# 数据库高可用

## 实现说明

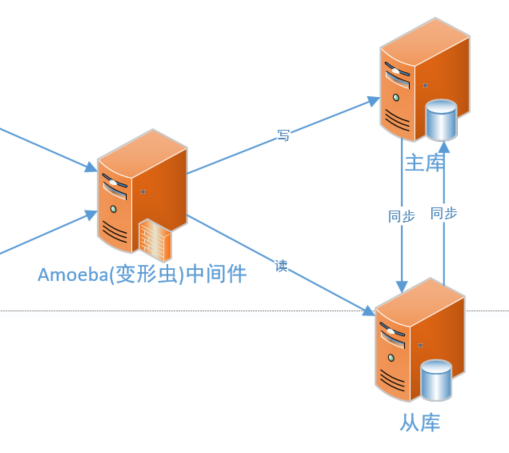
### 实现前提

问题说明:当数据库实现了高可用,但是由于操作的问题,最终导致了数据库数据不一致的问题.该问题在实际开发中不能出现.

核心问题:要想实现高可用,必须实现**数据的同步**.否则高可用无从谈起.

### 双机热备

说明:在实际开发过程中,小中型公司一般采用双机热备的形式.实现数据库的高可用.大型公司一般采用多个双机热备的形式.



### 双机热备规划

A库:主库 IP: 192.168.126.180 端口3306

B库:从库 IP: 192.168.126.163 端口3306

昨天: A库向B库同步数据

今天: B库向A库同步数据

**注意事项:只有用户直接操作的数据库才会修改二进制日志文件.**

### 实现双机实现

B库执行:

/\*我是B库 现在充当主库\*/

/\*1.检查当前主库的状态\*/

SHOW MASTER STATUS;

A库执行:

/\*A库,现在需要同步B的数据\*/

CHANGE MASTER TO

MASTER\_HOST="192.168.126.163",

MASTER\_PORT=3306,

MASTER\_USER="root",

MASTER\_PASSWORD="root",

MASTER\_LOG\_FILE="mysql-bin.000001",

MASTER\_LOG\_POS=380;

/\*2.开启主从同步的服务\*/

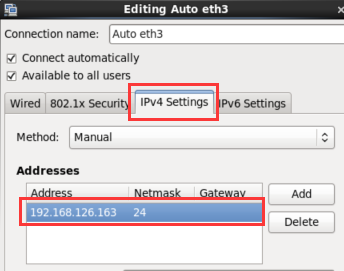
START SLAVE;

/\*3.检查主从同步的状态\*/

SHOW SLAVE STATUS;

3.挂载完成后,测试主从是否正常.

### 虚拟机设定静态IP地址



## Mycat

### Mycat介绍

网址: http://www.mycat.io/

 一个彻底开源的，面向企业应用开发的大数据库集群

 支持事务、ACID、可以替代MySQL的加强版数据库

 一个可以视为MySQL集群的企业级数据库，用来替代昂贵的Oracle集群

 一个融合内存缓存技术、NoSQL技术、HDFS大数据的新型SQL Server

 结合传统数据库和新型分布式数据仓库的新一代企业级数据库产品

 一个新颖的数据库中间件产品

### Mycat安装

1. 上传安装文件

tar -xvf Mycat-server-1.7.0-DEV-20170416134921-linux.tar.gz

### 编辑schema.xml配置文件

<?xml version="1.0"?>

<!DOCTYPE mycat:schema SYSTEM "schema.dtd">

<mycat:schema xmlns:mycat="http://io.mycat/">

<!--name属性是自定义的 dataNode表示数据库的节点信息-->

<schema name="jtdb" checkSQLschema="false" sqlMaxLimit="100" dataNode="jtdb"/>

<!--定义节点名称/节点主机/数据名称-->

<dataNode name="jtdb" dataHost="localhost1" database="jtdb" />

<!--参数介绍-->

<!--balance 0表示所有的读操作都会发往writeHost主机 -->

<!--1表示所有的读操作发往readHost和闲置的主节点中-->

<!--writeType=0 所有的写操作都发往第一个writeHost主机-->

<!--writeType=1 所有的写操作随机发往writeHost中-->

<!--dbType 表示数据库类型 mysql/oracle-->

<!--dbDriver="native" 固定参数 不变-->

<!--switchType=-1 表示不自动切换, 主机宕机后不会自动切换从节点-->

<!--switchType=1 表示会自动切换(默认值)如果第一个主节点宕机后,Mycat会进行3次心跳检测,如果3次都没有响应,则会自动切换到第二个主节点-->

<!--并且会更新/conf/dnindex.properties文件的主节点信息 localhost1=0 表示第一个节点.该文件不要随意修改否则会出现大问题-->

<dataHost name="localhost1" maxCon="1000" minCon="10" balance="1"

writeType="0" dbType="mysql" dbDriver="native" switchType="1" slaveThreshold="100">

<heartbeat>select 1</heartbeat>

<!--配置第一台主机主要进行写库操作,在默认的条件下Mycat主要操作第一台主机在第一台主机中已经实现了读写分离.因为默认写操作会发往137的数据库.读的操作默认发往141.如果从节点比较忙,则主节点分担部分压力.

-->

<writeHost host="hostM1" url="192.168.126.180:3306" user="root" password="root">

<!--读数据库-->

<readHost host="hostS1" url="192.168.126.163:3306" user="root" password="root" />

</writeHost>

<!--定义第二台主机 由于数据库内部已经实现了双机热备.-->

<!--Mycat实现高可用.当第一个主机137宕机后.mycat会自动发出心跳检测.检测3次.-->

<!--如果主机137没有给Mycat响应则判断主机死亡.则回启东第二台主机继续为用户提供服务.-->

<!--如果137主机恢复之后则处于等待状态.如果141宕机则137再次持续为用户提供服务.-->

<!--前提:实现双机热备.-->

<writeHost host="hostM2" url="192.168.126.163:3306" user="root" password="root">

<readHost host="hostS1" url="192.168.126.180:3306" user="root" password="root" />

</writeHost>

</dataHost>

</mycat:schema>

### 编辑server.xml

<!--用户标签-->

<user name="root">

<property name="password">root</property>

<!--与schema.xml中的配置相同 注意数据库的大小写-->

<property name="schemas">jtdb</property>

</user>

<user name="user">

<property name="password">user</property>

<property name="schemas">jtdb</property>

<property name="readOnly">true</property>

</user>

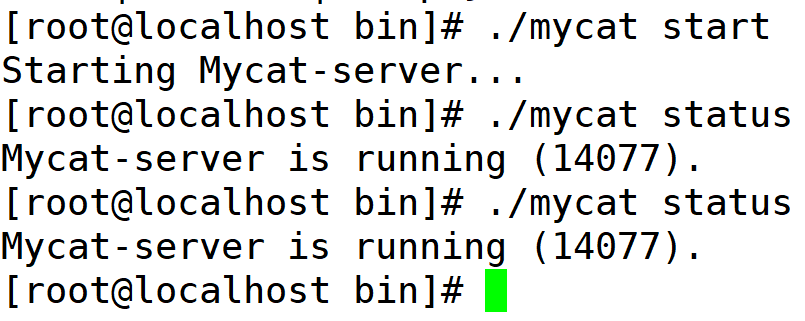
### Mycat命令

1. ./mycat start 启动mycat

2. ./mycat stop 关闭mycat

3. ./mycat restart 重启

4. ./mycat status 检测mycat状态



### 高可用测试

1. 将主库关闭.
2. 查询数据/修改数据
3. 重启主库.检查数据是否同步

## Mycat分库分表原理

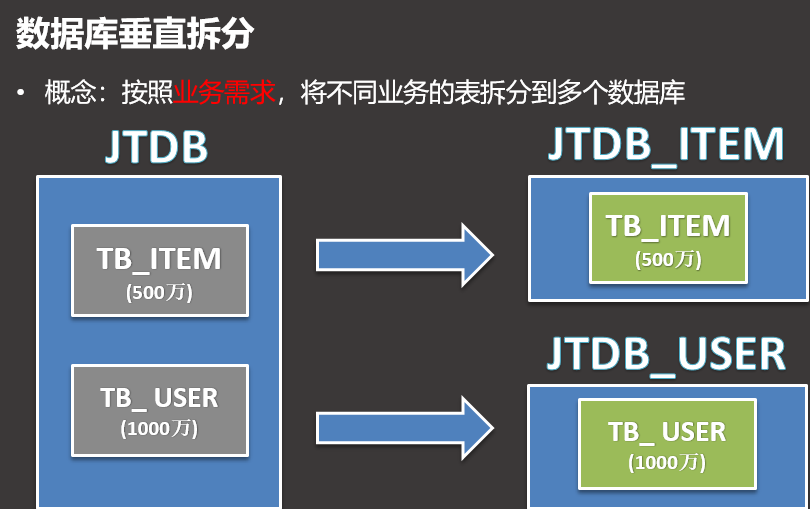
### 业务中遇到的问题

说明:如果在一个数据库中保存全部的数据,则可能会遇到数据的**瓶颈问题**.

核心问题:就是数据库中的数据太多了,更新操作/查询操作不能满足用户的需求.这时需要进行优化!!

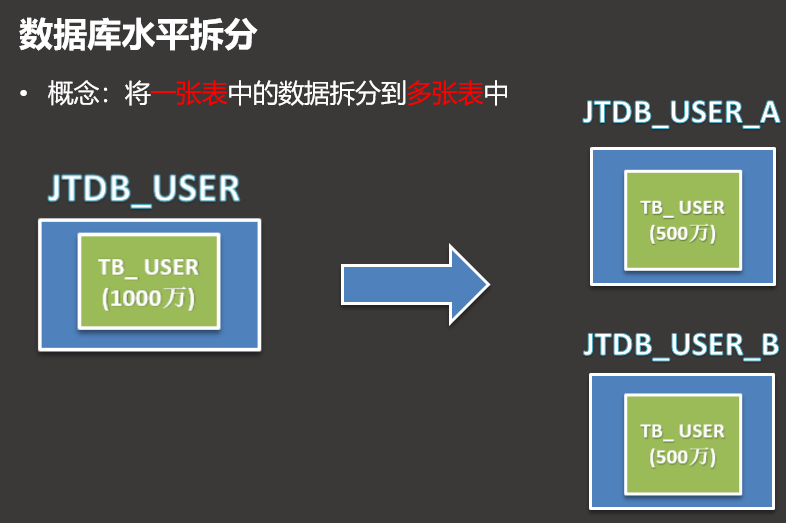
### 数据库的垂直拆分

核心:根据业务模块将数据库进行拆分

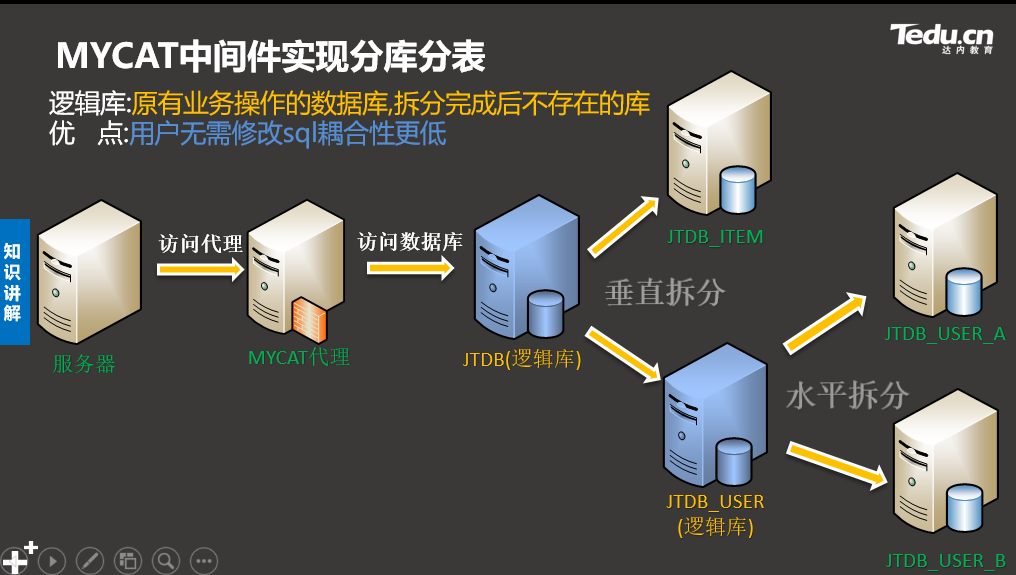


### 数据库的水平拆分

核心:将一个数据表的数据进行拆分



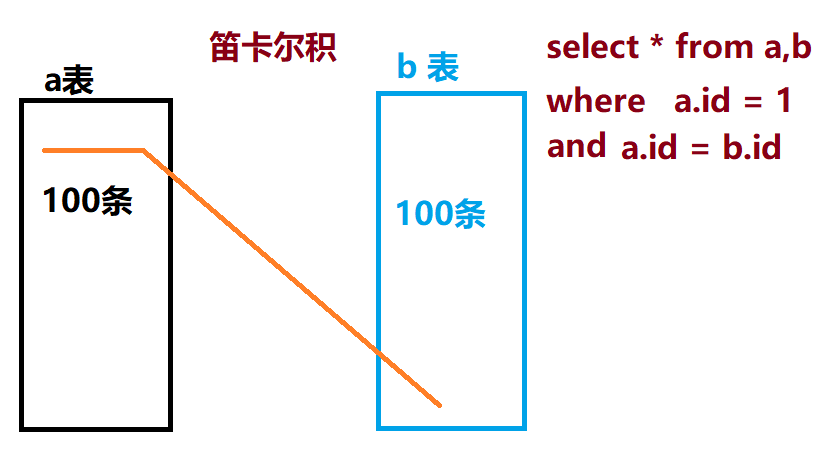
### 基于mycat实现分库分表



### 数据库如何优化

什么都不会:公司有专业运维人员,我没有考虑过

1. 优化sql语句(尽可能根据主键查询,和单表查询)
2. 创建索引/添加主键
3. 添加缓存(redis memcached),缓存数据在内存中
4. 提升硬件性能
5. 定期进行数据的转储
6. 分库分表(服务器数据量会增加,维护成本变高) 部分分库分表



# Redis学习

## 缓存机制

### 缓存机制是什么

添加缓存实际上是提高用户的查询的效率.缓存数据其实就是数据库中的数据.缓存机制其实就是有效的降低了用户访问物理设备的访问频次.

### 使用缓存需要解决哪些问题

1. 速度要快(运行在内存中,C语言)
2. 数据的一致性(当数据库数据发生改变时,同步更新缓存数据)
3. 如何维护缓存的大小(通过lru算法实现)

Lru:

[内存管理](https://baike.baidu.com/item/%E5%86%85%E5%AD%98%E7%AE%A1%E7%90%86/5633616)的一种页面置换算法，对于在内存中但又不用的[数据块](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%9D%97/107672)（内存块）叫做LRU，操作系统会根据哪些数据属于LRU而将其移出内存而腾出空间来加载另外的数据。

1. 缓存使用哪种数据结构??? K-V结构
2. 缓存数据如何持久化? 定期将缓存数据持久化到磁盘中.

### Redis介绍

Redis 是一个开源（BSD许可）的，内存中的数据结构存储系统，它可以用作**数据库、缓存和消息中间件**。 **它支持多种类型的数据结构，如**[**字符串（strings）**](http://www.redis.cn/topics/data-types-intro.html#strings)**，**[**散列（hashes）**](http://www.redis.cn/topics/data-types-intro.html#hashes)**，**[**列表（lists）**](http://www.redis.cn/topics/data-types-intro.html#lists)**，**[**集合（sets）**](http://www.redis.cn/topics/data-types-intro.html#sets)， [有序集合（sorted sets）](http://www.redis.cn/topics/data-types-intro.html#sorted-sets) 与范围查询， [bitmaps](http://www.redis.cn/topics/data-types-intro.html#bitmaps)， [hyperloglogs](http://www.redis.cn/topics/data-types-intro.html" \l "hyperloglogs) 和 [地理空间（geospatial）](http://www.redis.cn/commands/geoadd.html) 索引半径查询。 Redis 内置了 [复制（replication）](http://www.redis.cn/topics/replication.html)，[LUA脚本（Lua scripting）](http://www.redis.cn/commands/eval.html)， [LRU驱动事件（LRU eviction）](http://www.redis.cn/topics/lru-cache.html)，[事务（transactions）](http://www.redis.cn/topics/transactions.html) 和不同级别的 [磁盘持久化（persistence）](http://www.redis.cn/topics/persistence.html)， 并通过 [Redis哨兵（Sentinel）](http://www.redis.cn/topics/sentinel.html)和自动 [分区（Cluster）](http://www.redis.cn/topics/cluster-tutorial.html)提供高可用性（high availability）。

Redis速度:读11万次/秒 写8.6万次/秒,**redis每秒吞吐量10万/秒**

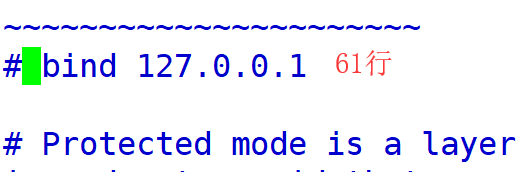
## Redis安装

1. 上传安装包
2. 解压文件
3. 编辑redis make
4. 安装redis make install

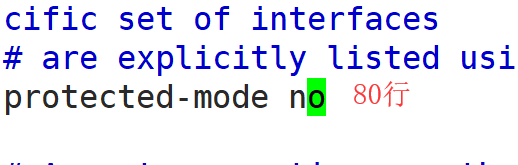
### 修改redis配置文件

说明:修改redis配置文件 vim redis.conf

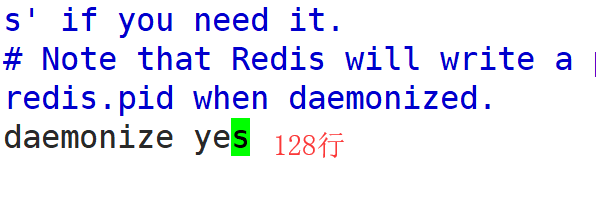
1. 关闭IP绑定



1. 关闭保护模式

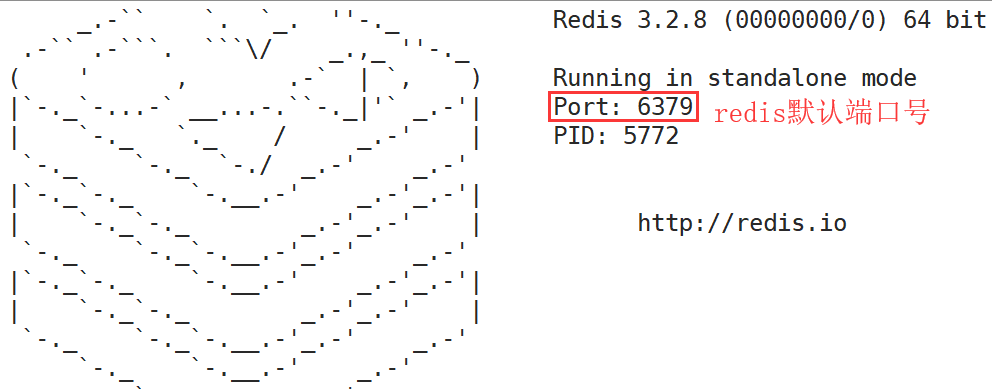


1. 开启后台启动

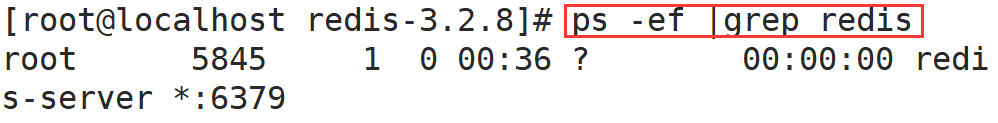


### Redis命令

1.启动命令(以后不用) redis-server



2.启动命令 redis-server redis.conf



3.关闭命令 redis-cli -p 6379 shutdown

4.客户端指令 redis-cli -p 6379

## Redis命令

### String类型

|  |  |  |
| --- | --- | --- |
| 命令 | 说明 | 案例 |
| set | 添加key-value | set username admin |
| get | 根据key获取数据 | get username |
| strlen | 获取key的长度 | strlen key |
| exists | 判断key是否存在 | exists name  返回1存在 0不存在 |
| del | 删除redis中的key | del key |
| Keys | 用于查询符合条件的key | keys \* 查询redis中全部的key  keys n?me 使用占位符获取数据  keys nam\* 获取nam开头的数据 |
| mset | 赋值多个key-value | mset key1 value1 key2 value2 key3 value3 |
| mget | 获取多个key的值 | mget key1 key2 |
| append | 对某个key的值进行追加 | append key value |
| type | 检查某个key的类型 | type key |
| select | 切换redis数据库 | select 0-15 redis中共有16个数据库 |
| flushdb | 清空单个数据库 | flushdb |
| flushall | 清空全部数据库 | flushall |
| incr | 自动加1 | incr key |
| decr | 自动减1 | decr key |
| incrby | 指定数值添加 | incrby 10 |
| decrby | 指定数值减 | decrby 10 |
| expire | 指定key的生效时间 单位秒 | expire key 20  key20秒后失效 |
| pexpire | 指定key的失效时间 单位毫秒 | pexpire key 2000  key 2000毫秒后失效 |
| ttl | 检查key的剩余存活时间 | ttl key |
| persist | 撤销key的失效时间 | persist key |

### Hash类型

说明:可以用散列类型保存对象和属性值

例子:User对象{id:2,name:小明,age:19}

|  |  |  |
| --- | --- | --- |
| 命令 | 说明 | 案例 |
| hset | 为对象添加数据 | hset key field value |
| hget | 获取对象的属性值 | hget key field |
| hexists | 判断对象的属性是否存在 | HEXISTS key field  1表示存在 0表示不存在 |
| hdel | 删除hash中的属性 | hdel user field [field ...] |
| hgetall | 获取hash全部元素和值 | HGETALL key |
| hkyes | 获取hash中的所有字段 | HKEYS key |
| hlen | 获取hash中所有属性的数量 | hlen key |
| hmget | 获取hash里面指定字段的值 | hmget key field [field ...] |
| hmset | 为hash的多个字段设定值 | hmset key field value [field value ...] |
| hsetnx | 设置hash的一个字段,只有当这个字段不存在时有效 | HSETNX key field value |
| hstrlen | 获取hash中指定key的长度 | HSTRLEN key field |
| hvals | 获取hash的所有值 | HVALS user |

### List类型

说明:Redis中的List集合是双端循环列表,分别可以从左右两个方向插入数据.

List集合可以当做队列使用,也可以当做栈使用

队列:存入数据的方向和获取数据的方向相反

栈:存入数据的方向和获取数据的方向相同

|  |  |  |
| --- | --- | --- |
| 命令 | 说明 | 案例 |
| lpush | 从队列的左边入队一个或多个元素 | LPUSH key value [value ...] |
| rpush | 从队列的右边入队一个或多个元素 | RPUSH key value [value ...] |
| lpop | 从队列的左端出队一个元素 | LPOP key |
| rpop | 从队列的右端出队一个元素 | RPOP key |
| lpushx | 当队列存在时从队列的左侧入队一个元素 | LPUSHX key value |
| rpushx | 当队列存在时从队列的右侧入队一个元素 | RPUSHx key value |
| lrange | 从列表中获取指定返回的元素 | LRANGE key start stop  Lrange key 0 -1 获取全部队列的数据 |
| lrem | 从存于 key 的列表里移除前 count 次出现的值为 value 的元素。 这个 count 参数通过下面几种方式影响这个操作：   * count > 0: 从头往尾移除值为 value 的元素。 * count < 0: 从尾往头移除值为 value 的元素。 * count = 0: 移除所有值为 value 的元素。 | LREM list -2 “hello” 会从存于 list 的列表里移除最后两个出现的 “hello”。  需要注意的是，如果list里没有存在key就会被当作空list处理，所以当 key 不存在的时候，这个命令会返回 0。 |
| Lset | 设置 index 位置的list元素的值为 value | LSET key index value |

### Redis事务命令

说明:redis中操作可以添加事务的支持.一项任务可以由多个redis命令完成,如果有一个命令失败导致入库失败时.需要实现事务回滚.

|  |  |  |
| --- | --- | --- |
| 命令 | 说明 | 案例 |
| multi | 标记一个事务开始 | 127.0.0.1:6379> MULTI  OK |
| exec | 执行所有multi之后发的命令 | 127.0.0.1:6379> EXEC  OK |
| discard | 丢弃所有multi之后发的命令 |  |

## Redis入门案例

### 导入jar包

<!-- jedis -->

<dependency>

<groupId>redis.clients</groupId>

<artifactId>jedis</artifactId>

<version>${jedis.version}</version>

</dependency>

<!--添加spring-datajar包 -->

<dependency>

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

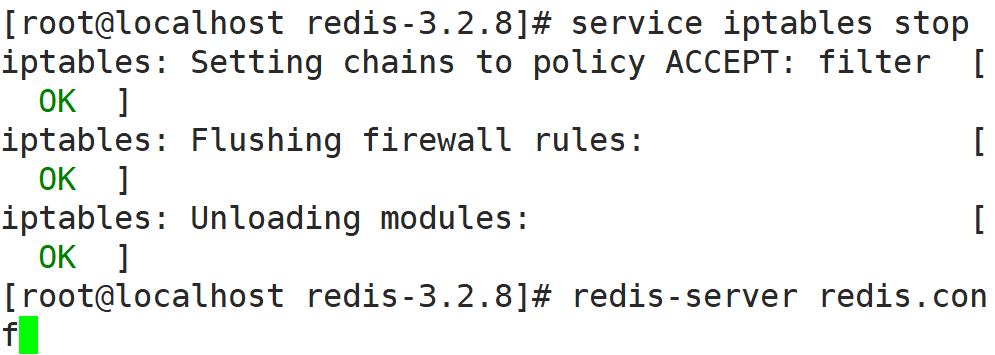
<artifactId>spring-data-redis</artifactId>

<version>1.4.1.RELEASE</version>

</dependency>

### 操作String类型数据

操作前,注意关闭防火墙



//1.操作String类型

@Test

**public** **void** testString() {

Jedis jedis =

**new** Jedis("192.168.126.174",6379);

jedis.set("1810","1810班");

String value = jedis.get("1810");

System.***out***.println("从redis中获取数据:"+value);

}