## Redis技术

### redis的下载与安装

#### 1.1.1.1 windows下的安装

* 直接在官网上下载：<https://redis.io/>
* 解压版本，不用安装即可使用

redis.windows.conf redis的配置文件

redis-server.exe redis的服务端

redis-cli.exe redis的客户端

#### 1.1.1.2 linux环境下的安装

* 安装 gcc wget命令

|  |
| --- |
| yum install -y gcc-c++  yum install -y wget |

* 下载redis

|  |
| --- |
| wget http://download.redis.io/releases/redis-3.0.4.tar.gz |

* 解压redis，并切换到redis解压目录，然后编译

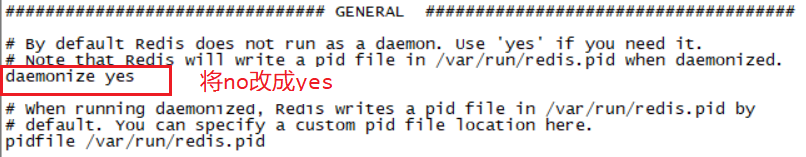
|  |
| --- |
| tar zxvf redis-3.0.4.tar.gz  cd redis-redis-3.0.4  make |

* 修改redis.conf开启守护进程模式

daemonize:yes:redis采用的是单进程多线程的模式。当redis.conf中选项daemonize设置成yes时，代表开启守护进程模式。在该模式下，redis会在后台运行，并将进程pid号写入至redis.conf选项pidfile设置的文件中，此时redis将一直运行，除非手动kill该进程。

daemonize:no: 当daemonize选项设置成no时，当前界面将进入redis的命令行界面，exit强制退出或者关闭连接工具(putty,xshell等)都会导致redis进程退出。

|  |
| --- |
| vim redis.conf |

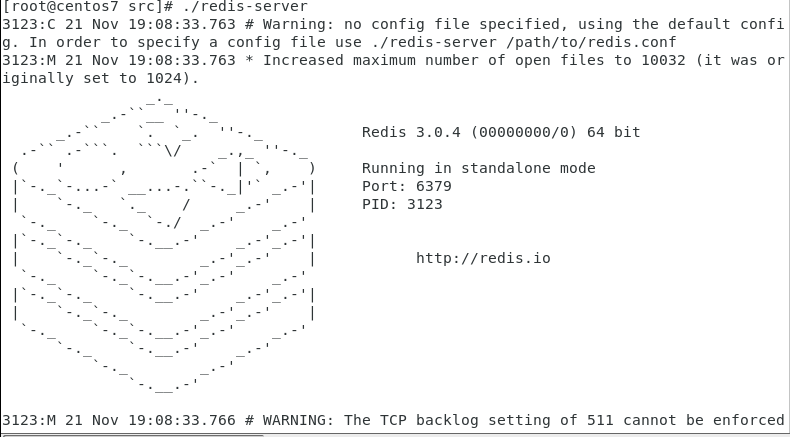


* 启动redis

进入redis的安装目录,启动redis

|  |
| --- |
| ./src reids-server |

如果出现下面效果，说明redis启动成功。



### 1.1.2 连接redis的客户端

注意: windows版本的启动方式:

使用cmd切换到redis的目录

启动服务端: redis-server.exe redis.windows.conf

启动客户端: redis-cli.exe

#### 1.1.2.1 redis自带客户端

redis安装以后，在src目录里面有一个redis-cli命令，这就是redis自带的客户端。我们执行这个命令，就可以使用redis客户端了。

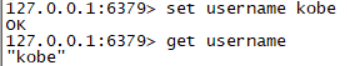
|  |
| --- |
| [root@centos7 redis-3.0.4]# cd src  [root@centos7 src]# ./redis-cli |

如果出现以下效果就可以使用自带客户端了。



测试：我们使用redis客户端，向redis存储一个数据，然后看能否获取成功

|  |
| --- |
| set username tiedan #存储数据  get username #获取数据 |

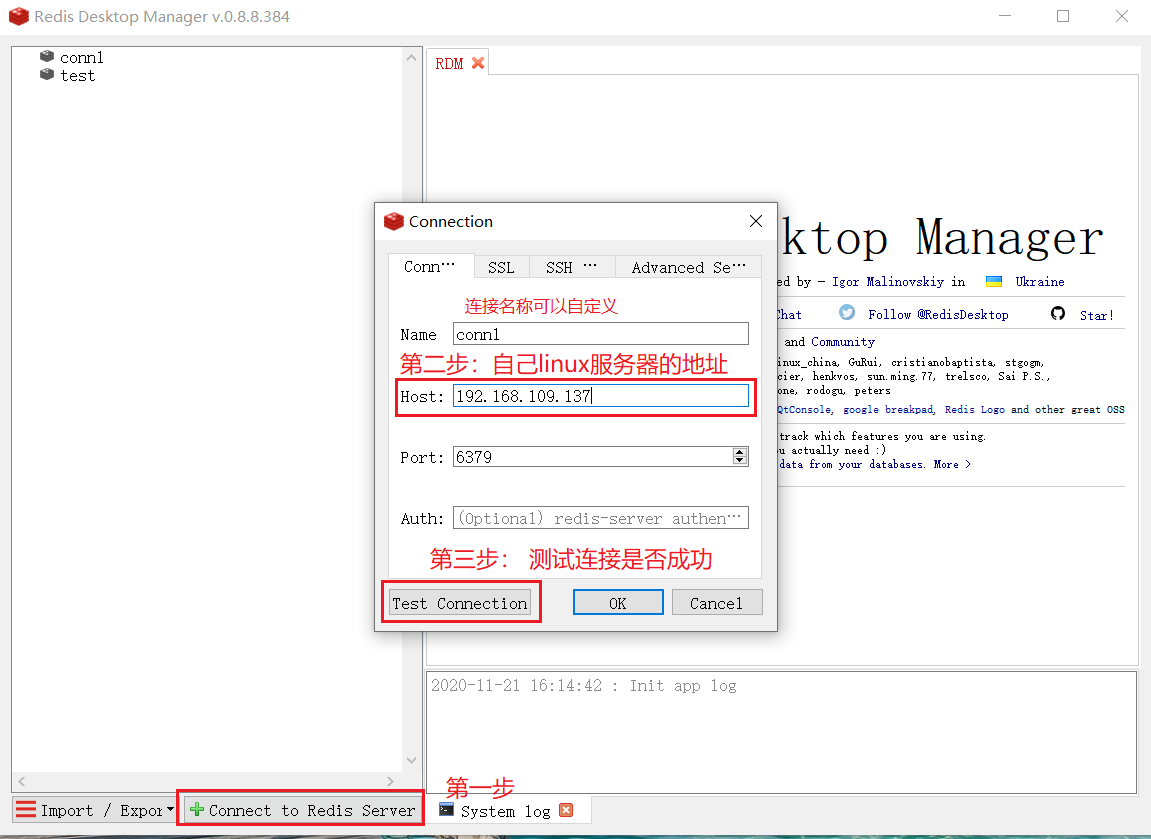


我们发现可以正常的存取数据，说明我们的客户端可以正常使用。

#### 1.1.2.2 第三方客户端

我们也可以使用第三方的图形化管理工具，来操作redis。





注意！！！！

此时测试连接，会出现连接被拒绝的情况，这是因为linux服务器没有开闭外界访问6379端口的权限。所以我们需要开启外界访问6379端口的权限。

输入firewall-cmd --query-port=6379/tcp，如果返回结果为no，那么证明6379端口确实没有开启。

输入firewall-cmd --add-port=6379/tcp，将6379端口开启，返回success。

然后再执行上一条命令，返回yes，证明端口已经成功开启。

### 1.1.3 redis多数据库

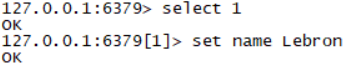
#### 1.1.3.1 redis实例

一个redis进程就是一个redis实例，一台服务器可以同时有多个redis实例，不同的redis实例提供不同的服务端口对外提供服务，每个redis实例之间互相影响。每个redis实例都包括自己的数据库，数据库中可以存储自己的数据。

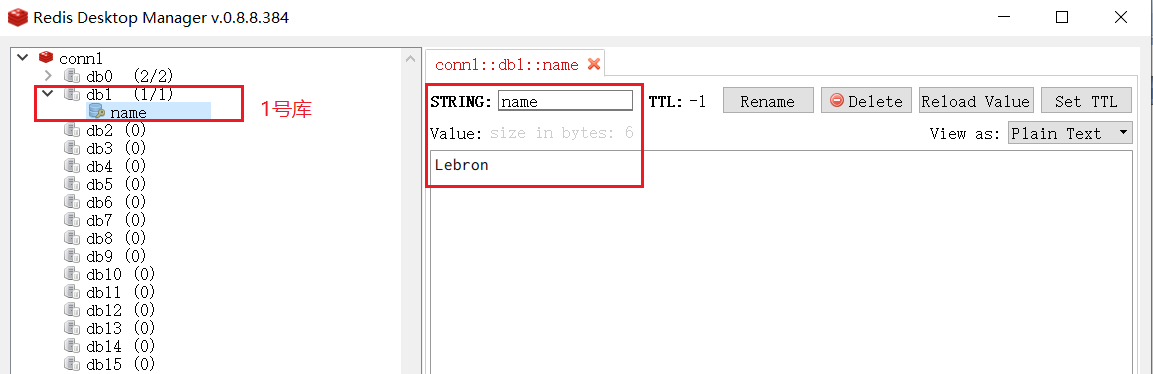
#### 1.1.3.2 多数据库测试

一个Redis实例可以包括多个数据库，客户端可以指定连接某个redis实例的哪个数据库，就好比一个mysql中创建多个数据库，客户端连接时指定连接哪个数据库。

一个redis实例最多可提供16个数据库，下标从0到15，客户端默认连接第0号数据库，也可以通过select选择连接哪个数据库，如下连接1号库：



我们通过图形化管理工具去查看：



### 1.1.4 redis的数据结构,及相关命令

#### 1.1.4.1 redis的数据结构

redis存储的数据是key,value格式的数据，其中key都是字符串，value有5种不同类型的数据结构。

Value值的数据类型：

1. string 字符串类型
2. hash 哈希类型
3. list 列表类型 linkedlist 有序支持重复元素
4. set 集合类型 无序 不支持重复元素
5. 有序集合类型 sortedset 有序的，不允许有重复的元素

#### 1.1.4.2 Redis的相关命令

* string: 字符串类型的数据

存储 set key

获取 get key

删除 del key

|  |
| --- |
| 127.0.0.1:6379> **set username kobe** 存储键为username的字符串数据 value值为kobe  OK  127.0.0.1:6379> **get username** 获取username的值  "kobe"  127.0.0.1:6379> set username lily 重新赋值  OK  127.0.0.1:6379> get username  "lily"  127.0.0.1:6379> **del username**  删除键  (integer) 1  127.0.0.1:6379> get username  127.0.0.1:6379> **strlen username** 获取字符串的长度  (integer) 9  127.0.0.1:6379> strlen name  (integer) 12  127.0.0.1:6379> **getrange name 1 5** 获取指定key对应值中的指定范围的子字符串 字符串的索引值从下标从0开始  "hangs"  127.0.0.1:6379> **setrange name 5 hello**  指定索引值范围的替换  (integer) 12  127.0.0.1:6379> get name  "zhanghellosi"  127.0.0.1:6379> **mset age 13 address usa score 99 email** [**zhansgan@qq.com**](mailto:zhansgan@qq.com) 批量的set字符串数据  OK  127.0.0.1:6379> **keys \*** 获取当前数据库实例中的所有key  1) "email"  2) "name"  3) "score"  4) "num"  5) "username"  6) "age"  7) "address"  127.0.0.1:6379> **mget name address age**  批量获取多个key的值  1) "zhanghellosi"  2) "usa"  3) "13"  127.0.0.1:6379> **incr num** 自增  (integer) 1  127.0.0.1:6379> get num  "1"  127.0.0.1:6379> **incr num** 自减  (integer) 2  127.0.0.1:6379> get num  "2"  127.0.0.1:6379> decr num  (integer) 1  127.0.0.1:6379> get num  "1"  127.0.0.1:6379> incrby num 2 按照指定数字增加  (integer) 3  127.0.0.1:6379> **set animal cat EX 20 设置key的过期时间 （秒为单位）**  OK  127.0.0.1:6379> get animal  "cat"  127.0.0.1:6379> **ttl animal 查看还剩多长时间过期**  (integer) 12 |

String类型数据的应用场景:

验证码的发送。一般会把手机的验证码以string类型的数据存储在redis里面，并给这个字符串设置过期时间。

String以json格式字符串保存对象(前提是这个对象的属性不会频繁的发生修改)

* 哈希类型 hash

存储： hset key field fieldvalue

获取： hget key field

获取所有： hgetall key

删除： hdel key field

|  |
| --- |
| 127.0.0.1:6379> **hset myhash name sunny**  通过myhash键 存储一个map结构的数据 name:sunny  (integer) 1  127.0.0.1:6379> **hset myhash password 345**  通过myhash键 存储一个map结构的数据 password 345  (integer) 1  127.0.0.1:6379> hget myhash name 通过myhash键 获取name字段的名称  "sunny"  127.0.0.1:6379> hget myhash password 通过myhash键 获取password字段的名称  "345"  127.0.0.1:6379> **hgetall myhash**  获取myhash键对应的所有map结构的数据  1) "name"  2) "sunny"  3) "password"  4) "345"  127.0.0.1:6379> **hdel myhash password** 删除myhash键对应的字段数据  (integer) 1  127.0.0.1:6379> hgetall myhash  1) "name"  2) "sunny"  127.0.0.1:6379> **hincrby user num 1** key值的增量  (integer) 2  127.0.0.1:6379> hget user num  "2"  127.0.0.1:6379> hincrby user num 2  (integer) 4  127.0.0.1:6379> hdecrby user num 2  (error) ERR unknown command 'hdecrby'  127.0.0.1:6379> hincrby user num -1 key值的减  (integer) 3  127.0.0.1:6379> **setnx sname lisi 如果key不存在，就赋值。如果key存在，什么都不做。**  (integer) 1  127.0.0.1:6379> get sname  "lisi"  127.0.0.1:6379> setnx sname wangwu  (integer) 0  127.0.0.1:6379> get sname  "lisi"  127.0.0.1:6379> **hkeys user** 获取所有的小key值  1) "username"  2) "age"  3) "num"  127.0.0.1:6379> **hlen user** 获取所有的小key数量  (integer) 3  127.0.0.1:6379> **hmset department id 1001 deptname sales location Beijing**  批量存储hash  OK  127.0.0.1:6379> **hmget department id deptname location** 批量获取  1) "1001"  2) "sales"  3) "beijing" |

应用场景:

用户的购物车信息，就可以使用hash的数据类型进行缓存。

哈希的数据结构是 key field value。在设计缓存购物车的时候。我们可以这样设计:

userid productId productNum

哈希数据类型也可以存放对象(适合频繁修改的属性)

* list 列表类型.可以从列表的头部或者尾部进行数据的添加

添加：

lpush key value 将元素从列表的左边开始添加

rpush key value 将元素从列表的右边开始添加

获取：

lrange key start end

删除：

lpop key: 删除列表最左边的元素，然后将其返回

rpop key: 删除列表最右边的元素，然后将其返回

|  |
| --- |
| 127.0.0.1:6379> **lpush mylist a 向mylist中的最左边追加一个元素a**  (integer) 1  127.0.0.1:6379> lpush mylist b  (integer) 2  127.0.0.1:6379> **rpush mylist c 向mylist中的最右边追加一个元素b**  (integer) 3  127.0.0.1:6379> rpush mylist d  (integer) 4  127.0.0.1:6379> **lrange mylist 0 -1 查询mylist中全部的元素 0 -1代表查询全部**  1) "b"  2) "a"  3) "c"  4) "d"  127.0.0.1:6379> **rpush mylist a 追加一个重复的元素a**  (integer) 5  127.0.0.1:6379> lrange mylist 0 -1  1) "b"  2) "a"  3) "c"  4) "d"  5) "a"  127.0.0.1:6379> |

|  |
| --- |
| 127.0.0.1:6379> **lpop mylist 删除列表的最左边的元素**  "b"  127.0.0.1:6379> lrange mylist 0 -1  1) "a"  2) "c"  3) "d"  4) "a"  127.0.0.1:6379> rpop mylist 删除列表最右边的元素  "a"  127.0.0.1:6379> lrange mylist 0 -1 查询列表中的所有元素  1) "a"  2) "c"  3) "d"  127.0.0.1:6379> **lindex mylist 0 根据下标获取指定的元素**  "b"  127.0.0.1:6379> lindex mylist 1  "a"  127.0.0.1:6379> lindex mylist 3  (nil)  127.0.0.1:6379> **llen mylist 获取列表中元素的个数**  (integer) 3  127.0.0.1:6379> **lrange mylist 0 2 根据指定的范围进行查询(列表的起始索引值为0)**  1) "b"  2) "a"  3) "c" |

应用场景:

商品的评论列表、文章的评论列表。(最新评论的信息总是在最前面显示 利用了list的 lpush 和lrange)

* set:不允许出现重复的元素

添加元素： sadd key value

获取元素： smembers key

删除元素： srem key value

|  |
| --- |
| 127.0.0.1:6379> **sadd myset a 向myset键里面添加value值a**  (integer) 1  127.0.0.1:6379> sadd myset b  (integer) 1  127.0.0.1:6379> **smembers myset 查询myset键里面的所有数据**  1) "a"  2) "b"  127.0.0.1:6379> sadd myset c  (integer) 1  127.0.0.1:6379> sadd myset m  (integer) 1  127.0.0.1:6379> smembers myset 元素添加进去的顺序和元素出来的顺序是不一样的 无序的  1) "a"  2) "m"  3) "c"  4) "b"  127.0.0.1:6379> sadd myset m 添加重复的元素 没有效果的  (integer) 0  127.0.0.1:6379> smembers myset  1) "a"  2) "m"  3) "c"  4) "b"  127.0.0.1:6379> **srem myset a 删除myset键对应的值**  (integer) 1  127.0.0.1:6379> srem myset m  (integer) 1  127.0.0.1:6379> smembers myset  1) "b"  2) "c"  127.0.0.1:6379> **sismember myset f 判断元素是否属于存在set集合中 1 true 0 false**  (integer) 1  127.0.0.1:6379> sismember myset z  (integer) 0  127.0.0.1:6379> sadd s1 12  (integer) 1  127.0.0.1:6379> sadd s1 13  (integer) 1  127.0.0.1:6379> sadd s1 15  (integer) 1  127.0.0.1:6379> sadd s1 18  (integer) 1  127.0.0.1:6379> sadd s2 12  (integer) 1  127.0.0.1:6379> sadd s2 13  (integer) 1  127.0.0.1:6379> sadd s2 22  (integer) 1  127.0.0.1:6379> sadd s2 25  (integer) 1  127.0.0.1:6379> **sdiff s1 s2 取 s1 和s2集合的差集**  1) "15"  2) "18"  127.0.0.1:6379> **sinter s1 s2 取s1 和 s2的交集**  1) "12"  2) "13"  127.0.0.1:6379> **sunion s1 s2 取s1 和 s2的并集**  1) "12"  2) "13"  3) "15"  4) "18"  127.0.0.1:6379> **scard s1 获取指定集合中元素的个数**  (integer) 4  127.0.0.1:6379> scard s2  (integer) 4  127.0.0.1:6379> **srandmember s1 随机获取集合中的一个元素**  "12"  127.0.0.1:6379> srandmember s1  "18"  127.0.0.1:6379> **sdiffstore s3 s1 s2 将s1和s2集合中的差集存储在s3集合里面**  (integer) 2  127.0.0.1:6379> smembers s3  1) "15"  2) "18"  127.0.0.1:6379> **sinterstore s4 s1 s2 将s1和s2集合中的交集存储在s4集合里面**  (integer) 2  127.0.0.1:6379> smembers s4  1) "12"  2) "13"  127.0.0.1:6379> **sunionstore s5 s1 s2 将s1和s2集合中的并集存储在s5集合里面**  (integer) 6  127.0.0.1:6379> smembers s5  1) "12"  2) "13"  3) "15"  4) "18"  5) "22"  6) "25" |

Set集合的应用场景:

点赞与取消点赞

点一次 点赞

再点一次 取消点赞

用户在点赞的时候，会将用户id存入到set集合。在点赞之前先需要判断当前的用户id是否在set集合中存在?

如果set集合中存在这个用户id，说明用户已经点过赞。如果继续点,会取消点赞(srem)。

如果set集合中不存在这个用户id。就允许点赞(将用户id存入到set集合中)。

* Sortedset:有序的集合类型，不允许出现重复的元素，且元素是有序的，每一个元素都会关联一个double类型的分数。redis正是通过这个分数来为集合中的成员进行从小到大的排序

添加：

zadd key score value

获取:

zrange key start end

删除：

zrem key value

|  |
| --- |
| 127.0.0.1:6379> zadd mysort 60 kobe 向键mysort里面添加元素kobe，给了一个排序的分数是60  (integer) 1  127.0.0.1:6379> zadd mysort 89 lily  (integer) 1  127.0.0.1:6379> zadd mysort 88 miiler  (integer) 1  127.0.0.1:6379> zadd mysort 34 durant  (integer) 1  127.0.0.1:6379> zrange mysort 0 -1 查询键mysort里面的所有元素(会按照分数进行从小到大的排序输出)  1) "durant"  2) "kobe"  3) "miiler"  4) "lily"  127.0.0.1:6379> zrange mysort 0 -1 withscores 查询键mysort对应的所有value值及其分数信息  1) "durant"  2) "34"  3) "kobe"  4) "60"  5) "miiler"  6) "88"  7) "lily"  8) "89"  127.0.0.1:6379> zrem mysort lily 删除mysort键里面对应的元素  (integer) 1  127.0.0.1:6379> zrange mysort 0 -1  1) "durant"  2) "kobe"  3) "miiler"  127.0.0.1:6379> |

排行榜的设计

### 1.1.5 Redis的持久化

* Redis是一个内存数据库，数据本质上还是保存在电脑的内存当中的，当redis服务器重启，数据就会丢失，我们可以将redis内存中的数据持久化的保存在电脑的硬盘文件中。
* Redis持久化的机制：
* RDB：默认的持久化的机制，在默认的情况下是不需要配置的。在一定的时间间隔中，会去监测到key值的变化情况，然后持久化数据

我们可以进入redis.windows.conf文件

# after 900 sec (15 min) if at least 1 key changed 15分钟之后如果监测到至少有1个key值发生变化，就进行持久化的存储

# after 300 sec (5 min) if at least 10 keys changed 5分钟之后，如果监测到至少有10个key值发生变化，就进行持久化的存储

# after 60 sec if at least 10000 keys changed 1分钟之后，如果监测到至少有10000个key值发生变化，就进行持久化存储

我们可以修改这些数据，来改变持久化数据的默认的规则

|  |
| --- |
| save 900 1  save 300 10  save 10 2 |

注意：修改完redis.Windows.conf配置文件之后，需要重启redis的服务器。需要在cmd命令行里面切换到对应的redis-servere.exe对应的目录中。调用命令: redis-servere.exe redis.Windows.conf 即可重启服务器

**dump.rdb 使用rdb的方式进行持久化存储，生成一个文件**

* AOF日志记录的方式，可以检测到每一条命令的操作。每一次命令操作之后，都会持久化数据。

我们可以进入redis.windows.conf文件，然后编辑该文件

appendonly no 修改成appendonly yes 开启AOF持久化数据的方式

# appendfsync always 每一次操作都进行持久化

appendfsync everysec 每隔1秒钟都进行持久化

# appendfsync no 从来不进行持久化

修改完redis.Windows.conf配置文件之后，需要重启redis的服务器。需要在cmd命令行里面切换到对应的redis-servere.exe对应的目录中。调用命令: redis-servere.exe redis.Windows.conf 即可重启服务器

appendonly.aof就是持久化的文件

<https://baijiahao.baidu.com/s?id=1654694618189745916&wfr=spider&for=pc>

### 使用jedis去操作redis数据库(重点)

Jedis:就是使用java去操作redis的技术

使用步骤:

1. 下载jedis的jar包



|  |
| --- |
| <**dependencies**>  *<!--jedis客户端依赖-->* <**dependency**>  <**groupId**>redis.clients</**groupId**>  <**artifactId**>jedis</**artifactId**>  <**version**>2.9.0</**version**>  </**dependency**>  <**dependency**>  <**groupId**>junit</**groupId**>  <**artifactId**>junit</**artifactId**>  <**version**>4.13</**version**>  </**dependency**> </**dependencies**> |

1. 使用jedis

* 操作string

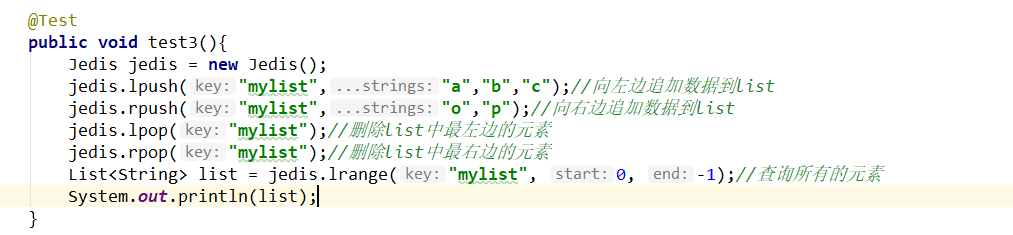


|  |
| --- |
| *//创建一个键为age的值 存储的时间是10秒钟，10秒钟之后，该键及其值会自定丢失* jedis.setex(**"age"**,10,**"password"**); |

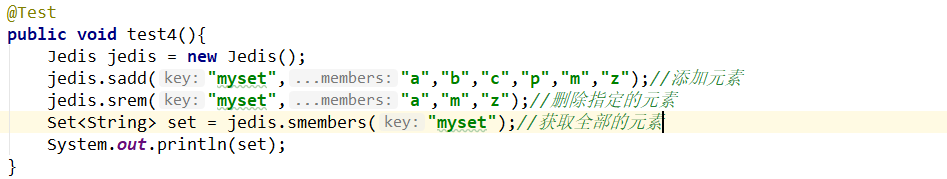
* 操作hash



* 操作list



* 集合类型set 不允许重复元素



* 有序集合类型 sortedSet 不允许重复元素，并且元素是根据一个double类型的分数进行从小到大进行排序的



### jedis的连接池



* 抽取出一个通过连接池获取连接对象的工具类

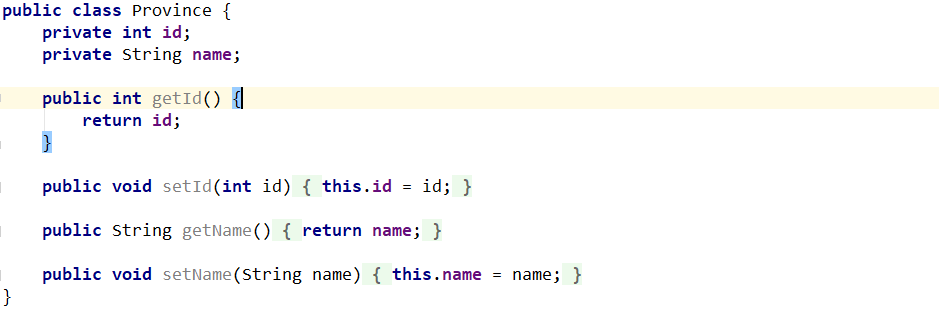
定义一个配置连接池的properties文件，然后放置在src目录下面



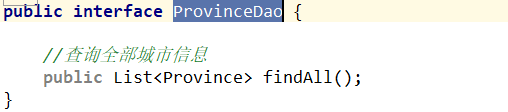
### 综合案例

需求：实现下拉框内容的加载，要求使用ajax和json配合mysql数据库+redis缓存

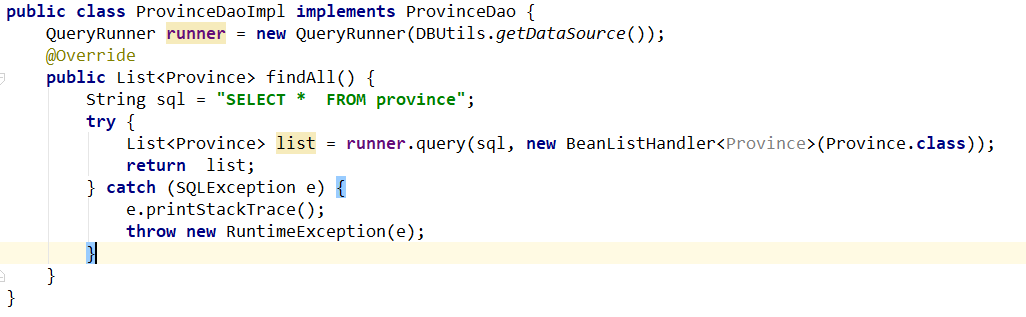
* 实体类



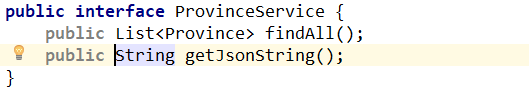
* Dao



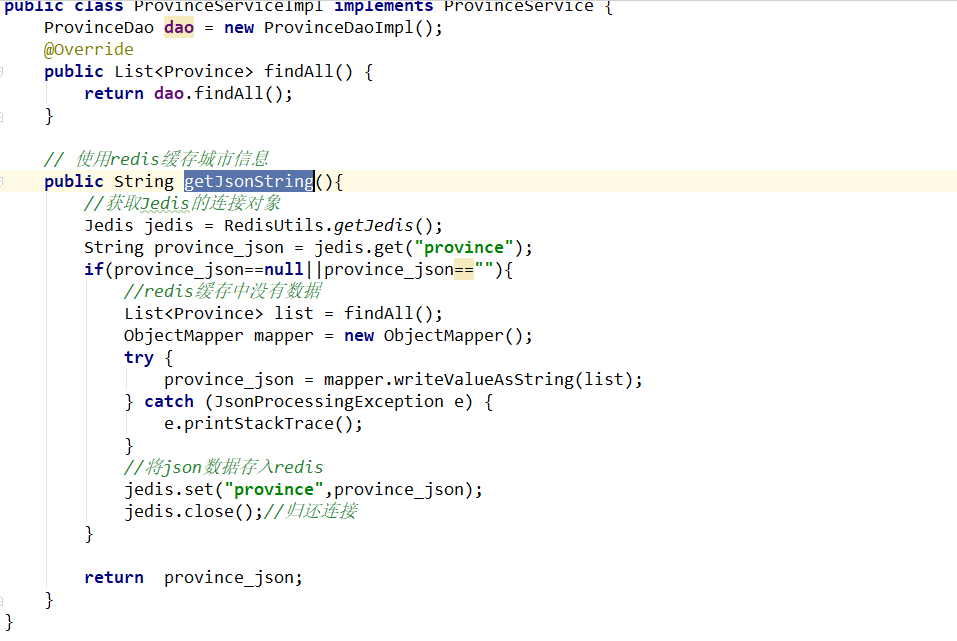
* Dao实现类



* Service



* Service实现类



* Servlet



* 工具类

略

* 前台页面



### 1.1.5 spring整合redis

#### 1.1.5.1 创建maven工程

创建maven工程，导入相关依赖。

|  |
| --- |
| <**dependencies**>  <**dependency**>  <**groupId**>org.springframework</**groupId**>  <**artifactId**>spring-context</**artifactId**>  <**version**>5.2.0.RELEASE</**version**>  </**dependency**>  <**dependency**>  <**groupId**>redis.clients</**groupId**>  <**artifactId**>jedis</**artifactId**>  <**version**>2.9.0</**version**>  </**dependency**>  <**dependency**>  <**groupId**>org.springframework</**groupId**>  <**artifactId**>spring-test</**artifactId**>  <**version**>5.2.0.RELEASE</**version**>  </**dependency**>  <**dependency**>  <**groupId**>junit</**groupId**>  <**artifactId**>junit</**artifactId**>  <**version**>4.13</**version**>  </**dependency**> </**dependencies**> |

#### 1.1.5.2 创建配置文件

在spring的配置文件里面，管理JedisPoolConfig JedisPool这两个bean

* 创建redis.properties配置文件

|  |
| --- |
| *#最大分配的对象数* **redis.pool.maxTotal**=**200** *#最大能够保持idel状态的对象数* **redis.pool.maxIdle**=**50 redis.pool.minIdle**=**10 redis.pool.maxWaitMillis**=**20000** |

* 定义spring的配置文件

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>* <**beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns:context="http://www.springframework.org/schema/context"  xsi:schemaLocation="http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans.xsd http://www.springframework.org/schema/context https://www.springframework.org/schema/context/spring-context.xsd"**>   *<!--导入外部的properties配置文件-->* <**context:property-placeholder location="classpath:redis.properties"**></**context:property-placeholder**>   *<!--管理redisPoolConfig-->* <**bean id="jedisPoolConfig" class="redis.clients.jedis.JedisPoolConfig"**>  <**property name="maxTotal" value="${redis.pool.maxTotal}"**></**property**>  <**property name="maxIdle" value="${redis.pool.maxIdle}"**></**property**>  <**property name="minIdle" value="${redis.pool.minIdle}"**></**property**>  <**property name="maxWaitMillis" value="${redis.pool.maxWaitMillis}"**></**property**>  </**bean**>   *<!--管理redisPool-->* <**bean id="jedisPool" class="redis.clients.jedis.JedisPool"**>  <**constructor-arg name="poolConfig" ref="jedisPoolConfig"**></**constructor-arg**>  <**constructor-arg name="host" value="127.0.0.1"**></**constructor-arg**>  <**constructor-arg name="port" value="6379"**></**constructor-arg**>  </**bean**> </**beans**> |

#### 1.1.5.3 测试

|  |
| --- |
| @RunWith(SpringJUnit4ClassRunner.**class**) @ContextConfiguration(locations = {**"classpath:applicationContext.xml"**}) **public class** TestRedis {   @Autowired  JedisPool **jedisPool**;   @Test  **public void** testJedisPool(){  System.***out***.println(**jedisPool**);  }   @Test  **public void** testString(){  Jedis jedis = **jedisPool**.getResource();  jedis.set(**"spring"**,**"spring"**);  jedis.close();  } } |