

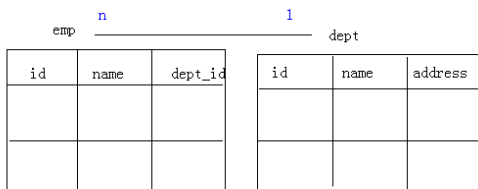
今日内容

1. redis

1. 概念
2. 下载安装
3. 命令操作

1. 数据结构
4. 持久化操作
5. 使用Java客户端操作redis

关系型数据库: mysql、oracle...

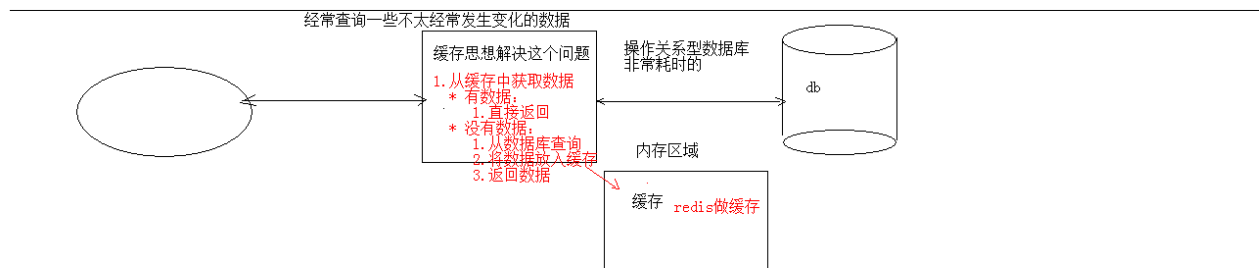


1. 数据之间有关联关系
2. 数据存储在硬盘的文件上

非关系型数据库 (NoSQL): redis、hbase...

存储key : value
name:zhangsan
age:23

1. 数据之间没有关联关系
2. 数据存储在内存中



Redis

概念： redis是一款高性能的NoSQL系列的非关系型数据库

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。
- 随着互联网web2.0网站的兴起，传统的关系数据库在应付web2.0网站，特别是超大规模和高并发的SNS类型的web2.0纯动态网站已经显得力不从心，暴露了很多难以克服的问题，而非关系型的数据库则由于其本身的特点得到了非常迅速的发展。NoSQL数据库的产生就是为了解决大规模数据集合多重数据种类带来的挑战，尤其是大数据应用难题。

1. NOSQL和关系型数据库比较

◦ 优点:

1. 成本: **nosql**数据库简单易部署, 基本都是开源软件, 不需要像使用**oracle**那样花费大量成本购买使用, 相比关系型数据库价格便宜。
2. 查询速度: **nosql**数据库将数据存储于缓存之中, 关系型数据库将数据存储于硬盘中, 自然查询速度远不及**nosql**数据库。
3. 存储数据的格式: **nosql**的存储格式是**key,value**形式、文档形式、图片形式等等, 所以可以存储基础类型以及对象或者是集合等各种格式, 而数据库则只支持基础类型。
4. 扩展性: 关系型数据库有类似**join**这样的多表查询机制的限制导致扩展很艰难。

◦ 缺点:

1. 维护的工具和资料有限, 因为**nosql**是属于新的技术, 不能和关系型数据库10几年的技术同日而语。
2. 不提供对**sql**的支持, 如果不支持**sql**这样的工业标准, 将产生一定用户的学习和使用成本。
3. 不提供关系型数据库对事务的处理。

2. 非关系型数据库的优势:

1. 性能**NOSQL**是基于键值对的, 可以想象成表中的主键和值的对应关系, 而且不需要经过**SQL**层的解析, 所以性能非常高。
2. 可扩展性同样也是因为基于键值对, 数据之间没有耦合性, 所以非常容易水平扩展。

3. 关系型数据库的优势:

1. 复杂查询可以用**SQL**语句方便的在一个表以及多个表之间做非常复杂的数据查询。
2. 事务支持使得对于安全性能很高的数据访问要求得以实现。对于这两类数据库, 对方的优势就是自己的弱势, 反之亦然。

4. 总结

- 关系型数据库与**NoSQL**数据库并非对立而是互补的关系, 即通常情况下使用关系型数据库, 在适合使用**NoSQL**的时候使用**NoSQL**数据库,
- 让**NoSQL**数据库对关系型数据库的不足进行弥补。
- 一般会将数据存储于关系型数据库中, 在**nosql**数据库中备份存储关系型数据库的数据

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。








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：1. 字符串类型 string 2. 哈希类型 hash 3. 列表类型 list 4. 集合类型 set 5. 有序集合类型 sortedset

1.3.1 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

下载安装

1. 官网: <https://redis.io>, 国外服务器
2. 中文网: <http://www.redis.net.cn/>
3. 解压直接可以使用:
 - redis.windows.conf: 配置文件
 - redis-cli.exe: redis的客户端
 - redis-server.exe: redis服务器端

 redis.windows.conf	2018/7/10 19:25	CONF 文件	28 KB
 redis-benchmark.exe	2014/6/16 15:55	应用程序	368 KB
 redis-check-aof.exe	2014/6/16 15:55	应用程序	284 KB
 redis-check-dump.exe	2014/6/16 15:55	应用程序	295 KB
 redis-cli.exe	2014/6/16 15:55	应用程序	404 KB
 redis-server.exe	2014/6/16 15:55	应用程序	1,120 KB
 stdout	2018/7/10 19:23	文件	0 KB

命令操作

redis的数据结构:


- redis存储的是: key,value格式的数据,其中key都是字符串,value有5种不同的数据结构
 - value的数据结构: 1) 字符串类型 string 2) 哈希类型 hash (map格式) 3) 列表类型 list (linkedlist格式。支持重复元素) 4) 集合类型 set (不允许重复元素) 5) 有序集合类型 sortedset(不允许重复元素,且元素有顺序)

- 1) 字符串类型 string
- 2) 哈希类型 hash : map格式
- 3) 列表类型 list : linkedlist格式
- 4) 集合类型 set :
- 5) 有序集合类型 sortedset

key	value				
mystring	zhangsan				
myhash	<table><tr><td>name</td><td>ls</td></tr><tr><td>age</td><td>24</td></tr></table>	name	ls	age	24
name	ls				
age	24				
mylist	<table><tr><td>zs</td><td>ls</td><td>ww</td></tr></table>	zs	ls	ww	
zs	ls	ww			
myset	<table><tr><td>zs</td><td>ls</td><td>ww</td></tr></table>	zs	ls	ww	
zs	ls	ww			
mysortedset	<table><tr><td>zs</td><td>ls</td><td>ww</td></tr></table>	zs	ls	ww	
zs	ls	ww			

字符串类型 string

1. 存储: set key value 127.0.0.1:6379> set username zhangsan OK
2. 获取: get key 127.0.0.1:6379> get username "zhangsan"
3. 删除: del key 127.0.0.1:6379> del age (integer) 1

 D:\redis-2.8.9\redis-cli.exe

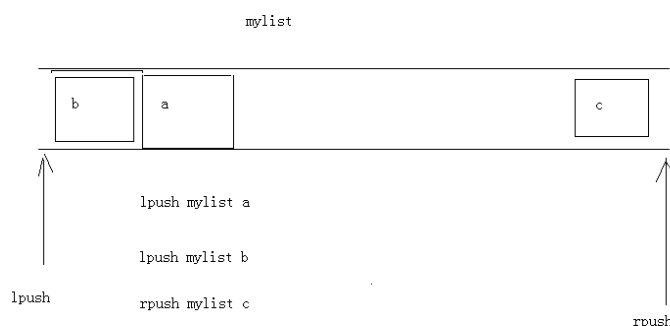
```
127.0.0.1:6379> set username zhangsan
OK
127.0.0.1:6379> get username
"zhangsan"
127.0.0.1:6379> del username
(integer) 1
127.0.0.1:6379> get username
(nil)
127.0.0.1:6379> _
```

哈希类型 hash

1. 存储: hset key field value 127.0.0.1:6379> hset myhash username lisi (integer) 1
127.0.0.1:6379> hset myhash password 123 (integer) 1
2. 获取:
 - hget key field: 获取指定的field对应的值
 - hgetall key: 获取所有的field和value
3. 删除: hdel key field

```
D:\redis-2.8.9\redis-cli.exe
127.0.0.1:6379> hset myhash username zhangsan
(integer) 1
127.0.0.1:6379> hset myhash password 123
(integer) 1
127.0.0.1:6379> hgetall myhash
1) "username"
2) "zhangsan"
3) "password"
4) "123"
127.0.0.1:6379> hget myhash username
"zhangsan"
127.0.0.1:6379> hdel myhash password
(integer) 1
127.0.0.1:6379> hgetall myhash
1) "username"
2) "zhangsan"
```

列表类型 list:可以添加一个元素到列表的头部（左边）或者尾部（右边）



1. 添加:
 1. lpush key value: 将元素加入列表左表
 2. rpush key value: 将元素加入列表右边
2. 获取:
 - lrange key start end : 范围获取,start-->end, 当end为-1时为获取所有
3. 删除:
 - lpop key: 删除列表最左边的元素, 并将元素返回
 - rpop key: 删除列表最右边的元素, 并将元素返回

```
D:\redis-2.8.9\redis-cli.exe
127.0.0.1:6379> lpush mylist a
(integer) 1
127.0.0.1:6379> lpush mylist b
(integer) 2
127.0.0.1:6379> rpush mylist c
(integer) 3
127.0.0.1:6379> lrange mylist 0 -1
1) "b"
2) "a"
3) "c"
127.0.0.1:6379> lpop mylist
"b"
127.0.0.1:6379> lpop mylist
"a"
127.0.0.1:6379>
```

这个允许存储相同的元素，不会自动排序

集合类型 set：不允许重复元素

1. 存储：sadd key value
2. 获取：smembers key:获取set集合中所有元素
3. 删除：srem key value:删除set集合中的某个元素

```
D:\redis-2.8.9\redis-cli.exe
127.0.0.1:6379> sadd myset a
(integer) 1
127.0.0.1:6379> sadd myset a
(integer) 0
127.0.0.1:6379> smembers myset
1) "a"
127.0.0.1:6379> sadd myset b c d
(integer) 3
127.0.0.1:6379> smembers myset
1) "d"
2) "a"
3) "c"
4) "b"
127.0.0.1:6379> srem myset a
(integer) 1
127.0.0.1:6379> smembers myset
1) "d"
2) "c"
3) "b"
127.0.0.1:6379>
```

不能存储重复数据

一次插入多条数据

不可以存储相同的数据，也不能自动排序

有序集合类型 sortedset

不允许重复元素，且元素有顺序。每个元素都会关联一个double类型的分数。redis正是通过分数来为集合中的成员进行从小到大的排序。

1. 存储：zadd key score value
2. 获取：zrange key start end [withscores]
3. 删除：zrem key value

```
D:\redis-2.8.9\redis-cli.exe
127.0.0.1:6379> zadd mysort 60 zhangsan
(integer) 1
127.0.0.1:6379> zadd mysort 50 lisi
(integer) 1
127.0.0.1:6379> zadd mysort 80 wangwu
(integer) 1
127.0.0.1:6379> zrange mysort 0 -1
1) "lisi"
2) "zhangsan"
3) "wangwu"
127.0.0.1:6379> zrange mysort 0 -1 withscores
1) "lisi"
2) "50"
3) "zhangsan"
4) "60"
5) "wangwu"
6) "80"
127.0.0.1:6379> zrem mysort lisi
(integer) 1
127.0.0.1:6379> zrange mysort 0 -1
1) "zhangsan"
2) "wangwu"
127.0.0.1:6379>
```

通用命令









1. keys * : 查询所有的键
2. type key : 获取键对应的value的类型
3. del key : 删除指定的key value

```
127.0.0.1:6379> keys *
1) "myset"
2) "mysort"
127.0.0.1:6379> type myset
set
127.0.0.1:6379> del myset
(integer) 1
127.0.0.1:6379> keys *
1) "mysort"
127.0.0.1:6379>
```

持久化

1. redis是一个内存数据库，当redis服务器重启，获取电脑重启，数据会丢失，我们可以将redis内存中的数据持久化保存到硬盘的文件中。
2. redis持久化机制：
 1. RDB：默认方式，不需要进行配置，默认就使用这种机制，推荐使用

- 在一定的间隔时间中，检测key的变化情况，然后持久化数据

 redis.windows.conf	2018/7/10 19:25	CONF 文件	28 KB
 redis-benchmark.exe	2014/6/16 15:55	应用程序	368 KB
 redis-check-aof.exe	2014/6/16 15:55	应用程序	284 KB
 redis-check-dump.exe	2014/6/16 15:55	应用程序	295 KB
 redis-cli.exe	2014/6/16 15:55	应用程序	404 KB
 RedisQFork_8028.dat	2019/5/24 10:13	KMPlayer.dat	8,301,056
 redis-server.exe	2014/6/16 15:55	应用程序	1,120 KB
 stdout	2018/7/10 19:23	文件	0 KB

```
D:\redis-2.8.9\redis.windows.conf - Notepad++
文件(F) 编辑(E) 搜索(S) 视图(V) 编码(M) 语言(L) 设置(T) 工具(O) 宏(M) 运行(R) 插件(P) 窗口(W) ?
FilterDesktop7.java  FilterDesktop8.java  FilterDesktop9.java  FindRedisServer.java  07-regist.html  RedisSystemSettings.ini  redis.windows.conf
88 # after 60 sec if at least 10000 keys changed
89 #
90 # Note: you can disable saving at all commenting all the "save" lines.
91 #
92 # It is also possible to remove all the previously configured save
93 # points by adding a save directive with a single empty string argument
94 # like in the following example:
95 #
96 # save ""
97 #
98 save 900 1
99 save 300 10
100 save 60 10000
101 #
102 # By default Redis will stop accepting writes if RDB snapshots are enabled
103 # (at least one save point) and the latest background save failed.
104 # This will make the user aware (in an hard way) that data is not persisting
105 # on disk properly, otherwise chances are that no one will notice and some
106 # distaster will happen.
107 #
108 # If the background saving process will start working again Redis will
109 # automatically allow writes again.
110 #
111 # However if you have setup your proper monitoring of the Redis server
112 # and persistence, you may want to disable this feature so that Redis will
113 # continue to work as usually even if there are problems with disk,
114 # permissions, and so forth.
115 stop-writes-on-bgsave-error yes
116
```

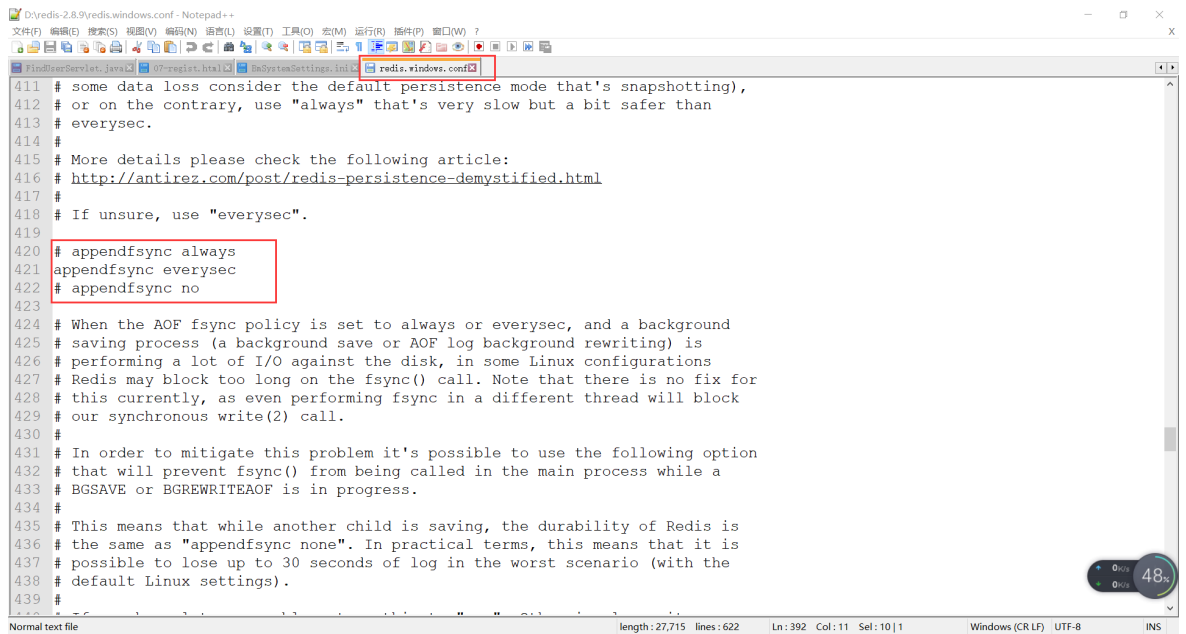
1. 编辑redis.windwos.conf文件

- after 900 sec (15 min) if at least 1 key changed save 900 1
如果900秒（15分钟）之后，只要有一个key被改变之后，那么持久化一次
- after 300 sec (5 min) if at least 10 keys changed save 300 10
如果300秒（5分钟）之后，只要有10个key被改变，那么就持久化一次
- after 60 sec if at least 10000 keys changed save 60 10000
如果60秒之后，只要有10000个key被改变，那么就持久化一次

持久化会保存在xxx.rdb文件中

2. 重新启动redis服务器，并指定配置文件名称

- 用命令行到redis目录下面，指定服务并指定文件名称



Java客户端 Jedis

- Jedis: 一款java操作redis数据库的工具.
- 使用步骤:
 1. 下载jedis的jar包
 2. 使用

```
package cn.jedis.test;

import org.junit.Test;
import redis.clients.jedis.Jedis;

/*
 * jedis的测试类
 * */
public class JedisTest {

    @Test
    public void test() {

        /*获取连接
        * 端口号默认就是6379
        * */
        Jedis jedis = new Jedis("localhost", 6379);

        /*操作*/
        jedis.set("username", "zhangsan");

        /*关闭连接*/
        jedis.close();
    }
}
```

```
}  
}
```

Jedis操作各种redis中的数据结构

字符串类型 string

```
- set  
- get
```

```
//1. 获取连接  
Jedis jedis = new Jedis();//如果使用空参构造, 默认值 "localhost", 6379端口  
//2. 操作  
//存储  
jedis.set("username", "zhangsan");  
//获取  
String username = jedis.get("username");  
System.out.println(username);  
  
//可以使用setex()方法存储可以指定过期时间的 key value  
jedis.setex("activecode", 20, "hehe");//将activecode: hehe键值对存入redis, 并且  
20秒后自动删除该键值对  
  
//3. 关闭连接  
jedis.close();
```

哈希类型 hash: map格式

```
- hset  
- hget  
- hgetAll
```

```
//1. 获取连接  
Jedis jedis = new Jedis();//如果使用空参构造, 默认值 "localhost", 6379端口  
//2. 操作  
// 存储hash  
jedis.hset("user", "name", "lisi");  
jedis.hset("user", "age", "23");  
jedis.hset("user", "gender", "female");  
  
// 获取hash  
String name = jedis.hget("user", "name");  
System.out.println(name);  
// 获取hash的所有map中的数据  
Map<String, String> user = jedis.hgetAll("user");  
  
// keyset  
Set<String> keySet = user.keySet();  
for (String key : keySet) {
```

```

        //获取value
        String value = user.get(key);
        System.out.println(key + ":" + value);
    }

    //3. 关闭连接
    jedis.close();

```

列表类型 list： linkedlist格式。支持重复元素

```

- lpush / rpush
- lpop / rpop
- lrange start end : 范围获取

```

```

//1. 获取连接
Jedis jedis = new Jedis();//如果使用空参构造，默认值 "localhost",6379端口
//2. 操作
// list 存储
jedis.lpush("mylist","a","b","c");//从左边存
jedis.rpush("mylist","a","b","c");//从右边存

// list 范围获取
List<String> mylist = jedis.lrange("mylist", 0, -1);
System.out.println(mylist);

// list 弹出
String element1 = jedis.lpop("mylist");//c
System.out.println(element1);

String element2 = jedis.rpop("mylist");//c
System.out.println(element2);

// list 范围获取
List<String> mylist2 = jedis.lrange("mylist", 0, -1);
System.out.println(mylist2);

//3. 关闭连接
jedis.close();

```

集合类型 set： 不允许重复元素

- sadd
- smembers:获取所有元素

```

//1. 获取连接
Jedis jedis = new Jedis();//如果使用空参构造, 默认值 "localhost", 6379端口
//2. 操作
// set 存储
jedis.sadd("myset", "java", "php", "c++");

// set 获取
Set<String> myset = jedis.smembers("myset");
System.out.println(myset);

//3. 关闭连接
jedis.close();

```

有序集合类型 sortedset: 不允许重复元素, 且元素有顺序

- zadd
- zrange

```

//1. 获取连接
Jedis jedis = new Jedis();//如果使用空参构造, 默认值 "localhost", 6379端口
//2. 操作
// sortedset 存储
jedis.zadd("mysortedset", 3, "亚瑟");
jedis.zadd("mysortedset", 30, "后裔");
jedis.zadd("mysortedset", 55, "孙悟空");

// sortedset 获取
Set<String> mysortedset = jedis.zrange("mysortedset", 0, -1);

System.out.println(mysortedset);
//3. 关闭连接
jedis.close();

```

存储可指定存活日期的数据

- jedis.setex()

```

//1. 获取连接
Jedis jedis = new Jedis();//如果使用空参构造, 默认值 "localhost", 6379端口
//2. 操作
//存储
jedis.set("username", "zhangsan");

//可以使用setex()方法存储可以指定过期时间的 key value
jedis.setex("activecode", 20, "hehe");//将activecode: hehe键值对存入redis, 并且
20秒后自动删除该键值对

//3. 关闭连接
jedis.close();

```

jedis连接池: JedisPool

jedis自带连接池

- 使用:

```
1. 创建JedisPool连接池对象
2. 调用方法 getResource() 方法获取Jedis连接
//0. 创建一个配置对象, 可以传递一些配置参数
JedisPoolConfig config = new JedisPoolConfig();
    /*比如设置一下最大连接数: */
config.setMaxTotal(50);
    /*比如设置一下最大空闲连接: */
config.setMaxIdle(10);

//1. 创建Jedis连接池对象, 传进配置
JedisPool jedisPool = new JedisPool(config, "localhost", 6379);

//2. 获取连接
Jedis jedis = jedisPool.getResource();
//3. 使用
jedis.set("hehe", "heihei");
//4. 关闭 归还到连接池中
jedis.close();
```

jedis详细配置

```
#最大活动对象数
redis.pool.maxTotal=1000
#最大能够保持idle状态的对象数
redis.pool.maxIdle=100
#最小能够保持idle状态的对象数
redis.pool.minIdle=50
#当池内没有返回对象时, 最大等待时间
redis.pool.maxWaitMillis=10000
#当调用borrow Object方法时, 是否进行有效性检查
redis.pool.testOnBorrow=true
#当调用return Object方法时, 是否进行有效性检查
redis.pool.testOnReturn=true
#“空闲链接”检测线程, 检测的周期, 毫秒数。如果为负值, 表示不运行“检测线程”。默认为-1.
redis.pool.timeBetweenEvictionRunsMillis=30000
#向调用者输出“链接”对象时, 是否检测它的空闲超时;
redis.pool.testWhileIdle=true
# 对于“空闲链接”检测线程而言, 每次检测的链接资源的个数。默认为3.
redis.pool.numTestsPerEvictionRun=50
#redis服务器的IP
redis.ip=xxxxxx
#redis服务器的Port
```

```
redis1.port=6379
```

连接池工具类

- jedis.properties

```
# 创建连接池的时候指定的, 并不需要加载
host=127.0.0.1
# 创建连接池的时候指定的, 并不需要加载
port=6379

# 这个需要加载
maxTotal=50
# 这个需要加载
maxIdle=10
```

- java

```
public class JedisPoolUtils {

    private static JedisPool jedisPool;

    static{
        //读取配置文件
        InputStream is =
JedisPoolUtils.class.getClassLoader().getResourceAsStream("jedis.properties");

        //创建Properties对象
        Properties pro = new Properties();

        //关联文件
        try {
            pro.load(is);
        } catch (IOException e) {
            e.printStackTrace();
        }

        //获取数据, 设置到JedisPoolConfig中
        JedisPoolConfig config = new JedisPoolConfig();

        /*注意, pro.getProperty("maxTotal")
        获取到的是一个字符串, 但是要求是一个数字, 所以使用Integer.parseInt()来转换*/
        config.setMaxTotal(Integer.parseInt(pro.getProperty("maxTotal")));
        config.setMaxIdle(Integer.parseInt(pro.getProperty("maxIdle")));

        //初始化JedisPool
        jedisPool = new
JedisPool(config,pro.getProperty("host"),Integer.parseInt(pro.getProperty("port"
)));
    }
}
```

```

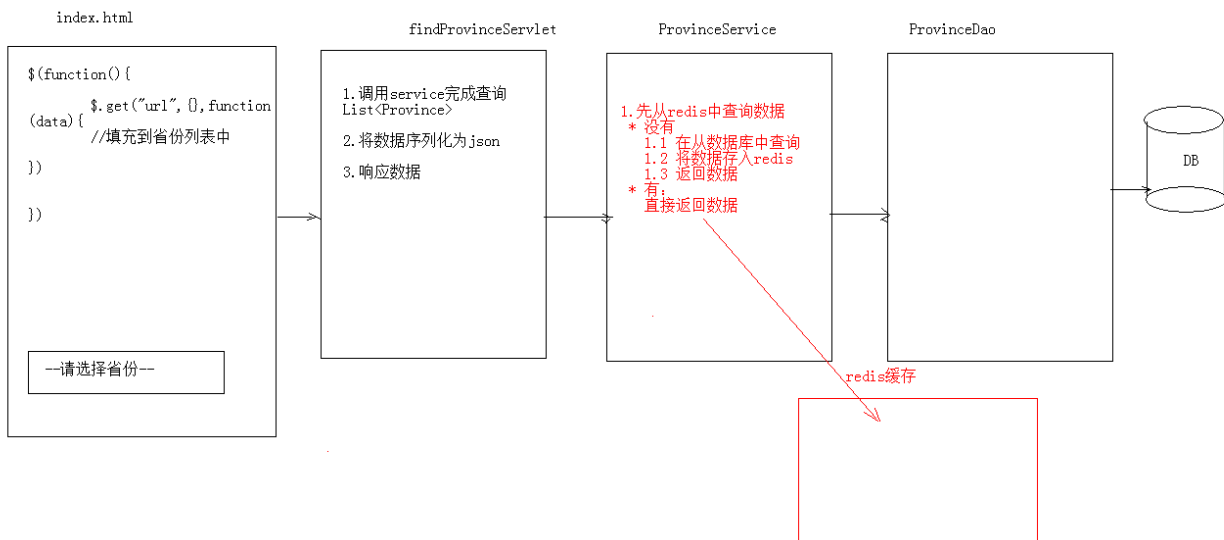
    /**
     * 获取连接方法
     */
    public static Jedis getJedis() {
        return jedisPool.getResource();
    }
}

```

案例：

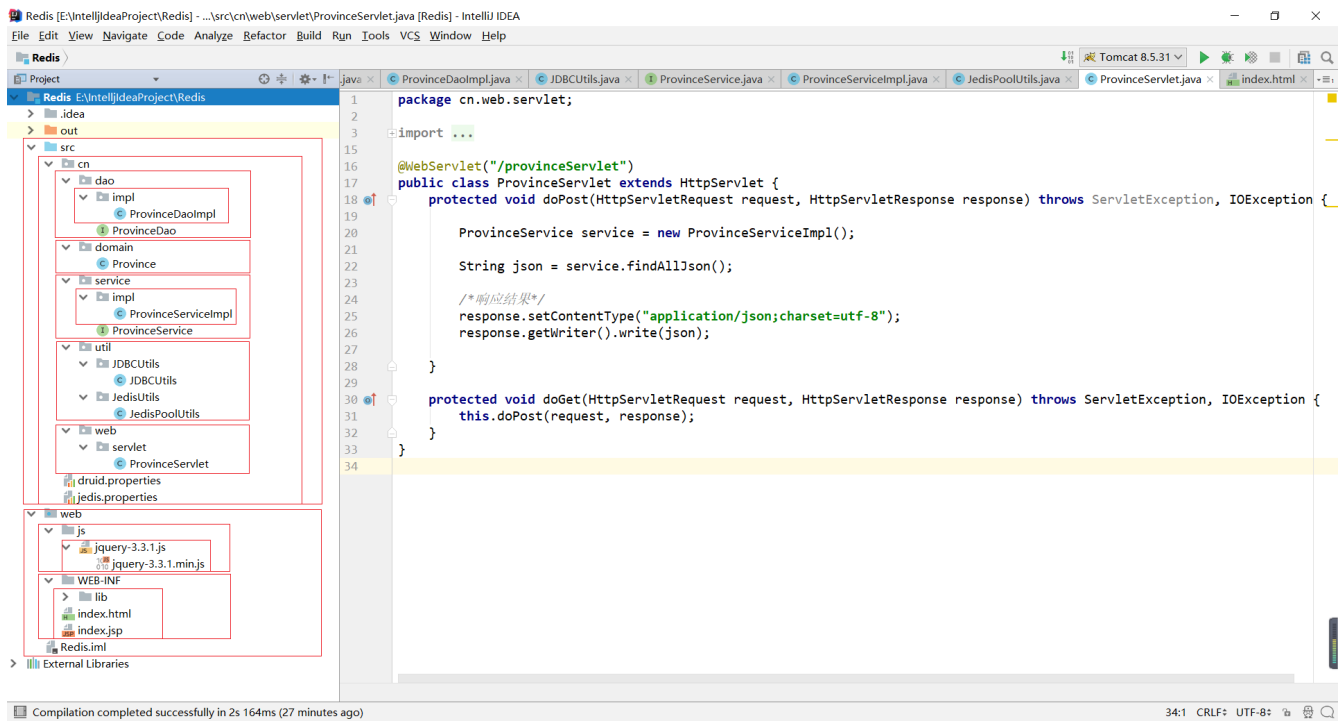
案例需求：

1. 提供index.html页面，页面中有一个省份下拉列表
2. 当页面加载完成后发送ajax请求，加载所有省份



- 注意：使用redis缓存一些不经常发生变化的数据。
 - 数据库的数据一旦发生改变，则需要更新缓存。
 - 数据库的表执行 增删改的相关操作，需要将redis缓存数据情况，再次存入
 - 在service对应的增删改方法中，将redis数据删除。

- 代码结构



- 设计数据库

```
CREATE DATABASE day23; -- 创建数据库
USE day23; -- 使用数据库
CREATE TABLE province( -- 创建表
    id INT PRIMARY KEY AUTO_INCREMENT,
    NAME VARCHAR(20) NOT NULL
);
-- 插入数据
INSERT INTO province VALUES (NULL, '北京');
INSERT INTO province VALUES (NULL, '上海');
INSERT INTO province VALUES (NULL, '广州');
INSERT INTO province VALUES (NULL, '陕西');
```

- Province

```
package cn.domain;

public class Province {

    private int id;
    private String name;

    public Province() {
    }

    public Province(int id, String name) {
        this.id = id;
        this.name = name;
    }
}
```

```

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}

```

- druid.properties

```

driverClassName=com.mysql.jdbc.Driver
url=jdbc:mysql:///day23
username=root
password=root
initialSize=5
maxActive=10
maxWait=3000

```

- JDBCUtils

```

package cn.util.JDBCUtils;

import com.alibaba.druid.pool.DruidDataSourceFactory;

import javax.sql.DataSource;
import java.io.IOException;
import java.io.InputStream;
import java.sql.Connection;
import java.sql.SQLException;
import java.util.Properties;

public class JDBCUtils {

    private static DataSource dataSource;

    static {

        try {

```

```

        Properties properties = new Properties();

        InputStream resourceAsStream =
JDBCUtils.class.getClassLoader().getResourceAsStream("druid.properties");

        properties.load(resourceAsStream);

        dataSource = DruidDataSourceFactory.createDataSource(properties);
    } catch (IOException e) {
        e.printStackTrace();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

public static Connection getConnection() throws SQLException {

    Connection connection = dataSource.getConnection();

    return connection;
}

public static DataSource getDatasouce() {
    return dataSource;
}
}

```

- jedis.properties

```

host=127.0.0.1
port=6379
maxTotal=50
maxIdle=10

```

- JedisUtils

```

package cn.util.JedisUtils;

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

import java.io.IOException;
import java.io.InputStream;
import java.util.Properties;

/**

```

JedisPool工具类

加载配置文件，配置连接池的参数

提供获取连接的方法

```
*/
public class JedisPoolUtils {

    private static JedisPool jedisPool;

    static{

        //读取配置文件
        InputStream is =
JedisPoolUtils.class.getClassLoader().getResourceAsStream("jedis.properties");
        //创建Properties对象
        Properties pro = new Properties();
        //关联文件
        try {
            pro.load(is);
        } catch (IOException e) {
            e.printStackTrace();
        }
        //获取数据，设置到JedisPoolConfig中
        JedisPoolConfig config = new JedisPoolConfig();
        config.setMaxTotal(Integer.parseInt(pro.getProperty("maxTotal")));
        config.setMaxIdle(Integer.parseInt(pro.getProperty("maxIdle")));

        //初始化JedisPool
        jedisPool = new
JedisPool(config,pro.getProperty("host"),Integer.parseInt(pro.getProperty("port"
)));

    }

    /**
     * 获取连接方法
     */
    public static Jedis getJedis(){

        return jedisPool.getResource();
    }
}
```

- ProvinceDao

```

package cn.dao;

import cn.domain.Province;

import java.util.List;

public interface ProvinceDao {

    public List<Province> findAll();

}

```

- ProvinceDaoImpl

```

package cn.dao.impl;

import cn.dao.ProvinceDao;
import cn.domain.Province;
import cn.util.JDBCUtils.JDBCUtils;
import org.springframework.jdbc.core.BeanPropertyRowMapper;
import org.springframework.jdbc.core.JdbcTemplate;

import java.util.List;

public class ProvinceDaoImpl implements ProvinceDao {

    private JdbcTemplate jdbcTemplate = new
JdbcTemplate(JDBCUtils.getDatasouce());

    @Override
    public List<Province> findAll() {

        String sql = "SELECT * FROM province";

        List<Province> list = jdbcTemplate.query(sql, new
BeanPropertyRowMapper<Province>(Province.class));

        return list;

    }

}

```

- ProvinceService

```

package cn.service;

import cn.domain.Province;

import java.util.List;

public interface ProvinceService {

    public List<Province> findAll();

    public String findAllJson();

}

```

- ProvinceServiceImpl

```

package cn.service.impl;

import cn.dao.ProvinceDao;
import cn.dao.impl.ProvinceDaoImpl;
import cn.domain.Province;
import cn.service.ProvinceService;
import cn.util.JedisUtils.JedisPoolUtils;
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import redis.clients.jedis.Jedis;

import java.util.List;

public class ProvinceServiceImpl implements ProvinceService {

    private ProvinceDao dao = new ProvinceDaoImpl();

    @Override
    public List<Province> findAll() {
        return dao.findAll();
    }

    /*使用redis缓存技术*/
    @Override
    public String findAllJson() {
        /*获取redis客户端连接*/
        Jedis jedis = JedisPoolUtils.getJedis();

        String province = jedis.get("province");

        /*判断province是否为空或者是否为null*/

        if (province==null || province.length()==0){
            System.out.println("--redis中无数据，查询数据库中...--");

```

```

        /*查询数据库*/
        List<Province> list = dao.findAll();
        /*将list序列化为json*/
        ObjectMapper mapper = new ObjectMapper();
        try {
            province = mapper.writeValueAsString(list);
        } catch (JsonProcessingException e) {
            e.printStackTrace();
        }

        /*将json数据存入到redis*/
        jedis.set("province", province);
        /*归还jedis*/
        jedis.close();
    } else {
        /*redis中有缓存*/
        System.out.println("查询中...");
    }

    return province;
}
}

```

- ProvinceServlet

```

package cn.web.servlet;

import cn.domain.Province;
import cn.service.ProvinceService;
import cn.service.impl.ProvinceServiceImpl;
import com.fasterxml.jackson.databind.ObjectMapper;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.util.List;

@WebServlet("/provinceServlet")
public class ProvinceServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {

        ProvinceService service = new ProvinceServiceImpl();
    }
}

```

```

        String json = service.findAllJson();

        /*响应结果*/
        response.setContentType("application/json;charset=utf-8");
        response.getWriter().write(json);

    }

    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        this.doPost(request, response);
    }
}

```

- index.html

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>

    <script src="js/jquery-3.3.1.min.js"></script>

    <script>
        $(function () {
            /*发送ajax请求, 加载所有省份数据*/
            $.get("provinceServlet", {}, function (data) {
                // [{"id":1,"name":"北京"}, {"id":2,"name":"上海"},
                {"id":3,"name":"广州"}, {"id":4,"name":"陕西"}]

                /*获取select*/
                var province = $("#province");

                /*遍历json数组*/
                $(data).each(function () {

                    /*创建option*/
                    var option = "<option name='"+this.id+"'>"+this.name+"

</option>>"

                    /*调用select的append追加option*/
                    province.append(option);

                });

            });
        });
    </script>

```



```
});
```

```
</script>
```

```
</head>
```

```
<body>
```

```
<select id="province">
```

```
<option>--请选择省份--</option>
```

```
</select>
```

```
</body>
```

```
</html>
```