

传智健康项目 第13章

本章内容我们的主题为Redis，目前Redis在企业中的应用已经非常广泛，同时Redis也是面试中的重点内容。

1. Redis缓存相关问题

1.1 缓存穿透

缓存穿透是指查询一个数据库一定不存在的数据。

我们以前正常的使用Redis缓存的流程大致是：

- 1、数据查询首先进行缓存查询
- 2、如果数据存在则直接返回缓存数据
- 3、如果数据不存在，就对数据库进行查询，并把查询到的数据放进缓存
- 4、如果数据库查询数据为空，则不放进缓存

例如我们的数据表中主键是自增产生的，所有的主键值都大于0。此时如果用户传入的参数为-1，会怎么样？这个-1，就是一定不存在的对象。程序就会每次都去查询数据库，而每次查询都是空，每次又都不会进行缓存。假如有人恶意攻击，就可以利用这个漏洞，对数据库造成压力，甚至压垮我们的数据库。

为了防止有人利用这个漏洞恶意攻击我们的数据库，我们可以采取如下措施：

如果从数据库查询的对象为空，也放入缓存，key为用户提交过来的主键值，value为null，只是设定的缓存过期时间较短，比如设置为60秒。这样下次用户再根据这个key查询redis缓存就可以查询到值了（当然值为null），从而保护我们的数据库免遭攻击。

1.2 缓存雪崩

缓存雪崩，是指在某一个时间段，缓存集中过期失效。在缓存集中失效的这个时间段对数据的访问查询，都落到了数据库上，对于数据库而言，就会产生周期性的压力波峰。

为了避免缓存雪崩的发生，我们可以将缓存的数据设置不同的失效时间，这样就可以避免缓存数据在某一个时间段集中失效。例如对于热门的数据（访问频率高的数据）可以缓存的时间长一些，对于冷门的数据可以缓存的时间短一些。甚至对于一些特别热门的数据可以设置永不过期。

1.3 缓存击穿

缓存击穿，是指一个key非常热点（例如双十一期间进行抢购的商品数据），在不停的扛着大并发，大并发集中对这一个点进行访问，当这个key在失效的瞬间，持续的大并发就穿破缓存，直接请求到数据库上，就像在一个屏障上凿开了一个洞。

我们同样可以将这些热点数据设置永不过期就可以解决缓存击穿的问题了。

2. Redis集群方案

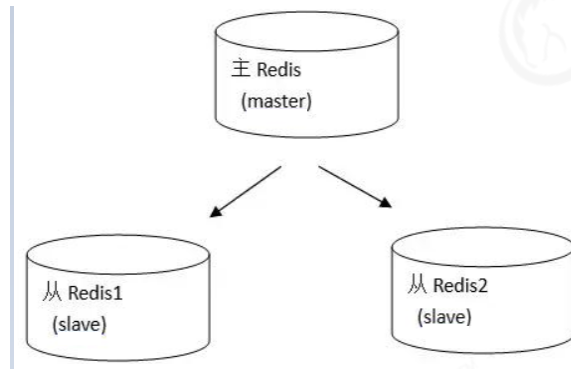
单台Redis服务性能不足的问题。这就需要使用到Redis的集群了。Redis集群有多种方案，下面分别进行讲解。

2.1 主从复制Replication

redis支持主从复制的模式。

在主从复制模式下Redis节点分为两种角色：主节点(也称为master)和从节点(也称为slave)。这种模式集群是由一个主节点和多个从节点构成。

原则：Master会将数据同步到slave，而slave不会将数据同步到master。Slave启动时会连接master来同步数据。



这是一个典型的分布式读写分离模型。我们可以利用master来处理写操作，slave提供读操作。这样可以有效减少单个机器的并发访问数量。

要实现主从复制这种模式非常简单，主节点不用做任何修改，直接启动服务即可。从节点需要修改redis.conf配置文件，加入配置：slaveof <主节点ip地址> <主节点端口号>，例如master的ip地址为192.168.200.129，端口号为6379，那么slave只需要在redis.conf文件中配置slaveof 192.168.200.129 6379即可。

分别连接主节点和从节点，测试发现主节点的写操作，从节点立刻就能看到相同的数据。但是在从节点进行写操作，提示 `READONLY You can't write against a read only slave` 不能写数据到从节点。

现在我们就可以通过这种方式配置多个从节点进行读操作，主节点进行写操作，实现读写分离。

2.2 哨兵sentinel

我们现在已经给Redis实现了主从复制，可将主节点数据同步给从节点，实现了读写分离，提高Redis的性能。但是现在还存在一个问题，就是在主从复制这种模式下只有一个主节点，一旦主节点宕机，就无法再进行写操作了。也就是说主从复制这种模式没有实现高可用。那么什么是高可用呢？如何实现高可用呢？

2.2.1 高可用介绍

高可用(HA)是分布式系统架构设计中必须考虑的因素之一，它是通过架构设计减少系统不能提供服务的时间。保证高可用通常遵循下面几点：

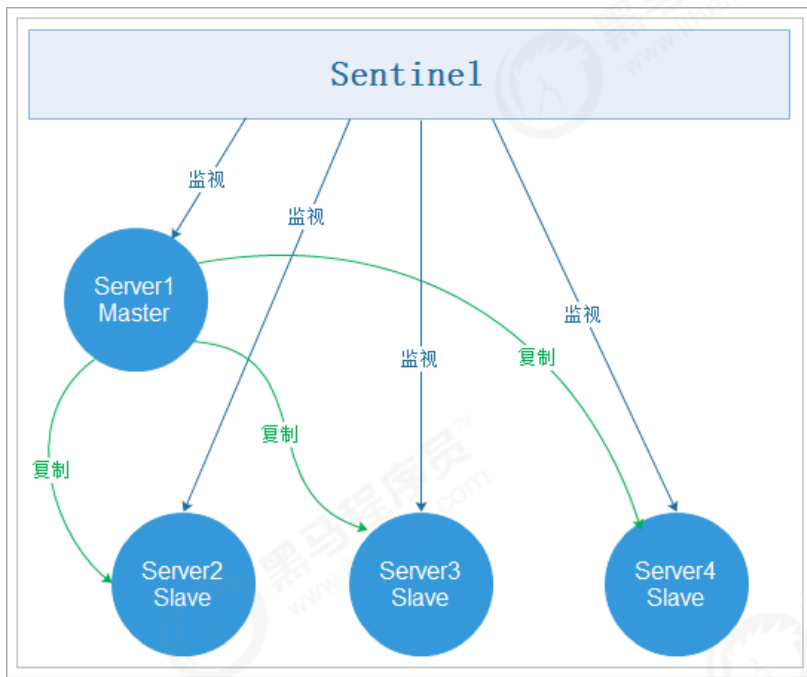
1. 单点是系统高可用的大敌，应该尽量在系统设计的过程中避免单点。
2. 通过架构设计而保证系统高可用的，其核心准则是：冗余。
3. 实现自动故障转移。

sentinel(哨兵)是用于监控redis集群中Master状态的工具，其本身也是一个独立运行的进程，是Redis的高可用解决方案，sentinel哨兵模式已经被集成在redis2.4之后的版本中。

sentinel可以监视一个或者多个redis master服务，以及这些master服务的所有从服务；当某个master服务下线时，自动将该master下的某个从服务升级为master服务替代已下线的master服务继续处理请求，并且其余从节点开始从新的主节点复制数据。

在redis安装完成后，会有一个redis-sentinel的文件，这就是启动sentinel的脚本文件，同时还有一个sentinel.conf文件，这个是sentinel的配置文件。

sentinel工作模式：



注意：可能有些同学会有疑问，现在我们已经基于sentinel实现了高可用，但是如果sentinel挂了怎么办呢？其实sentinel本身也可以实现集群，也就是说sentinel也是高可用的。

2.2.3 Redis sentinel使用

2.2.3.1 配置sentinel

Sentinel在redis的安装包中有，我们直接使用就可以了，但是先需要修改配置文件，执行命令：

```
cd /usr/local/redis/

# 复制sentinel配置文件
cp /root/redis-4.0.14/sentinel.conf sentinel01.conf

# 修改配置文件：
vi sentinel01.conf
```

在sentinel01.conf配置文件中添加：



```
sentinel monitor mymaster 127.0.0.1 6379 1
sentinel down-after-milliseconds mymaster 10000
sentinel failover-timeout mymaster 60000
sentinel parallel-syncs mymaster 1
```

注意：如果有sentinel monitor mymaster 127.0.0.1 6379 2配置，则注释掉。

参数说明：

- sentinel monitor mymaster 192.168.200.129 6379 1
mymaster 主节点名,可以任意起名，但必须和后面的配置保持一致。
192.168.200.128 6379 主节点连接地址。
1 将主服务器判断为失效需要投票，这里设置至少需要 1个 Sentinel 同意。
- sentinel down-after-milliseconds mymaster 10000
设置Sentinel认为服务器已经断线所需的毫秒数。
- sentinel failover-timeout mymaster 60000
设置failover（故障转移）的过期时间。当failover开始后，在此时间内仍然没有触发任何failover操作，当前 sentinel 会认为此次failover失败。
- sentinel parallel-syncs mymaster 1
设置在执行故障转移时，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，表示同时进行同步的从服务器越少，那么完成故障转移所需的时间就越长。

2.2.3.2 启动sentinel

配置文件修改后，执行以下命令，启动sentinel：

```
/root/redis-4.0.14/src/redis-sentinel sentinel01.conf
```

效果如下：

```
Sentinel ID is 369141c78c1cd7582c2bfcd6b12a7543b6708b65
+monitor master mymaster 192.168.200.129 6379 quorum 1
+slave slave 192.168.200.129:6380 192.168.200.129 6380 @ mymaster 192.168.200.129 6379
+slave slave 192.168.200.129:6381 192.168.200.129 6381 @ mymaster 192.168.200.129 6379
```

可以看到，6379是主服务，6380和6381是从服务。

2.2.3.3 测试sentinel

我们在6379执行shutdown，关闭主服务，Sentinel提示如下：

```
+sdown master mymaster 192.168.200.129 6379 #主节点宕机
+odown master mymaster 192.168.200.129 6379 #quorum 1/1
+new-epoch 1
+try-failover master mymaster 192.168.200.129 6379 #尝试故障转移
+vote-for-leader 00a6933e0cfa2b1bf0c3aab0d6b7a1a6455832ec 1 #选举领导
+elected-leader master mymaster 192.168.200.129 6379
+failover-state-select-slave master mymaster 192.168.200.129 6379 #故障转移选择从服务
```



```
#故障转移状态发送 发送到6380
+failover-state-send-slaveof-noone slave 192.168.200.129:6380 192.168.200.129
6380 @ mymaster 192.168.200.129 6379
+failover-state-wait-promotion slave 192.168.200.129:6380 192.168.200.129 6380 @
mymaster 192.168.200.129 6379
+promoted-slave slave 192.168.200.129:6380 192.168.200.129 6380 @ mymaster
192.168.200.129 6379
+failover-state-reconf-slaves master mymaster 192.168.200.129 6379
+slave-reconf-sent slave 192.168.200.129:6381 192.168.200.129 6381 @ mymaster
192.168.200.129 6379
+slave-reconf-inprog slave 192.168.200.129:6381 192.168.200.129 6381 @ mymaster
192.168.200.129 6379
+slave-reconf-done slave 192.168.200.129:6381 192.168.200.129 6381 @ mymaster
192.168.200.129 6379
+failover-end master mymaster 192.168.200.129 6379 #故障转移结束，原来的主服务是6379
+switch-master mymaster 192.168.200.129 6379 192.168.200.129 6380 #转换主服务，由原
来的6379转为现在的6380
+slave slave 192.168.200.129:6381 192.168.200.129 6381 @ mymaster
192.168.200.129 6380
+slave slave 192.168.200.129:6379 192.168.200.129 6379 @ mymaster
192.168.200.129 6380
+sdown slave 192.168.200.129:6379 192.168.200.129 6379 @ mymaster
192.168.200.129 6380
```

根据提示信息，我们可以看到，6379故障转移到了6380，通过投票选择6380为新的主服务器。

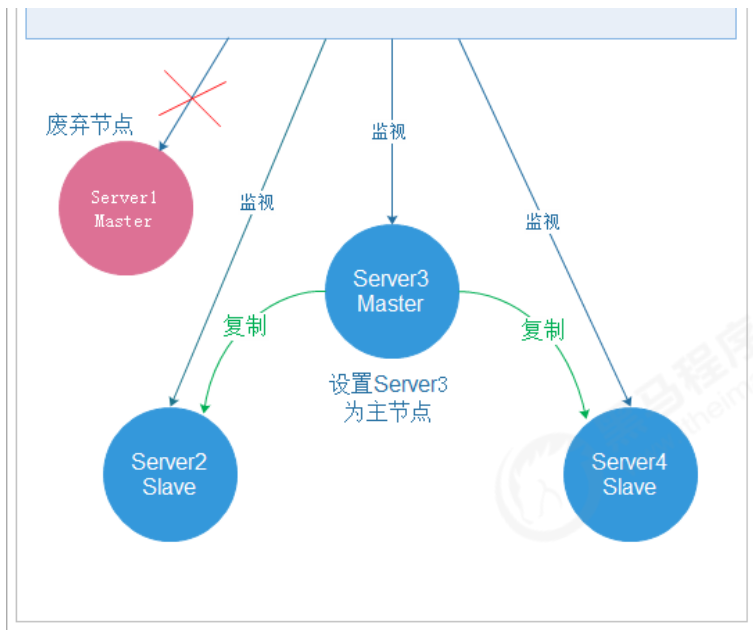
在6380执行info

```
# Replication
role:master
connected_slaves:1
slave0:ip=127.0.0.1,port=6381,state=online,offset=80531,lag=1
```

在6381执行info

```
# Replication
role:slave
master_host:127.0.0.1
master_port:6380
master_link_status:up
```

故障转移如下图：



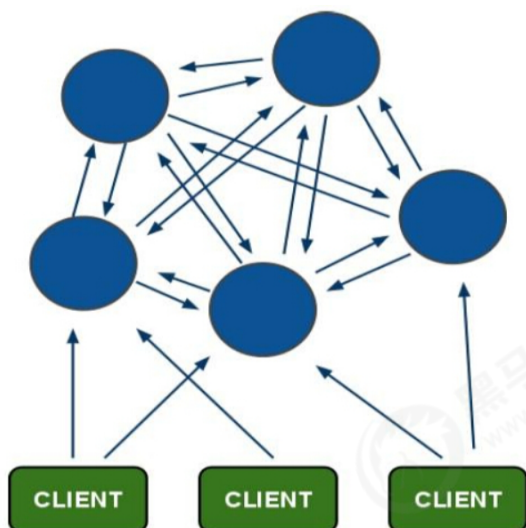
2.3 Redis内置集群cluster

2.3.1 Redis cluster介绍

Redis Cluster是Redis的内置集群，在Redis3.0推出的实现方案。在Redis3.0之前是没有这个内置集群的。Redis Cluster是无中心节点的集群架构，依靠Gossip协议协同自动化修复集群的状态。

Redis cluster在设计的时候，就考虑到了去中心化，去中间件，也就是说，集群中的每个节点都是平等的关系，都是对等的，每个节点都保存各自的数据和整个集群的状态。每个节点都和其他所有节点连接，而且这些连接保持活跃，这样就保证了我们只需要连接集群中的任意一个节点，就可以获取到其他节点的数据。

Redis cluster集群架构图如下：



2.3.2 哈希槽方式分配数据

需要注意的是，这种集群模式下集群中每个节点保存的数据并不是所有的数据，而只是一部分数据。那么数据是如何合理的分配到不同的节点上的呢？

Redis 集群是采用一种叫做 哈希槽 (hash slot) 的方式来分配数据的。redis cluster 默认分配了 16384 个slot，当我们set一个key时，会用 CRC16 算法来取模得到所属的 slot，然后将这个key分到哈希槽区间的节点上，具体算法就是： $CRC16(key) \% 16384$ 。

三个节点分别承担的slot 区间是：

- 节点A覆盖0 - 5460
- 节点B覆盖5461 - 10922
- 节点C覆盖10923 - 16383

那么，现在要设置一个key ,比如叫 my_name：

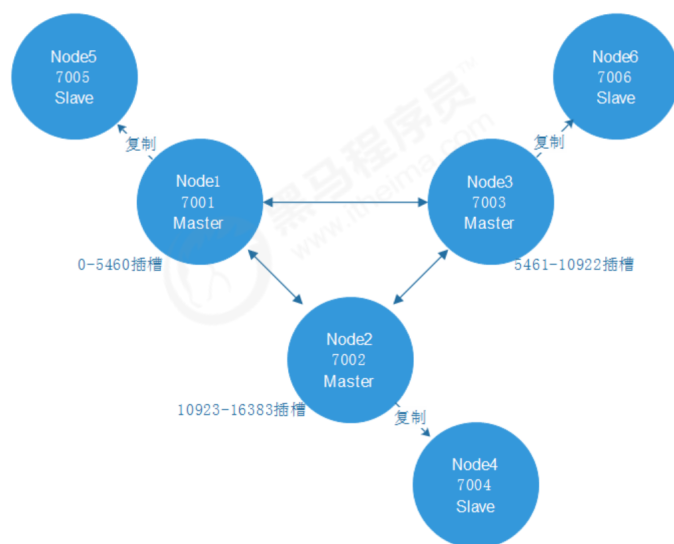
```
set my_name itcast
```

按照redis cluster的哈希槽算法： $\text{CRC16}(\text{'my_name'}) \% 16384 = 2412$ 。那么就会把这个key 的存储分配到 节点A 上了。

2.3.3 Redis cluster的主从模式

redis cluster 为了保证数据的高可用性，加入了主从模式，一个主节点对应一个或多个从节点，主节点提供数据存取，从节点则是从主节点拉取数据备份，当这个主节点挂掉后，就会在这些从节点中选取一个来充当主节点，从而保证集群不会挂掉。

redis cluster加入了主从模式后的效果如下：



2.3.4 Redis cluster搭建

2.3.4.1 准备Redis节点

为了保证可以进行投票，需要至少3个主节点。

每个主节点都需要至少一个从节点,所以需要至少3个从节点。

一共需要6台redis服务器，我们这里使用6个redis实例，端口号为7001~7006。

先准备一个干净的redis环境，复制原来的bin文件夹，清理后作为第一个redis节点，具体命令如下：

```
# 进入redis安装目录
cd /usr/local/redis
# 复制redis
mkdir cluster
cp -R bin/ cluster/node1
# 删除持久化文件
```

```
rm -rf appendonly.aof
# 删除原来的配置文件
rm -rf redis.conf
# 复制新的配置文件
cp /root/redis-4.0.14/redis.conf ./
# 修改配置文件
vi redis.conf
```

集群环境redis节点的配置文件如下：

```
# 不能设置密码，否则集群启动时会连接不上
# Redis服务器可以跨网络访问
bind 0.0.0.0
# 修改端口号
port 7001
# Redis后台启动
daemonize yes
# 开启aof持久化
appendonly yes
# 开启集群
cluster-enabled yes
# 集群的配置 配置文件首次启动自动生成
cluster-config-file nodes.conf
# 请求超时
cluster-node-timeout 5000
```

第一个redis节点node1准备好之后，再分别复制5份，

```
cp -R node1/ node2
```

修改六个节点的端口号为7001~7006，修改redis.conf配置文件即可

编写启动节点的脚本：

```
vi start-all.sh
```

内容为：

```
cd node1
./redis-server redis.conf
cd ..
cd node2
./redis-server redis.conf
cd ..
cd node3
./redis-server redis.conf
cd ..
cd node4
./redis-server redis.conf
cd ..
cd node5
./redis-server redis.conf
cd ..
```

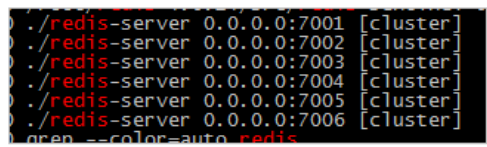



```
cd ..
```

设置脚本的权限，并启动：

```
chmod 744 start-all.sh  
./start-all.sh
```

使用命令 `ps -ef | grep redis` 查看效果如下：



2.3.4.2 启动Redis集群

redis集群的管理工具使用的是ruby脚本语言，安装集群需要ruby环境，先安装ruby环境：

```
# 安装ruby  
yum -y install ruby ruby-devel rubygems rpm-build  
  
# 升级ruby版本，redis4.0.14集群环境需要2.2.2以上的ruby版本  
yum install centos-release-scl-rh  
yum install rh-ruby23 -y  
scl enable rh-ruby23 bash  
  
# 查看ruby版本  
ruby -v
```

下载符合环境要求的gem，下载地址如下：

<https://rubygems.org/gems/redis/versions/4.1.0>

课程资料中已经提供了redis-4.1.0.gem，直接上传安装即可，安装命令：

```
gem install redis-4.1.0.gem
```

进入redis安装目录，使用redis自带的集群管理脚本，执行命令：

```
# 进入redis安装包  
cd /root/redis-4.0.14/src/  
# 查看集群管理脚本  
ll *.rb  
# 使用集群管理脚本启动集群，下面命令中的1表示为每个主节点创建1个从节点  
./redis-trib.rb create --replicas 1 127.0.0.1:7001 127.0.0.1:7002 \  
127.0.0.1:7003 127.0.0.1:7004 127.0.0.1:7005 127.0.0.1:7006
```

效果如下：

```
>>> Creating cluster
```

北京市昌平区建材城西路金燕龙办公楼一层 电话：400-618-9090



```
192.168.200.129:7001
192.168.200.129:7002
192.168.200.129:7003
Adding replica 192.168.200.129:7005 to 192.168.200.129:7001
Adding replica 192.168.200.129:7006 to 192.168.200.129:7002
Adding replica 192.168.200.129:7004 to 192.168.200.129:7003
>>> Trying to optimize slaves allocation for anti-affinity
[WARNING] Some slaves are in the same host as their master
M: f0094f14b59c023acd38098336e2adcd3d434497 192.168.200.129:7001
  slots:0-5460 (5461 slots) master
M: 0eba44418d7e88f4d819f89f90da2e6e0be9c680 192.168.200.129:7002
  slots:5461-10922 (5462 slots) master
M: ac16c5545d9b099348085ad8b3253145912ee985 192.168.200.129:7003
  slots:10923-16383 (5461 slots) master
S: edc7a799e1cfd75e4d80767958930d86516ffc9b 192.168.200.129:7004
  replicates ac16c5545d9b099348085ad8b3253145912ee985
S: cbd415973b3e85d6f3ad967441f6bcb5b7da506a 192.168.200.129:7005
  replicates f0094f14b59c023acd38098336e2adcd3d434497
S: 40fdde45b16e1ac85c8a4c84db75b43978d1e4d2 192.168.200.129:7006
  replicates 0eba44418d7e88f4d819f89f90da2e6e0be9c680
Can I set the above configuration? (type 'yes' to accept): yes #注意选择为yes
>>> Nodes configuration updated
>>> Assign a different config epoch to each node
>>> Sending CLUSTER MEET messages to join the cluster
Waiting for the cluster to join..
>>> Performing Cluster Check (using node 192.168.200.129:7001)
M: f0094f14b59c023acd38098336e2adcd3d434497 192.168.200.129:7001
  slots:0-5460 (5461 slots) master
  1 additional replica(s)
M: ac16c5545d9b099348085ad8b3253145912ee985 192.168.200.129:7003
  slots:10923-16383 (5461 slots) master
  1 additional replica(s)
S: cbd415973b3e85d6f3ad967441f6bcb5b7da506a 192.168.200.129:7005
  slots: (0 slots) slave
  replicates f0094f14b59c023acd38098336e2adcd3d434497
S: 40fdde45b16e1ac85c8a4c84db75b43978d1e4d2 192.168.200.129:7006
  slots: (0 slots) slave
  replicates 0eba44418d7e88f4d819f89f90da2e6e0be9c680
M: 0eba44418d7e88f4d819f89f90da2e6e0be9c680 192.168.200.129:7002
  slots:5461-10922 (5462 slots) master
  1 additional replica(s)
S: edc7a799e1cfd75e4d80767958930d86516ffc9b 192.168.200.129:7004
  slots: (0 slots) slave
  replicates ac16c5545d9b099348085ad8b3253145912ee985
[OK] All nodes agree about slots configuration.
>>> Check for open slots...
>>> Check slots coverage...
[OK] All 16384 slots covered.
```

2.3.5 使用Redis集群

按照redis cluster的特点，它是去中心化的，每个节点都是对等的，所以连接哪个节点都可以获取和设置数据。

```
./redis-cli -h 192.168.200.129 -p 7001 -c
```

其中-c 一定要加，这个是redis集群连接时，进行节点跳转的参数。

连接到集群后可以设置一些值，可以看到这些值根据前面提到的哈希槽方式分散存储在不同的节点上了。