel连接池
28. JedisSentinelPool pool = **new** JedisSentinelPool("mymaster", sentinels,poolConfig);
29. // 获取master的主机和端口
30. HostAndPort currentHostMaster = pool.getCurrentHostMaster();
31. System.out.println(currentHostMaster.getHost() + "--"+ currentHostMaster.getPort());
32. // 从Sentinel池中获取资源
33. Jedis resource = pool.getResource();
34. // 打印资源中key为name的值
35. System.out.println(resource.get("name"));
36. // 关闭资源
37. resource.close();
38. }
39. }

打印结果：

