# Redis应用初探

# 前言：Redis使用场景

Redis是一个开源的使用ANSI C语言编写、支持网络、可基于内存亦可持久化的日志型、Key-Value数据库，并提供多种语言的API。

我们都知道，在日常的应用中，数据库瓶颈是最容易出现的。数据量太大和频繁的查询，由于磁盘IO性能的局限性，导致项目的性能越来越低。

这时候，基于内存的缓存框架，就能解决我们很多问题。例如Memcache，EHcache，Redis等。将一些频繁使用的数据放入缓存读取，大大降低了数据库的负担。提升了系统的性能。

# 1 Redis介绍

## 1.1 Redis介绍

(1) Redis客户端与Redis服务端关系图：

Redis server

读写

读写

读写

Redis client

Redis client

Redis client

(2) Redis应用时与前后端关系图：

第一种情况是无缓存的时候，后台返回给前台渲染的数据直接从数据库中读取，如果当读取的数据量非常大，或者读取次数非常频繁的时候前台页面渲染速度就会非常慢，页面就会一直处于空白刷新的情况，此时可能就会用到缓存。

第二种情况就是有缓存的时候，如上问题的时候，如果有缓存，后台返回给前台渲染的数据就不会直接从数据库中读取，而是直接从缓存中读取，这个方式效率是非常高的，页面的渲染速度就会非常快，页面就不会一直处于空白刷新的状态。

Redis

2、有缓存

读写

Controller

DB

Client

读写

1、无缓存

## 1.2 Redis常用数据类型

A、String: 常规key-value缓存应用

String是最常用的一种数据类型，普通的key/ value 存储都可以归为此类.

B、Hash: 存储部分变更数据

我们经常将一些结构化的信息打包成hashmap，在客户端序列化后存储为一个字符串的值，比如用户的昵称、年龄、性别、积分等，这时候在需要修改其中某一项时，通常需要将所有值取出反序列化后，修改某一项的值，再序列化存储回去。这样不仅增大了开销，也不适用于一些可能并发操作的场合（比如两个并发的操作都需要修改积分）。Hash结构可以使你像在数据库中Update一个属性一样只修改某一项属性值。

C、List：消息队列系统、取最新N个数据的操作

Lists 就是链表，使用Lists结构，我们可以轻松地实现最新消息排行等功能。Lists的另一个应用就是消息队列，可以利用Lists的PUSH操作，将任务存在Lists中，然后工作线程再用POP操作将任务取出进行执行。Redis还提供了操作Lists中某一段的api，你可以直接查询，删除Lists中某一段的元素。

D、Set：交集，并集，差集; 获取某段时间所有数据去重值

Sets 就是一个无序集合，set是通过hashtable实现的，集合的概念就是一堆不重复值的组合，概念和数学中个的集合基本类似，可以交集，并集，差集等等。利用Redis提供的Sets数据结构，可以存储一些集合性的数据。

E、Sorted set

F、pub/sub

G、Transactions

## 1.3 Redis特点

1、Redis不仅仅支持简单的k/v类型的数据，同时还提供list，set，hash等数据结构的存储。

2、Redis支持数据的备份，即master-slave模式的数据备份。

3、Redis支持数据的持久化，可以将内存中的数据保持在磁盘中，重启的时候可以再次加载进行使用。

4、Redis可以实现主从复制，实现故障恢复。

5、Redis的Sharding技术： 很容易将数据分布到多个Redis实例中

## 1.4 Redis基本配置

1. port 6379

监听端口号，默认为6379，若果设置为0，redis将不在socket上监听任何客户端连接。

1. timeout 0

指定在一个client空闲多少秒之后关闭连接，默认为0，就是不去管它。

(3) loglevel notice

定义日志级别：debug适用于开发或者测试阶段、verbose、notice适用于生产环境、warning仅仅记录一些重要的信息。

(4) logfiel “”

指定日志文件的位置。

(5) redis服务端根据给定的时间间隔和写入次数将数据保存到磁盘

save 900 1：900 秒内如果至少有 1 个 key 的值变化，则保存。

save 300 10：300 秒内如果至少有 10 个 key 的值变化，则保存

save 600 10000：60 秒内如果至少有 10000 个 key 的值变化，则保存

(7) dbfilename dump.rdb

设置redis数据库文件的位置

(8) requirepass 123456

密码设置。此处设置密码后，在连接池配置的时候就要配置对应的密码。

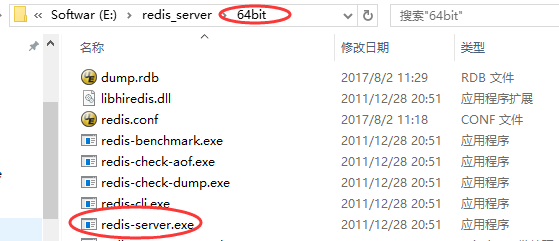
# Redis Java客户端Jedis介绍

Redis：属于独立的运行程序，需要单独安装后，使用Java中的Jedis来操作。因为它是独立，所以如果你写个单元测试程序，放一些数据在redis中，然后又写一个程序去拿数据，那么是可以拿到这个数据的。

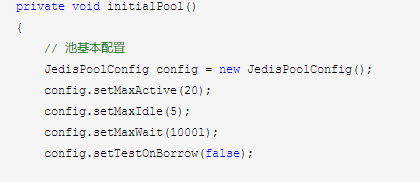
## Jedis使用流程

1. 因为redis属于独立的运行程序，需要单独安装redis-server，在redis.conf文件中可进行配置修改。

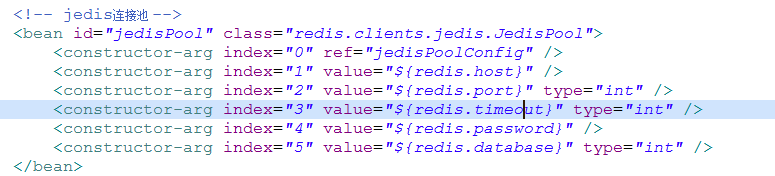
如下，在Windows系统中，双击redis-server.exe即可启动redis server。



(2) Redis的服务端启动后就可以开始java测试了，首先是连接池配置的设置JedisPoolConfig，设置对应的maxIdle、maxTotal、testOnBorrow、maxWait



(3) JedisPoolConfig设置完成后就进行链接池JedisPool连接初始化，初始化对应的host、port、timeout、password、database



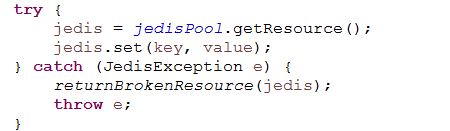
Jedis连接池五种构造方法：

1. public JedisPool(final GenericObjectPoolConfig poolConfig, final String host, int port, int timeout, final String password)；
2. public JedisPool(final GenericObjectPoolConfig poolConfig, final String host, final int port)；
3. public JedisPool(final GenericObjectPoolConfig poolConfig, final String host, final int port, final int timeout)；
4. public JedisPool(final GenericObjectPoolConfig poolConfig, final String host, int port, int timeout, final String password, final int database)；
5. public JedisPool(final GenericObjectPoolConfig poolConfig, final String host, int port, int timeout, final String password, final int database, final String clientName)；

注意：

连接池的配置参数是根据其构造方法里面的参数来动态匹配的，上面的连接池使用的是第二种构造方法，有三个构造参数就会匹配构造参数为三的构造方法。当你Redis服务端设置了密码就用对应的有password构造方法；Redis默认有16个数据库（从0-15），你可以用对应构造方法配置database，当没有配置时，默认是0；你也可以配置连接池超时时间，默认是0。

(4) JedisPool初始化完成后就可以根据JedisPool获取jedis资源对象进行缓存操作了



## 缓存的两种基本操作方式

### 2.2.1采用底层连接池JedisPool进行缓存操作

(1) Pom.xml引入jar包

<dependency>

<groupId>redis.clients</groupId>

<artifactId>jedis</artifactId>

<version>2.5.1</version>

</dependency>

(2) spring加载配置属性文件

<!—jedis连接池配置 -->

<bean id=*"jedisPoolConfig"* class=*"redis.clients.jedis.JedisPoolConfig"*>

<property name=*"maxIdle"* value=*"300"* /> <!-- 最大空闲连接数-->

<property name=*"maxTotal"* value=*"60000"* /> <!—池中最大分配的连接数-->

<!-- 当调用borrow Object方法时，是否进行有效性检查-->

<property name=*"testOnBorrow"* value=*"true"* />

</bean>

<!—jedis的连接池 jedisPool -->

<bean id=*"jedisPool"* class=*"redis.clients.jedis.JedisPool"*>

<constructor-arg index=*"0"* ref=*"jedisPoolConfig"* />

<constructor-arg index=*"1"* value=*"${redis.host}"* />

<constructor-arg index=*"2"* value=*"${redis.port}"* type=*"int"* />

</bean>

(3)用资源对象Jedis进行缓存操作

第一步：从spring容器中获取JedisPool并创建jedis工厂JedisFactory

private static ApplicationContext context = new ClassPathXmlApplicationContext(

"spring-context.xml");

private static JedisPool jedisPool = context.getBean(JedisPool.class);

//获取资源

public static Jedis getResource() throws JedisException {

Jedis jedis = null;

try {

jedis = jedisPool.getResource();

} catch (JedisException e) {

returnBrokenResource(jedis);

throw e;

}

return jedis;

}

第二步：从jedis工厂获取资源jedis后进行缓存操作

以操作string和Object为例：

A、把String放入缓存：result = *jedis*.set(key, value)

从缓存中取出String： value = jedis.get(key)

Redis对Object进行操作时，该java对象首先要实现Serializable接口，其次是jedis操作对象是要把key值和对象序列化为字节数组之后进行缓存，当获取对象缓存的时候在进行反序列化

B、把对象放入缓存：result = *jedis*.set(JedisFactory.*getBytesKey*(key), JedisFactory.*toBytes*(value));

从缓存取出对象：value =

JedisFactory.toObject(jedis.get(JedisFactory.getBytesKey(key)));

其中JedisFactory.*getBytesKey*()、JedisFactory.*toBytes*()、JedisFactory.toObject()均为工程类自定义工具类方法

JedisFactory.*getBytesKey*() ：获取字节数组类型的key值

JedisFactory.*toBytes*() ：把缓存的对象序列化为字节数组

JedisFactory.toObject() ：把缓存中取出字节数组反序列化为对象

### 2.2.2采用template进行缓存操作

注意：template使用起来确实方便 ，不过带来的却是Redis特性的大量丢失 ，尤其是，不支持Sharding机制。

(1) pom.xml引入所需jar包

<dependency>

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

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

<version>1.4.2.RELEASE</version>

</dependency>

<dependency>

<groupId>redis.clients</groupId>

<artifactId>jedis</artifactId>

<version>2.5.1</version>

</dependency>

(2) spring加载属性配置文件

<bean id="poolConfig" class="redis.clients.jedis.JedisPoolConfig">

<property name="maxIdle" value="${redis.maxIdle}" />

<property name="maxTotal" value="${redis.maxActive}" />

<property name="maxWaitMillis" value="${redis.maxWait}" />

<property name="testOnBorrow" value="${redis.testOnBorrow}" />

</bean>

<bean id="jedisConnectionFactory"

class="org.springframework.data.redis.connection.jedis.JedisConnectionFactory"

destroy-method="destroy">

<property name="hostName" value="localhost" />

<property name="port" value="6379" />

<property name="timeout" value="15000" />

<property name="database" value="0" />

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

<property name="usePool" value="true" />

<property name="poolConfig" ref="jedisPoolConfig" />

</bean>

<!-- 开启事务，可以通过transcational注解控制 -->

<bean id="redisTemplate" class="org.springframework.data.redis.core.RedisTemplate">

<property name="connectionFactory" ref="connectionFactory" />

<property name="keySerializer" ref="stringSerializer" />

<property name="enableTransactionSupport" value="true" />

</bean>

1. 配置完成后再程序里面获取缓存操作对象template模板进行缓存操作。

如何应用缓存模板去进行缓存操作在此处没有进行详细介绍，请详见下面的参考资料。

参考以下资料：<http://www.cnblogs.com/luochengqiuse/p/4638988.html>

# 3 业务中的接口封装

(1) 为了实现接口的可扩展行和高复用性，接口封装以接口RedisBaseDao.java作为顶层接口类。示例代码如下：

/\*\*

\* **@Description**: 设置缓存

\*/

**Public** String set(String key, String value, **int** cacheSeconds);

/\*\*

\* **@Description**: 获取缓存

\*/

**public** String get(String key);

...

(2) 顶层接口类封装完成之后就是接口类的实现类RedisBaseDaoImpl.java，该实现类用的是redis顶层的JedisPool连接池来进行缓存操作，用JedisPool操作更加灵活，示例代码如下：

/\*

\* <p> 设置缓存 </p>

\*/

@Override

**public** String set(String key, String value, **int** cacheSeconds) {

Jedis jedis = **null**;

String result = **null**;

**try** {

jedis = *JedisFactory*.*getResource*();

result = jedis.set(key, value);

**if** (cacheSeconds != 0) {

jedis.expire(key, cacheSeconds);

}

*logger*.debug("set {} = {}", key, value);

} **catch** (Exception e) {

*logger*.warn("set {} = {}", key, value, e);

} **finally** {

*JedisFactory*.*returnResource*(jedis);

}

**return** result;

}

/\*

\* <p> 设置缓存 </p>

\*/

@Override

**public** String get(String key) {

String value = **null**;

Jedis jedis = **null**;

**try** {

jedis = *JedisFactory*.*getResource*();

**if** (jedis.exists(key)) {

value = jedis.get(key);

value = StringUtils.*isNotBlank*(value) && !"nil".equalsIgnoreCase(value) ? value

: **null**;

*logger*.debug("get {} = {}", key, value);

}

} **catch** (Exception e) {

*logger*.warn("get {} = {}", key, value, e);

} **finally** {

*JedisFactory*.*returnResource*(jedis);

}

**return** value;

}

...

1. 实现类封装完成之后就是业务接口的封装，业务接口采用泛型类的实现方式，之所以用泛型是应为设置或者获取缓存时，缓存的键值是由程序自动拼接而成的，代码示例如下：

**public** **class** RedisBaseService<T **extends** BaseEntity<T>> {

...

}

存储时的键值有两种实现方式：

1. 传入的key值，即自定义键值

public void addString(String key, String value) {

redisBaseDaoImpl.set(key, value, 0);

}

(2) Key=泛型类名称\_存储类型名称\_id，调用上面方法

public void addString(T t, String value) {

String keyPreix = t.getClass().getName() + String\_KEY\_SPLIT;

String id = t.getId();

String key = keyPreix + id;

this.addString(key, value);

}