NOSQL数据库-Redis

学习目标：

1. 能够理解nosql的概念
2. 能够说出redis的常用数据类型
3. 能够使用redis的string操作命令
4. 能够使用redis的hash操作命令
5. 能够使用redis的list操作命令
6. 能够使用redis的set操作命令
7. 能够说出redis的两种持久化机制
8. 能够使用jedis对redis进行操作

第1章 NOSQL概述

1.1 什么是NOSQL

NoSQL(NoSQL = Not Only SQL)，意即“不仅仅是SQL”，是一项全新的数据库理念，泛指非关系型的数据库。

1.2 为什么需要NOSQL

随着互联网的高速崛起，网站的用户群的增加，访问量的上升，传统数据库上都开始出现了性能瓶颈，web程序不再仅仅专注在功能上，同时也在追求性能。所以NOSQL数据库应运而上，具体表现为对如下三高问题的解决：

- High performance - 对数据库高并发读写的需求

  web2.0网站要根据用户个性化信息来实时生成动态页面和提供动态信息，所以基本上无法使用动态页面静态化技术，因此数据库并发负载非常高，往往要达到每秒上万次读写请求。关系数据库应付上万次SQL查询还勉强顶得住，但是应付上万次SQL写数据请求，硬盘IO就已经无法承受了。其实对于普通的BBS网站，往往也存在对高并发写请求的需求，例如网站的实时统计在线用户状态，记录热门帖子的点击次数，投票计数等，因此这是一个相当普遍的需求。

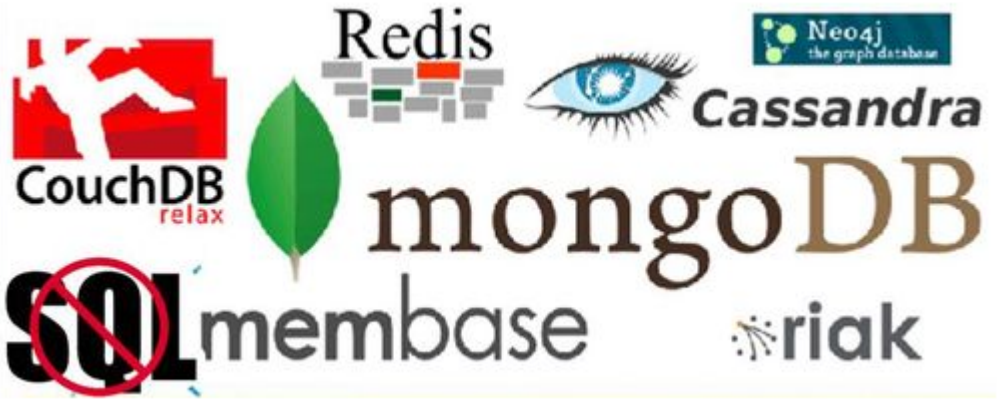
- Huge Storage - 对海量数据的高效率存储和访问的需求

  类似Facebook，twitter，Friendfeed这样的SNS网站，每天用户产生海量的用户动态，以Friendfeed为例，一个月就达到了2.5亿条用户动态，对于关系数据库来说，在一张2.5亿条记录的表里面进行SQL查询，效率是极其低下乃至不可忍受的。再例如大型web网站的用户登录系统，例如腾讯，盛大，动辄数以亿计的帐号，关系数据库也很难应付。

- High Scalability && High Availability- 对数据库的高可扩展性和高可用性的需求

  在基于web的架构当中，数据库是最难进行横向扩展的，当一个应用系统的用户量和访问量与日俱增的时候，你的数据库却没有办法像web server和app server那样简单的通过添加更多的硬件和服务节点来扩展性能和负载能力。对于很多需要提供24小时不间断服务的网站来说，对数据库系统进行升级和扩展是非常痛苦的事情，往往需要停机维护和数据迁移，为什么数据库不能通过不断的添加服务器节点来实现扩展呢？

1.3 主流的NOSQL产品



- 键值(Key-Value)存储数据库

  相关产品： Tokyo Cabinet/Tyrant、Redis、Voldemort、Berkeley DB

  典型应用： 内容缓存，主要用于处理大量数据的高访问负载。

  数据模型： 一系列键值对

  优势： 快速查询

  劣势： 存储的数据缺少结构化

- 列存储数据库

  相关产品：Cassandra, HBase, Riak

  典型应用：分布式的文件系统

  数据模型：以列簇式存储，将同一列数据存在一起

  优势：查找速度快，可扩展性强，更容易进行分布式扩展

  劣势：功能相对局限

- 文档型数据库

  相关产品：CouchDB、MongoDB

  典型应用：Web应用（与Key-Value类似，Value是结构化的）

  数据模型： 一系列键值对

  优势：数据结构要求不严格

  劣势： 查询性能不高，而且缺乏统一的查询语法

- 图形(Graph)数据库

  相关数据库：Neo4J、InfoGrid、Infinite Graph

  典型应用：社交网络

  数据模型：图结构

  优势：利用图结构相关算法。

  劣势：需要对整个图做计算才能得出结果，不容易做分布式的集群方案。

1.4 NOSQL的特点

在大数据存取上具备关系型数据库无法比拟的性能优势，例如：

- 易扩展

NoSQL数据库种类繁多，但是一个共同的特点都是去掉关系数据库的关系型特性。数据之间无关系，这样就非常容易扩展。也无形之间，在架构的层面上带来了可扩展的能力。

- 大数据量，高性能

  NoSQL数据库都具有非常高的读写性能，尤其在大数据量下，同样表现优秀。这得益于它的无关系性，数据库的结构简单。

- 灵活的数据模型

  NoSQL无需事先为要存储的数据建立字段，随时可以存储自定义的数据格式。而在关系数据库里，增删字段是一件非常麻烦的事情。如果是非常大数据量的表，增加字段简直就是一个噩梦。这点在大数据量的Web2.0时代尤其明显。

- 高可用

  NoSQL在不太影响性能的情况，就可以方便的实现高可用的架构。比如Cassandra，HBase模型，通过复制模型也能实现高可用。

第2章 Redis概述

2.1 什么是Redis

Redis是用C语言开发的一个开源的高性能键值对（key-value）数据库，官方提供测试数据，50个并发执行100000个请求,读的速度是110000次/s,写的速度是81000次/s ，且Redis通过提供多种键值数据类型来适应不同场景下的存储需求，目前为止Redis支持的键值数据类型如下：

- 字符串类型 string
- 散列类型 hash
- 列表类型 list
- 集合类型 set
- 有序集合类型 sortedset

2.2 redis的应用场景

- 缓存（数据查询、短连接、新闻内容、商品内容等等）
- 聊天室的在线好友列表
- 任务队列。（秒杀、抢购、12306等等）
- 应用排行榜
- 网站访问统计
- 数据过期处理（可以精确到毫秒
- 分布式集群架构中的session分离

第3章 window版Redis的安装与使用

3.1 windows版Redis的下载

官方提倡使用Linux版的Redis，所以官网值提供了Linux版的Redis下载，我们可以从GitHub上下载window版的Redis，具体链接地址如下：

- 官网下载地址：http://redis.io/download

- github下载地址：https://github.com/MSOpenTech/redis/tags

在今天的课程资料中提供的下载完毕的window版本的Redis：

3.2 window版Redis的目录结构

解压Redis压缩包后，见到如下目录机构：

目录或文件	作用
redis-benchmark	性能测试工具
redis-check-aof	AOF文件修复工具
redis-check-dump	RDB文件检查工具（快照持久化文件）
redis-cli	命令行客户端
redis-server	redis服务器启动命令
redis.windows.conf	redis核心配置文件

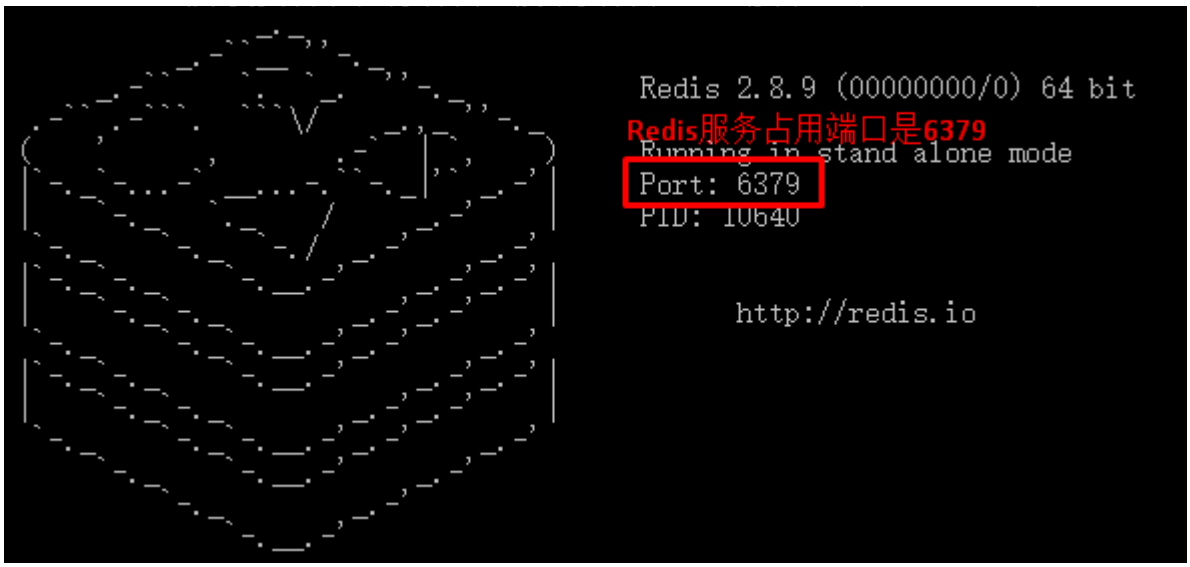
3.3 window版Redis的安装与启动

3.3.1 window版Redis的安装

window版的安装及其简单，解压Redis压缩包完成即安装完毕

3.3.2 window版Redis的启动与关闭

双击Redis目录中redis-server.exe可以启动redis服务，Redis服务占用的端口是6379



关闭Redis的控制台窗口就可以关闭Redis服务

3.4 window版Redis的使用

双击Redis目录中redis-cli.exe可以启动redis客户端

第4章 Redis的数据类型

4.1 Redis的5种数据类型

redis是一种高级的key-value的存储系统，其中value支持五种数据类型：

- **字符串（String）**
- **哈希（hash）**
- **字符串列表（list）**
- **字符串集合（set）**
- 有序字符串集合（sorted set）

  在日常开发中主要使用比较多的有字符串、哈希、字符串列表、字符串集合四种类型，其中最为常用的是字符串类型。

关于key的定义，注意如下几点：

- key不要太长，最好不要操作1024个字节，这不仅会消耗内存还会降低查找效率
- key不要太短，如果太短会降低key的可读性
- 在项目中，key最好有一个统一的命名规范

4.2 字符串类型string

4.2.1 字符串类型string概述

字符串类型是Redis中最为基础的数据存储类型，它在Redis中是二进制安全的，这便意味着该类型存入和获取的数据相同。在Redis中字符串类型的Value最多可以容纳的数据长度是512M。

4.2.2 字符串类型string常用命令

- **set key value**

  设定key持有指定的字符串value，如果该key存在则进行覆盖操作。总是返回"OK"

```
127.0.0.1:6379> set company "itcast"
OK
127.0.0.1:6379>
```

- **get key**

  获取key的value。如果与该key关联的value不是String类型，redis将返回错误信息，因为get命令只能用于获取String value；如果该key不存在，返回(nil)。

```
127.0.0.1:6379> set name "itcast"
OK
127.0.0.1:6379> get name
"itcast"
```

- **del key**

  删除指定key

```
127.0.0.1:6379> del name
(integer) 1
127.0.0.1:6379> get name
(nil)
```

4.3 哈希类型hash

4.3.1 哈希类型hash概述

Redis中的Hash类型可以看成具有String Key和String Value的map容器。所以该类型非常适合于存储值对象的信息。如Username、Password和Age等。如果Hash中包含很少的字段，那么该类型的数据也将仅占用很少的磁盘空间。每一个Hash可以存储4294967295个键值对。

4.3.2 哈希类型hash常用命令

- **hset key field value**

  为指定的key设定field/value对（键值对）。

```
127.0.0.1:6379> hset myhash username haohao
(integer) 1
127.0.0.1:6379>
```

- **hget key field**

  返回指定的key中的field的值

```
127.0.0.1:6379> hset myhash username haohao
(integer) 1
127.0.0.1:6379> hget myhash username
"haohao"
```

- **hdel key field [field ...]**

  可以删除一个或多个字段，返回值是被删除的字段个数

```
127.0.0.1:6379> hdel myhash username
(integer) 1
127.0.0.1:6379> hget myhash username
(nil)
127.0.0.1:6379>
```

4.4 列表类型list

4.4.1 列表类型list概述

在Redis中，List类型是按照插入顺序排序的字符串链表。和数据结构中的普通链表一样，我们可以在其头部(left)和尾部(right)添加新的元素。在插入时，如果该键并不存在，Redis将为该键创建一个新的链表。与此相反，如果链表中所有的元素均被移除，那么该键也将会被从数据库中删除。List中可以包含的最大元素数量是4294967295

4.4.2 列表类型list

- **lpush key values[value1 value2...]**

  在指定的key所关联的list的头部插入所有的values，如果该key不存在，该命令在插入的之前创建一个与该key关联的空链表，之后再向该链表的头部插入数据。插入成功，返回元素的个数。

```
127.0.0.1:6379> lpush mylist a b c
(integer) 3
127.0.0.1:6379>
```

- **lpop key**

  返回并弹出指定的key关联的链表中的第一个元素，即头部元素。如果该key不存在，返回nil；若key存在，则返回链表的头部元素。

```
127.0.0.1:6379> lpush mylist a b c
(integer) 3
127.0.0.1:6379> lpop mylist
"c"
127.0.0.1:6379> lpop mylist
"b"
```

- **rpop key**

  从尾部弹出元素。

```
127.0.0.1:6379> lpush mylist a b c
(integer) 3
127.0.0.1:6379> rpop mylist
"a"
```

4.5 集合类型set

4.5.1 集合类型set

在Redis中，我们可以将Set类型看作为没有排序的字符集合，和List类型一样，我们也可以在该类型的数据值上执行添加、删除或判断某一元素是否存在等操作。需要说明的是，这些操作的时间复杂度为O(1)，即常量时间内完成次操作。Set可包含的最大元素数量是4294967295，和List类型不同的是，Set集合中不允许出现重复的元素。

4.5.2 集合类型set的常用命令

- **sadd key values[value1、value2...]**

  向set中添加数据，如果该key的值已有则不会重复添加

```
127.0.0.1:6379> sadd myset a b c
(integer) 3
```

- **smembers key**

  获取set中所有的成员

```
127.0.0.1:6379> sadd myset a b c
(integer) 3
127.0.0.1:6379> smembers myset
1) "c"
2) "a"
3) "b"
```

- **srem key members[member1、member2...]**

  删除set中指定的成员

```
127.0.0.1:6379> srem myset a b
(integer) 2
127.0.0.1:6379> smembers myset
1) "c"
127.0.0.1:6379>
```

第5章 Redis的通用命令

- **keys pattern**

  获取所有与pattern匹配的key，返回所有与该key匹配的keys。*表示任意一个或多个字符，?表示任意一个字符

```
127.0.0.1:6379> keys *
1) "company"
2) "mylist"
3) "myhash"
4) "myset"
```

- **del key1 key2...**

  删除指定的key

```
127.0.0.1:6379> del company
(integer) 1
```

- **exists key**

  判断该key是否存在，1代表存在，0代表不存在

```
127.0.0.1:6379> exists compnay
(integer) 0
127.0.0.1:6379> exists mylist
(integer) 1
127.0.0.1:6379>
```

- **type key**

  获取指定key的类型。该命令将以字符串的格式返回。 返回的字符串为string、list、set、hash，如果key不存在返回none

```
127.0.0.1:6379> type company
string
127.0.0.1:6379> type mylist
list
127.0.0.1:6379> type myset
set
127.0.0.1:6379> type myhash
hash
127.0.0.1:6379>
```

第6章 Redis的持久化

6.1 Redis持久化概述

Redis的高性能是由于其将所有数据都存储在了内存中，为了使Redis在重启之后仍能保证数据不丢失，需要将数据从内存中同步到硬盘中，这一过程就是持久化。Redis支持两种方式的持久化，一种是RDB方式，一种是AOF方式。可以单独使用其中一种或将二者结合使用。

- RDB持久化（默认支持，无需配置）

  该机制是指在指定的时间间隔内将内存中的数据集快照写入磁盘。

- AOF持久化

  该机制将以日志的形式记录服务器所处理的每一个写操作，在Redis服务器启动之初会读取该文件来重新构建数据库，以保证启动后数据库中的数据是完整的。

- 无持久化

  我们可以通过配置的方式禁用Redis服务器的持久化功能，这样我们就可以将Redis视为一个功能加强版的memcached了。

- redis可以同时使用RDB和AOF

6.2 RDB持久化机制

6.2.1 RDB持久化机制优点

- 一旦采用该方式，那么你的整个Redis数据库将只包含一个文件，这对于文件备份而言是非常完美的。比如，你可能打算每个小时归档一次最近24小时的数据，同时还要每天归档一次最近30天的数据。通过这样的备份策略，一旦系统出现灾难性故障，我们可以非常容易的进行恢复。

- 对于灾难恢复而言，RDB是非常不错的选择。因为我们可以非常轻松的将一个单独的文件压缩后再转移到其它存储介质上

- 性能最大化。对于Redis的服务进程而言，在开始持久化时，它唯一需要做的只是fork（分叉）出子进程，之后再由子进程完成这些持久化的工作，这样就可以极大的避免服务进程执行IO操作了。

相比于AOF机制，如果数据集很大，RDB的启动效率会更高

6.2.2 RDB持久化机制缺点

- 如果你想保证数据的高可用性，即最大限度的避免数据丢失，那么RDB将不是一个很好的选择。因为系统一旦在定时持久化之前出现宕机现象，此前没有来得及写入磁盘的数据都将丢失。
- 由于RDB是通过fork子进程来协助完成数据持久化工作的，因此，如果当数据集较大时，可能会导致整个服务器停止服务几百毫秒，甚至是1秒钟

6.2.3 RDB持久化机制的配置

在redis.windows.conf配置文件中有如下配置：

```
################################ SNAPSHOTTING  ################################
#
# Save the DB on disk:
#
#   save <seconds> <changes>
#
#   Will save the DB if both the given number of seconds and the given
#   number of write operations against the DB occurred.
#
#   In the example below the behaviour will be to save:
#   after 900 sec (15 min) if at least 1 key changed
#   after 300 sec (5 min) if at least 10 keys changed
#   after 60 sec if at least 10000 keys changed
#
#   Note: you can disable saving at all commenting all the "save" lines.
#
#   It is also possible to remove all the previously configured save
#   points by adding a save directive with a single empty string argument
#   like in the following example:
#
#   save ""

save 900 1
save 300 10
save 60 10000
```

其中，上面配置的是RDB方式数据持久化时机：

关键字	时间(秒)	key修改数量	解释
save	900	1	每900秒(15分钟)至少有1个key发生变化，则dump内存快照
save	300	10	每300秒(5分钟)至少有10个key发生变化，则dump内存快照
save	60	10000	每60秒(1分钟)至少有10000个key发生变化，则dump内存快照

6.3 AOF持久化机制

6.3.1 AOF持久化机制优点

- 该机制可以带来更高的数据安全性，即数据持久性。Redis中提供了3中同步策略，即每秒同步、每修改同步和不同步。事实上，每秒同步也是异步完成的，其效率也是非常高的，所差的是一旦系统出现宕机现象，那么这一秒钟之内修改的数据将会丢失。而每修改同步，我们可以将其视为同步持久化，即每次发生的数据变化都会被立即记录到磁盘中。可以预见，这种方式在效率上是最低的。至于无同步，无需多言，我想大家都能正确的理解它。
- 由于该机制对日志文件的写入操作采用的是append模式，因此在写入过程中即使出现宕机现象，也不会破坏日志文件中已经存在的内容。然而如果我们本次操作只是写入了一半数据就出现了系统崩溃问题，不用担心，在Redis下一次启动之前，我们可以通过redis-check-aof工具来帮助我们解决数据一致性的问题。
- 如果日志过大，Redis可以自动启用rewrite机制。即Redis以append模式不断的将修改数据写入到老的磁盘文件中，同时Redis还会创建一个新的文件用于记录此期间有哪些修改命令被执行。因此在进行rewrite切换时可以更好的保证数据安全性。
- AOF包含一个格式清晰、易于理解的日志文件用于记录所有的修改操作。事实上，我们也可以通过该文件完成数据的重建

6.3.2 AOF持久化机制缺点

- 对于相同数量的数据集而言，AOF文件通常要大于RDB文件
- 根据同步策略的不同，AOF在运行效率上往往会慢于RDB。总之，每秒同步策略的效率是比较高的，同步禁用策略的效率和RDB一样高效。

6.3.3 AOF持久化机制配置

6.3.3.1 开启AOF持久化

```
############################## APPEND ONLY MODE ###############################

# By default Redis asynchronously dumps the dataset on disk. This mode is
# good enough in many applications, but an issue with the Redis process or
# a power outage may result into a few minutes of writes lost (depending on
# the configured save points).
#
# The Append Only File is an alternative persistence mode that provides
# much better durability. For instance using the default data fsync policy
# (see later in the config file) Redis can lose just one second of writes in a
# dramatic event like a server power outage, or a single write if something
# wrong with the Redis process itself happens, but the operating system is
# still running correctly.
#
# AOF and RDB persistence can be enabled at the same time without problems.
# If the AOF is enabled on startup Redis will load the AOF, that is the file
# with the better durability guarantees.
#
# Please check http://redis.io/topics/persistence for more information.

appendonly no
```

将appendonly修改为yes，开启aof持久化机制，默认会在目录下产生一个appendonly.aof文件

6.3.3.2 AOF持久化时机

```
# appendfsync always
appendfsync everysec
# appendfsync no
```

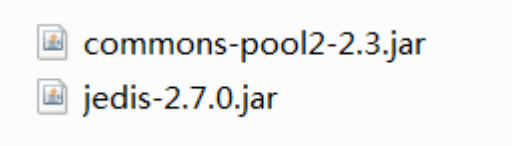
上述配置为aof持久化的时机，解释如下：

关键字	持久化时机	解释
appendfsync	always	每执行一次更新命令，持久化一次
appendfsync	everysec	每秒钟持久化一次
appendfsync	no	不持久化

第7章 Jedis的基本使用

7.1 jedis的介绍

Redis不仅是使用命令来操作，现在基本上主流的语言都有客户端支持，比如java、C、C#、C++、php、Node.js、Go等。 在官方网站里列一些Java的客户端，有Jedis、Redisson、Jredis、JDBC-Redis、等其中官方推荐使用Jedis和Redisson。 在企业中用的最多的就是Jedis，Jedis同样也是托管在github上，地址：https://github.com/xetorthio/jedis。下载jedis解压后得到jar包如下：



7.2 jedis的基本操作

7.2.1 jedis常用API

方法	解释
new Jedis(host, port)	创建jedis对象，参数host是redis服务器地址，参数port是redis服务端口
set(key,value)	设置字符串类型的数据
get(key)	获得字符串类型的数据
hset(key,field,value)	设置哈希类型的数据
hget(key,field)	获得哈希类型的数据
lpush(key,values)	设置列表类型的数据
lpop(key)	列表左面弹栈
rpop(key)	列表右面弹栈
del(key)	删除指定的key

7.2.2 jedis的基本操作

```
@Test
public void testJedisSingle(){
    //1 设置ip地址和端口
    Jedis jedis = new Jedis("localhost", 6379);
    //2 设置数据
    jedis.set("name", "itheima");
    //3 获得数据
    String name = jedis.get("name");
    System.out.println(name);
    //4 释放资源
    jedis.close();
}
```

7.3 jedis连接池的使用

7.3.1 jedis连接池的基本概念

jedis连接资源的创建与销毁是很消耗程序性能，所以jedis为我们提供了jedis的池化技术，jedisPool在创建时初始化一些连接资源存储到连接池中，使用jedis连接资源时不需要创建，而是从连接池中获取一个资源进行redis的操作，使用完毕后，不需要销毁该jedis连接资源，而是将该资源归还给连接池，供其他请求使用。

7.3.2 jedisPool的基本使用

```
@Test
public void testJedisPool(){
    //1 获得连接池配置对象，设置配置项
    JedisPoolConfig config = new JedisPoolConfig();
    // 1.1 最大连接数
```

```
        config.setMaxTotal(30);
        // 1.2  最大空闲连接数
        config.setMaxIdle(10);

        //2 获得连接池
        JedisPool jedisPool = new JedisPool(config, "localhost", 6379);

        //3 获得核心对象
        Jedis jedis = null;
        try {
            jedis = jedisPool.getResource();

            //4 设置数据
            jedis.set("name", "itcast");
            //5 获得数据
            String name = jedis.get("name");
            System.out.println(name);

        } catch (Exception e) {
            e.printStackTrace();
        } finally{
            if(jedis != null){
                jedis.close();
            }
            // 虚拟机关闭时，释放pool资源
            if(jedisPool != null){
                jedisPool.close();
            }
        }
    }
```

7.4 案例-编写jedis连接池工具类

JedisUtils.java

```
package com.itheima.utils;

import java.util.ResourceBundle;

import redis.clients.jedis.Jedis;
import redis.clients.jedis.JedisPool;
import redis.clients.jedis.JedisPoolConfig;

public class JedisUtils {

    private static JedisPoolConfig poolConfig = null;
    private static JedisPool jedisPool = null;

    private static Integer maxTotal = null;
    private static Integer maxIdle = null;
    private static String host = null;

    private static Integer port = null;
```

```
    static{

        //读取配置文件 获得参数值
        ResourceBundle rb = ResourceBundle.getBundle("jedis");
        maxTotal = Integer.parseInt(rb.getString("jedis.maxTotal"));
        maxIdle = Integer.parseInt(rb.getString("jedis.maxIdle"));
        port = Integer.parseInt(rb.getString("jedis.port"));
        host = rb.getString("jedis.host");

        poolConfig = new JedisPoolConfig();
        poolConfig.setMaxTotal(maxTotal);
        poolConfig.setMaxIdle(maxIdle);
        jedisPool = new JedisPool(poolConfig,host,port);
    }

    public static Jedis getJedis(){
        Jedis jedis = jedisPool.getResource();
        return jedis;
    }

}
```

jedis.properties

```
jedis.host=localhost
jedis.port=6379
jedis.maxTotal=30
jedis.maxIdle=10
```