## Redis Sentinel主从高可用方案（附Jedis Sentinel教程）

本文介绍一种通过Jed和Sentinel实现Redis集群(主从)的高可用方案，该方案需要使用Jedis2.2.2及以上版本（强制），Redis2.8及以上版本(可选，Sentinel最早出现在Redis2.4中，**Redis2.8中Sentinel更加稳定)，**

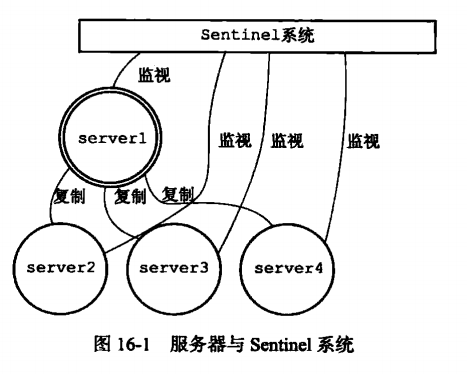
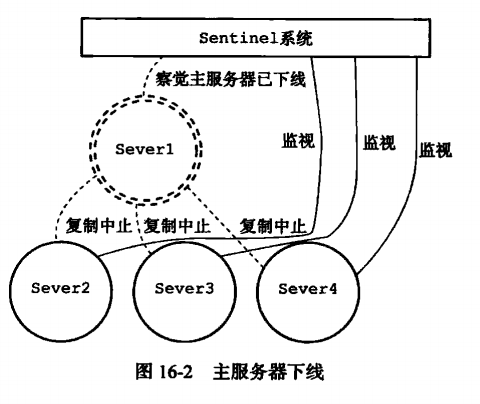
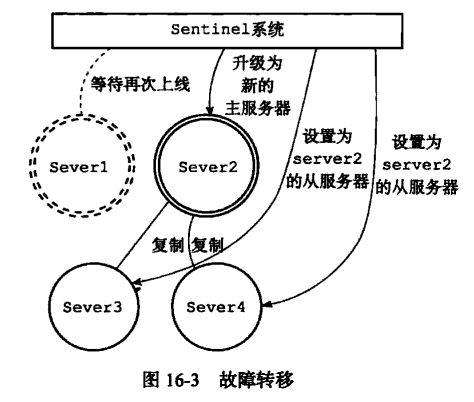
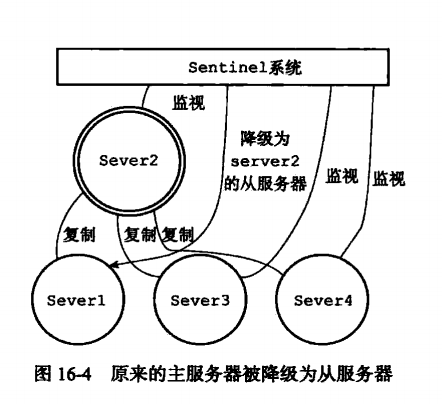
附：  
Redis Cluster集群主从方案：[http://wosyingjun.iteye.com/blog/2289220](http://wosyingjun.iteye.com/blog/2289220" \t "http://wosyingjun.iteye.com/blog/_blank)  
Redis Sentinel主从高可用方案：[http://wosyingjun.iteye.com/blog/2289593](http://wosyingjun.iteye.com/blog/2289593" \t "http://wosyingjun.iteye.com/blog/_blank)

### 一、Sentinel介绍

Sentinel是Redis的高可用性（HA）解决方案，由一个或多个Sentinel实例组成的Sentinel系统可以监视任意多个主服务器，以及这些主服务器属下的所有从服务器，并在被监视的主服务器进行下线状态时，自动将下线主服务器属下的某个从服务器升级为新的主服务器，然后由新的主服务器代替已下线的主服务器继续处理命令请求。Redis提供的sentinel（哨兵）机制，通过sentinel模式启动redis后，自动监控master/slave的运行状态，基本原理是：心跳机制+投票裁决

* 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。

### 二、Sentinel的主从原理

之前介绍过为什么Jedis要用2.2.2及以上版本，因为主从实例地址(IP PORT)是不同的，当故障发生进行主从切换后，应用程序无法知道新地址，故在Jedis2.2.2中新增了对Sentinel的支持，应用通过redis.clients.jedis.JedisSentinelPool.getResource()取得的Jedis实例会及时更新到新的主实例地址。

### 三、Redis Sentinel配置

**这里我采用2个哨兵，1个主redis，2个从redis的方式，配置文件如下：**



**sentinel\_63791.conf 配置：**

port 63791

daemonize yes

logfile "/var/log/sentinel\_63791.log"

#master-1

sentinel monitor master-1 192.168.78.99 6379 2

sentinel down-after-milliseconds master-1 5000

sentinel failover-timeout master-1 18000

sentinel auth-pass master-1 yingjun

sentinel parallel-syncs master-1 1

**sentinel\_63792.conf 配置：**

port 63792

daemonize yes

logfile "/var/log/sentinel\_63792.log"

#master-1

sentinel monitor master-1 192.168.78.99 6379 2

sentinel down-after-milliseconds master-1 5000

sentinel failover-timeout master-1 18000

sentinel auth-pass master-1 yingjun

sentinel parallel-syncs master-1 1

**redis\_master\_6379.conf 配置：**  
**在原配置文件中作如下修改：**

port 6379

daemonize yes

requirepass yingjun

masterauth yingjun

**redis\_slave\_6380.conf 配置：**  
**在原配置文件中作如下修改：**

port 6380

daemonize yes

requirepass yingjun

slaveof 192.168.78.99 6379

masterauth yingjun

**redis\_slave\_6381.conf 配置：**  
**在原配置文件中作如下修改：**

port 6381

daemonize yes

requirepass yingjun

slaveof 192.168.78.99 6379

masterauth yingjun

**按如下顺序依次启动服务：**

./redis-server ../conf/redis\_master\_6379.conf

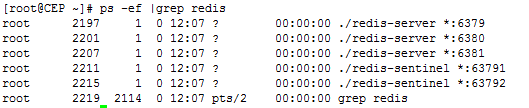
./redis-server ../conf/redis\_slave\_6381.conf

./redis-server ../conf/redis\_slave\_6382.conf

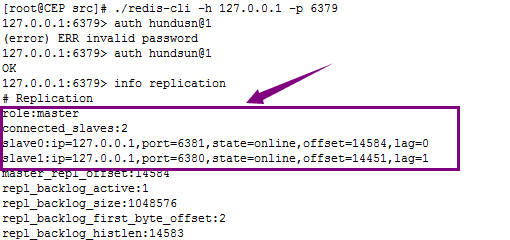
./redis-sentinel ../conf/sentinel\_63791.conf

./redis-sentinel ../conf/sentinel\_63792.conf

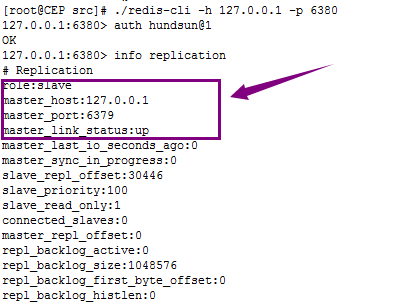
**查看进程是否都已经启动：**



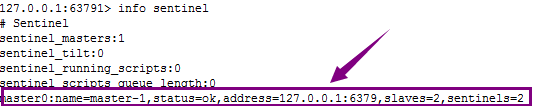
**查看master的状态：**



**查看slave的状态：**



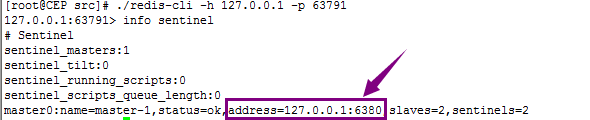
**查看sentinel的状态：**



**接下来验证redis sentinel的主从切换：**

* 首先关闭主redis（6379）服务（shutdown）。

查看哨兵，发现端口号为6380的从服务变成了主服务,sentinel自动完成了故障切换。



* 启动刚才被shutdown的6379服务并查看，发现它变成了从服务。
* 

### 三、Jedis Sentinel教程

**Maven依赖：**

<dependency>

<groupId>redis.clients</groupId>

<artifactId>jedis</artifactId>

<version>2.8.0</version>

</dependency>

<!-- spring-redis -->

<dependency>

<groupId>org.springframework.data</groupId>

<artifactId>spring-data-redis</artifactId>

<version>1.6.4.RELEASE</version>

</dependency>

**redis的配置文件：**

#redis config

redis.pass=yingjun

redis.pool.maxTotal=105

redis.pool.maxIdle=10

redis.pool.maxWaitMillis=60000

redis.pool.testOnBorrow=true

sentinel1.ip=192.168.78.99

sentinel1.port=63791

sentinel2.ip=192.168.78.99

sentinel2.port=63792

**Spring的配置文件：**

<!-- Redis 配置 -->

<bean id="jedisPoolConfig" class="redis.clients.jedis.JedisPoolConfig">

<property name="maxTotal" value="${redis.pool.maxTotal}" />

<property name="maxIdle" value="${redis.pool.maxIdle}" />

<property name="maxWaitMillis" value="${redis.pool.maxWaitMillis}" />

<property name="testOnBorrow" value="${redis.pool.testOnBorrow}" />

</bean>

<bean id="sentinelConfiguration"

class="org.springframework.data.redis.connection.RedisSentinelConfiguration">

<property name="master">

<bean class="org.springframework.data.redis.connection.RedisNode">

<property name="name" value="master-1"></property>

</bean>

</property>

<property name="sentinels">

<set>

<bean class="org.springframework.data.redis.connection.RedisNode">

<constructor-arg name="host" value="${sentinel1.ip}"></constructor-arg>

<constructor-arg name="port" value="${sentinel1.port}"></constructor-arg>

</bean>

<bean class="org.springframework.data.redis.connection.RedisNode">

<constructor-arg name="host" value="${sentinel2.ip}"></constructor-arg>

<constructor-arg name="port" value="${sentinel2.port}"></constructor-arg>

</bean>

</set>

</property>

</bean>

<!-- Jedis ConnectionFactory连接配置 -->

<bean id="jedisConnectionFactory"

class="org.springframework.data.redis.connection.jedis.JedisConnectionFactory">

<property name="password" value="${redis.pass}"></property>

<property name="poolConfig" >

<ref bean="jedisPoolConfig"/>

</property>

<constructor-arg name="sentinelConfig" ref="sentinelConfiguration"></constructor-arg>

</bean>

<!-- redisTemplate配置，redisTemplate是对Jedis的对redis操作的扩展，有更多的操作，封装使操作更便捷 -->

<bean id="redisTemplate" class="org.springframework.data.redis.core.StringRedisTemplate">

<property name="connectionFactory" ref="jedisConnectionFactory" />

</bean>

**代码中直接用redisTemplate调用：**

@Override

public boolean add(final KeyToken tkey) {

boolean result = redisTemplate.execute(new RedisCallback<Boolean>() {

@Override

public Boolean doInRedis(RedisConnection connection) throws DataAccessException {

RedisSerializer<String> serializer = getRedisSerializer();

byte[] key = serializer.serialize(tkey.getIndex());

byte[] name = serializer.serialize(tkey.getExpire\_time());

return connection.setNX(key, name);

}

});

return result;

}