第九章 企业项目开发--分布式缓存Redis(1)

注意：本章代码将会建立在上一章的代码基础上，上一章链接《第八章 企业项目开发--分布式缓存memcached》

1、为什么用Redis

1.1、为什么用分布式缓存（或者说本地缓存存在的问题）？

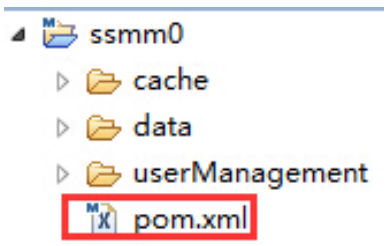
- 见《第八章 企业项目开发--分布式缓存memcached》

1.2、有了memcached，为什么还要用redis？

- 见《第一章 常用的缓存技术》

2、代码实现

2.1、ssmm0



pom.xml

只在dev环境下添加了以下代码：

```
                <!--
                    redis:多台服务器支架用什么符号隔开无所谓，只要在程序中用相应的符号去分隔就好。
                    这里只配置了一个redis.servers,如果系统特别大的时候，可以为每一种业务或某几种业务配置一个
redis.xxx.servers
                -->
                <redis.servers><![CDATA[127.0.0.1:6379]]></redis.servers>
                <!--
                    下边各个参数的含义在RedisFactory.java中有介绍，
                    当我们三种环境(dev/rc/prod)下的一些参数都相同时，可以将这些参数直接设置到cache_conf.properties文
件中去
                -->
                <redis.timeout>2000</redis.timeout><!-- 操作超时时间：2s,单位：ms -->
                <redis.conf.lifo>true</redis.conf.lifo>
                <redis.conf.maxTotal>64</redis.conf.maxTotal>
                <redis.conf.blockWhenExhausted>true</redis.conf.blockWhenExhausted>
                <redis.conf.maxWaitMillis>-1</redis.conf.maxWaitMillis>

                <redis.conf.testOnBorrow>false</redis.conf.testOnBorrow>
                <redis.conf.testOnReturn>false</redis.conf.testOnReturn>

                <!-- 空闲连接相关 -->
                <redis.conf.maxIdle>8</redis.conf.maxIdle>
                <redis.conf.minIdle>0</redis.conf.minIdle>
                <redis.conf.testWhileIdle>true</redis.conf.testWhileIdle>

<redis.conf.timeBetweenEvictionRunsMillis>30000</redis.conf.timeBetweenEvictionRunsMillis><!-- 30s -->
                <redis.conf.numTestsPerEvictionRun>8</redis.conf.numTestsPerEvictionRun>
                <redis.conf.minEvictableIdleTimeMillis>60000</redis.conf.minEvictableIdleTimeMillis><!-- 60s
-->
```

注意：看注释。

完整版的根pom.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <groupId>com.xxx</groupId>
    <artifactId>ssmm0</artifactId>
    <version>1.0-SNAPSHOT</version>

    <name>ssmm0</name>
    <packaging>pom</packaging><!-- 父模块 -->

    <!-- 管理子模块 -->
    <modules>
        <module>userManagement</module><!-- 具体业务1-人员管理系统 -->
        <module>data</module><!-- 封装数据操作 -->
        <module>cache</module><!-- 缓存模块 -->
    </modules>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
    </properties>

    <!-- dependencyManagement不会引入实际的依赖，只是作为一个依赖池，供其和其子类使用 -->
    <dependencyManagement>
        <dependencies>
            <!-- json -->
            <dependency>
                <groupId>com.alibaba</groupId>
                <artifactId>fastjson</artifactId>
                <version>1.1.39</version>
            </dependency>
            <!-- servlet -->
            <dependency>
                <groupId>javax.servlet</groupId>
                <artifactId>javax.servlet-api</artifactId>
                <version>3.0.1</version>
                <scope>provided</scope>
            </dependency>
            <!-- spring -->
            <dependency>
                <groupId>org.springframework</groupId>
                <artifactId>spring-core</artifactId>
                <version>3.2.6.RELEASE</version>
            </dependency>
            <dependency>
                <groupId>org.springframework</groupId>
                <artifactId>spring-beans</artifactId>
                <version>3.2.6.RELEASE</version>
            </dependency>
            <dependency>
                <groupId>org.springframework</groupId>
                <artifactId>spring-context</artifactId>
                <version>3.2.6.RELEASE</version>
            </dependency>
            <dependency>
                <groupId>org.springframework</groupId>
                <artifactId>spring-web</artifactId>
                <version>3.2.6.RELEASE</version>
            </dependency>
            <dependency>
                <groupId>org.springframework</groupId>
                <artifactId>spring-webmvc</artifactId>
```

```
            <version>3.2.6.RELEASE</version>
        </dependency>
        <!-- 这个是使用velocity的必备包 -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context-support</artifactId>
            <version>3.2.6.RELEASE</version>
        </dependency>
        <!-- mysql -->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.27</version>
            <scope>runtime</scope>
        </dependency>
        <!-- 数据源 -->
        <dependency>
            <groupId>org.apache.tomcat</groupId>
            <artifactId>tomcat-jdbc</artifactId>
            <version>7.0.47</version>
        </dependency>
        <!-- mybatis -->
        <dependency>
            <groupId>org.mybatis</groupId>
            <artifactId>mybatis</artifactId>
            <version>3.1.1</version>
        </dependency>
        <dependency>
            <groupId>org.mybatis</groupId>
            <artifactId>mybatis-spring</artifactId>
            <version>1.1.1</version>
        </dependency>
        <!-- velocity -->
        <dependency>
            <groupId>org.apache.velocity</groupId>
            <artifactId>velocity</artifactId>
            <version>1.5</version>
        </dependency>
        <dependency>
            <groupId>velocity-tools</groupId>
            <artifactId>velocity-tools-generic</artifactId>
            <version>1.2</version>
        </dependency>
        <!-- 用于加解密 -->
        <dependency>
            <groupId>commons-codec</groupId>
            <artifactId>commons-codec</artifactId>
            <version>1.7</version>
        </dependency>
        <dependency>
            <groupId>org.bouncycastle</groupId>
            <artifactId>bcprov-jdk15on</artifactId>
            <version>1.47</version>
        </dependency>
        <!-- 集合工具类 -->
        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-collections4</artifactId>
            <version>4.0</version>
        </dependency>
        <!-- 字符串处理类 -->
        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-lang3</artifactId>
            <version>3.4</version>
```

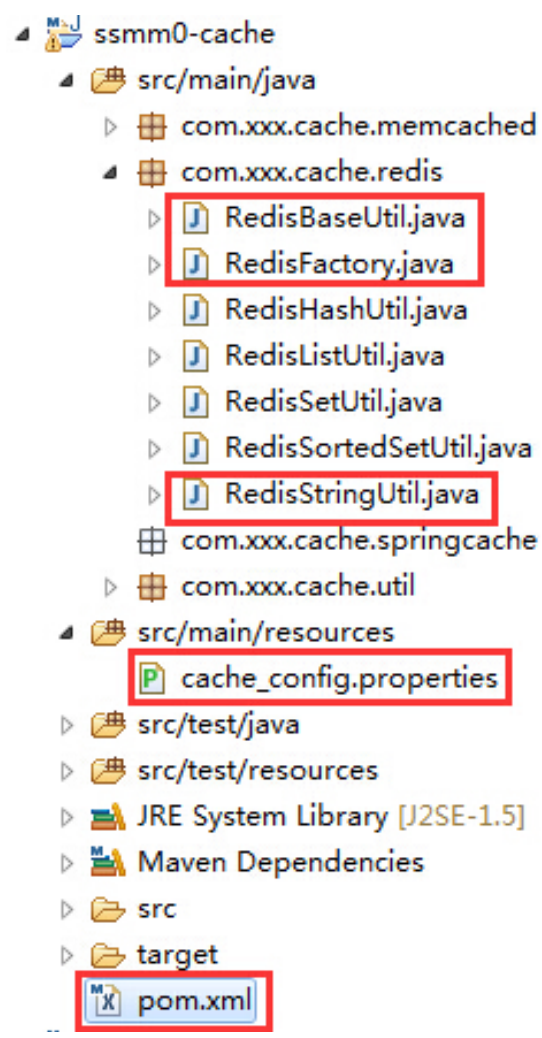
```
            </dependency>
            <!-- http -->
            <dependency>
                <groupId>org.apache.httpcomponents</groupId>
                <artifactId>httpclient</artifactId>
                <version>4.2.6</version>
            </dependency>
        </dependencies>
    </dependencyManagement>

    <!-- 引入实际依赖 -->
    <dependencies>
        <!-- json -->
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>fastjson</artifactId>
        </dependency>
        <!-- spring -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-beans</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
        </dependency>
        <!-- 集合工具类 -->
        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-collections4</artifactId>
        </dependency>
        <!-- 字符串处理类 -->
        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-lang3</artifactId>
        </dependency>
    </dependencies>

    <build>
        <resources>
            <!-- 这里配置了这一块儿true，才可以让指定文件（这里是src/main/resources/spring-data.xml）读到pom.xml中的
配置信息
                ， 值得注意的是，如果src/main/resources下还有其他文件，而你不想让其读pom.xml， 你还必须得把src/main
/resources下的其余文件再配置一遍，配置为false（不可读pom.xml），
                    如下边的注释那样，否则，会报这些文件找不到的错误
            -->
            <resource>
                <directory>src/main/resources</directory>
                <filtering>true</filtering>
                <includes>
                    <include>*.xml</include>
                    <include>*.properties</include>
                </includes>
            </resource>
            <!--
            <resource>
                <directory>src/main/resources</directory>
                <filtering>false</filtering>
                <includes>
                    <include>*.properties</include>
                </includes>
```

```
            </resource>
            -->
            <resource>
                <directory>src/main/resources</directory>
                <filtering>false</filtering>
                <includes>
                <!-- 这里如果不加这一条，那么在spring-data.xml中配置的xml将找不到classpath:mapper/admin
/AdminMapper.xml -->
                    <include>mapper/**/*.xml</include>
                </includes>
            </resource>
        </resources>
    </build>

    <!--
        profiles可以定义多个profile，然后每个profile对应不同的激活条件和配置信息，从而达到不同环境使用不同配置信息的效
果
        注意两点：
        1）<activeByDefault>true</activeByDefault>这种情况表示服务器启动的时候就采用这一套env（在这里，就是prod）
        2）当我们启动服务器后，想采用开发模式，需切换maven的env为dev，如果env的配置本身就是dev，需要将env换成rc或prod，
点击apply，然后再将env切换成dev，点击apply才行
    -->
    <profiles>
        <!-- 开发env -->
        <profile>
            <id>dev</id>
            <activation>
                <!-- 这里为了测试方便，改为了true，在上线的时候一定要改成false，否则线上使用的就是这一套dev的环境了
-->
                <activeByDefault>true</activeByDefault>
                <property>
                    <name>env</name>
                    <value>dev</value>
                </property>
            </activation>
            <properties>
                <env>dev</env>

                <jdbc.driverClassName>com.mysql.jdbc.Driver</jdbc.driverClassName>
                <!--
                     对于jdbc.url中内容的配置，如果需要配置 &amp;时，有两种方法：
                    1）如下边这样，使用<![CDATA[XXX]]>包起来
                    2）使用jdbc.properties文件来读取此pom.xml，然后spring.xml再读取jdbc.properties文件 显然，前者更
方便，而且还省了一个jdbc.properties的文件，但是，有的时候，还是会用后者的；
                    在使用后者的时候，注意三点：
                    1）需要修改上边的build中的内容
                    2）需要在spring.xml中配置<context:property-placeholder
location="classpath:jdbc.properties"/>
                    3）将jdbc.properties放在ssmm0-data项目中，之后需要将ssmm0-data项目的env配置为dev
                -->
                <jdbc.url><![CDATA[jdbc:mysql://127.0.0.1:3306/blog?zeroDateTimeBehavior=convertToNull&
amp;useUnicode=true&amp;characterEncoding=utf-8]]></jdbc.url>
                <jdbc.username>root</jdbc.username>
                <jdbc.password>123456</jdbc.password>

                <!-- memcache，多台服务器之间需要使用空格隔开，而不要使用英文逗号隔开，因为Xmemcached的AddrUtil源码是
根据空格隔开的 -->
                <memcached.servers><![CDATA[127.0.0.1:11211]]></memcached.servers>
                <memcached.max.client>10</memcached.max.client><!-- 最多的客户端数 -->
                <memcached.expiretime>900</memcached.expiretime><!-- 过期时间900s -->
                <memcached.hash.consistent>true</memcached.hash.consistent><!-- 是否使用一致性hash算法 -->
                <memcached.connection.poolsize>1</memcached.connection.poolsize><!-- 每个客户端池子的连接数 -->
                <memcached.op.timeout>2000</memcached.op.timeout><!-- 操作超时时间 -->

                <!--
```

```
                    redis:多台服务器支架用什么符号隔开无所谓，只要在程序中用相应的符号去分隔就好。
                    这里只配置了一个redis.servers,如果系统特别大的时候，可以为每一种业务或某几种业务配置一个
redis.xxx.servers
                -->
                <redis.servers><![CDATA[127.0.0.1:6379]]></redis.servers>
                <!--
                    下边各个参数的含义在RedisFactory.java中有介绍，
                    当我们三种环境(dev/rc/prod)下的一些参数都相同时，可以将这些参数直接设置到cache_conf.properties文
件中去
                -->
                <redis.timeout>2000</redis.timeout><!-- 操作超时时间：2s,单位：ms -->
                <redis.conf.lifo>true</redis.conf.lifo>
                <redis.conf.maxTotal>64</redis.conf.maxTotal>
                <redis.conf.blockWhenExhausted>true</redis.conf.blockWhenExhausted>
                <redis.conf.maxWaitMillis>-1</redis.conf.maxWaitMillis>

                <redis.conf.testOnBorrow>false</redis.conf.testOnBorrow>
                <redis.conf.testOnReturn>false</redis.conf.testOnReturn>

                <!-- 空闲连接相关 -->
                <redis.conf.maxIdle>8</redis.conf.maxIdle>
                <redis.conf.minIdle>0</redis.conf.minIdle>
                <redis.conf.testWhileIdle>true</redis.conf.testWhileIdle>

<redis.conf.timeBetweenEvictionRunsMillis>30000</redis.conf.timeBetweenEvictionRunsMillis><!-- 30s -->
                <redis.conf.numTestsPerEvictionRun>8</redis.conf.numTestsPerEvictionRun>
                <redis.conf.minEvictableIdleTimeMillis>60000</redis.conf.minEvictableIdleTimeMillis><!-- 60s
-->
            </properties>
        </profile>
        <!-- 预上线env -->
        <profile>
            <id>rc</id>
            <activation>
                <activeByDefault>false</activeByDefault>
                <property>
                    <name>env</name>
                    <value>rc</value>
                </property>
            </activation>
            <properties>
                <env>rc</env>

                <jdbc.driverClassName>com.mysql.jdbc.Driver</jdbc.driverClassName>
                <!-- 假设的一个地址 -->
                <jdbc.url><![CDATA[jdbc:mysql://10.10.10.100:3306/blog?zeroDateTimeBehavior=convertToNull&
amp;useUnicode=true&amp;characterEncoding=utf-8]]></jdbc.url>
                <jdbc.username>root2</jdbc.username>
                <jdbc.password>1234562</jdbc.password>
            </properties>
        </profile>
        <!-- 线上env -->
        <profile>
            <id>prod</id>
            <activation>
                <!-- 这里为了测试方便，改为了false，在上线的时候一定要改成true，否则线上使用的就不是这一套环境了 -->
                <activeByDefault>false</activeByDefault>
                <property>
                    <name>env</name>
                    <value>prod</value>
                </property>
            </activation>
            <properties>
                <env>prod</env>
```

```
                <jdbc.driverClassName>com.mysql.jdbc.Driver</jdbc.driverClassName>
                <!-- 假设的一个地址 -->
                <jdbc.url><![CDATA[jdbc:mysql://99.99.99.999:3307/blog?zeroDateTimeBehavior=convertToNull&
amp;useUnicode=true&amp;characterEncoding=utf-8]]></jdbc.url>
                <jdbc.username>sadhijhqwui</jdbc.username>
                <jdbc.password>zxczkchwihcznk=</jdbc.password>
            </properties>
        </profile>
    </profiles>
</project>
```

2.2、ssmm0-cache



pom.xml完整版（只是添加了jedis的依赖包）

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <!-- 指定父模块 -->
    <parent>
        <groupId>com.xxx</groupId>
        <artifactId>ssmm0</artifactId>
        <version>1.0-SNAPSHOT</version>
    </parent>

    <groupId>com.xxx.ssmm0</groupId>
    <artifactId>ssmm0-cache</artifactId>

    <name>ssmm0-cache</name>
    <packaging>jar</packaging>

    <!-- 引入实际依赖 -->
    <dependencies>
```

```
        <!-- memcached -->
        <dependency>
            <groupId>com.googlecode.xmemcached</groupId>
            <artifactId>xmemcached</artifactId>
            <version>1.4.3</version>
        </dependency>
        <!-- redis -->
        <dependency>
            <groupId>redis.clients</groupId>
            <artifactId>jedis</artifactId>
            <version>2.6.1</version>
        </dependency>
    </dependencies>
</project>
```

cache_config.properties（添加了redis相关配置）

```
#memcached配置#
#memcached服务器集群
memcached.servers = ${memcached.servers}
#缓存过期时间
memcached.expiretime = ${memcached.expiretime}
#是否使用一致性hash算法
memcached.hash.consistent = ${memcached.hash.consistent}
#memcached的最大客户端数量
memcached.max.client = ${memcached.max.client}
#每个客户端池子的连接数
memcached.connection.poolsize = ${memcached.connection.poolsize}
#操作超时时间
memcached.op.timeout = ${memcached.op.timeout}

#redis配置#
#redis集群
redis.servers = ${redis.servers}
#超时时间
redis.timeout = ${redis.timeout}
#是否启用后进先出
redis.conf.lifo = ${redis.conf.lifo}
#最多创建几个ShardJedis，即连接
redis.conf.maxTotal = ${redis.conf.maxTotal}
#连接耗尽是否阻塞等待
redis.conf.blockWhenExhausted = ${redis.conf.blockWhenExhausted}
#等待获取连接的最长时间
redis.conf.maxWaitMillis = ${redis.conf.maxWaitMillis}
#获取连接前，是否对连接进行测试
redis.conf.testOnBorrow = ${redis.conf.testOnBorrow}
#归还连接前，是否对连接进行测试
redis.conf.testOnReturn = ${redis.conf.testOnReturn}
#最大空闲连接数
redis.conf.maxIdle = ${redis.conf.maxIdle}
#最小空闲连接数
redis.conf.minIdle = ${redis.conf.minIdle}
#对空闲连接进行扫描，检查连接有效性
redis.conf.testWhileIdle = ${redis.conf.testWhileIdle}
#两次扫描空闲连接的时间间隔
redis.conf.timeBetweenEvictionRunsMillis = ${redis.conf.timeBetweenEvictionRunsMillis}
#每次空闲扫描时扫描的控线连接的个数
redis.conf.numTestsPerEvictionRun = ${redis.conf.numTestsPerEvictionRun}
#一个空闲连接至少连续保持多长时间空闲才会被空闲扫描
redis.conf.minEvictableIdleTimeMillis = ${redis.conf.minEvictableIdleTimeMillis}
```

7个Java类：

- RedisFactory：构建ShardJedisPool。
  - 一个ShardJedisPool中配置了多个JedisShardInfo
  - 每一个JedisShardInfo都是一个server
  - 一个ShardJedisPool中可以获取多个ShardJedis连接实例，具体数目由maxTotal属性而定
- RedisBaseUtil：获取获取ShardJedis连接与归还ShardJedis连接（这是其他所有缓存操作都需要的方法）
- RedisStringUtil：redis的第一种数据结构--字符串，操作类
- RedisListUtil：redis的第二种数据结构--list，操作类
- RedisSetUtil：redis的第三种数据结构--set，操作类
- RedisSortedSetUtil：redis的第四种数据结构--sorted set，操作类
- RedisHashUtil：redis的第五种数据结构--hash，操作类

RedisFactory：

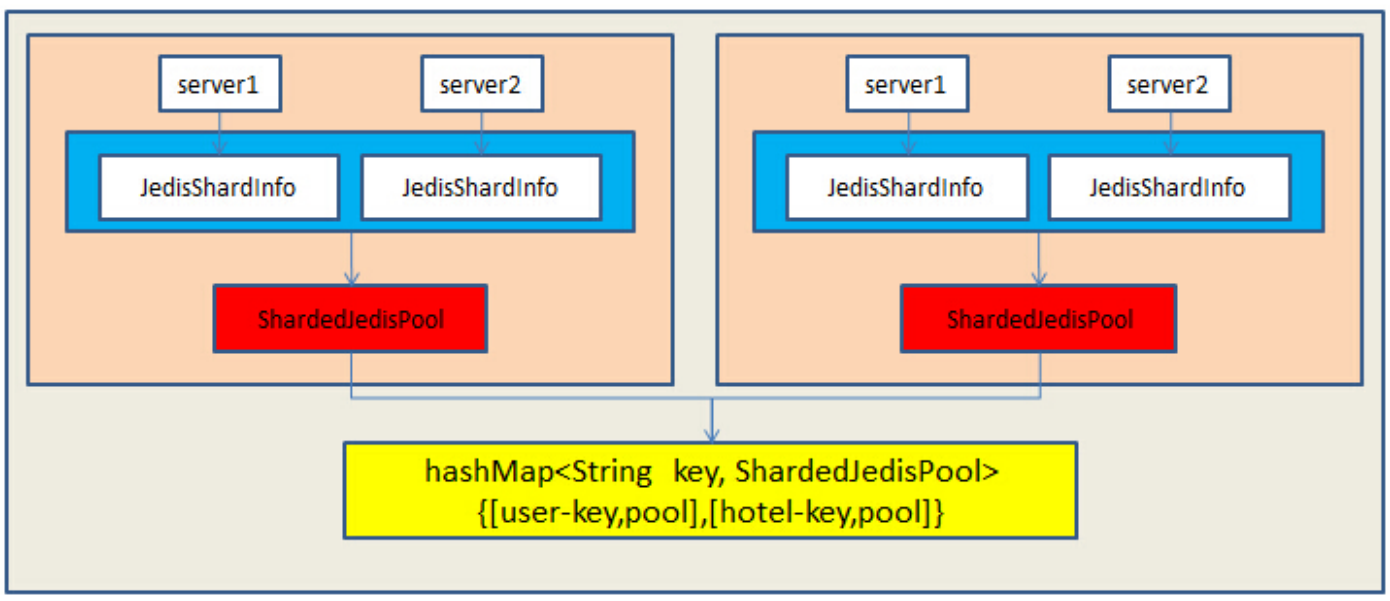
```
 1 package com.xxx.cache.redis;
 2
 3 import java.util.ArrayList;
 4 import java.util.List;
 5 import java.util.Properties;
 6
 7 import org.apache.commons.lang3.math.NumberUtils;
 8
 9 import com.xxx.cache.util.FileUtil;
10
11 import redis.clients.jedis.JedisPoolConfig;
12 import redis.clients.jedis.JedisShardInfo;
13 import redis.clients.jedis.ShardedJedisPool;
14
15 public class RedisFactory {
16     private static ShardedJedisPool jedisPool = null;
17     /**
18      * 构建ShardJedisPool
19      * 一个ShardJedisPool中配置了多个JedisShardInfo
20      * 每一个JedisShardInfo都是一个server
21      * 一个ShardJedisPool中可以获取多个ShardJedis连接实例，具体数目由maxTotal属性而定
22      * 注意：
23      * 1、这里只有一个ShardJedisPool，如果你有很多业务，而且不想这些业务都共用几台redis服务器的话，
24      *         你可以创建多个ShardJedisPool,每个pool中放置不同的服务器即可
25      * 2、这时候多个ShardJedisPool可以放置在一个hashmap中，key由自己指定（写在一个Enum类中去），key的名称一般与业
务挂钩就好
26      */
27     static{
28         Properties props = FileUtil.loadProps("cache_config.properties");//加载属性文件
29         /*
30          * 从属性文件读取参数
31          */
32         String servers = props.getProperty("redis.servers", "127.0.0.1:6379");
33         String[] serverArray = servers.split(" ");//获取服务器数组
34
35         int timeout = FileUtil.getInt(props, "redis.timeout", 5000);//默认：2000ms(超时时间：单位ms)
36         boolean lifo = FileUtil.getBoolean(props, "redis.conf.lifo", true);//默认：true
37
38         int maxTotal = FileUtil.getInt(props, "redis.conf.maxTotal", 64);//默认：8个(最多创建几个
ShardJedis，即连接)
39         boolean blockWhenExhausted = FileUtil.getBoolean(props, "redis.conf.blockWhenExhausted", true);//
默认：true(连接耗尽是否阻塞等待)
40         long maxWaitMillis = FileUtil.getLong(props, "redis.conf.maxWaitMillis", -1);//默认：-1，即无限等待
(等待获取连接的最长时间)
41
42         boolean testOnBorrow = FileUtil.getBoolean(props, "redis.conf.testOnBorrow", false);//默
```

```
认：false(获取连接前，是否对连接进行测试)
 43         boolean testOnReturn = FileUtil.getBoolean(props, "redis.conf.testOnReturn", false);//默
认：false(归还连接前，是否对连接进行测试)
 44
 45         int maxIdle = FileUtil.getInt(props, "redis.conf.maxIdle", 8);//默认：8(最大空闲连接数)
 46         int minIdle = FileUtil.getInt(props, "redis.conf.minIdle", 0);//默认：0(最小空闲连接数)
 47         boolean testWhileIdle = FileUtil.getBoolean(props, "redis.conf.testWhileIdle", true);//默
认：false(对空闲连接进行扫描，检查连接有效性)
 48         long timeBetweenEvictionRunsMillis = FileUtil.getLong(props,
"redis.conf.timeBetweenEvictionRunsMillis", 30000);//默认：-1，(两次扫描空闲连接的时间间隔)
 49         int numTestsPerEvictionRun = FileUtil.getInt(props, "redis.conf.numTestsPerEvictionRun", 3);//默
认：3(每次空闲扫描时扫描的控线连接的个数)
 50         long minEvictableIdleTimeMillis = FileUtil.getLong(props,
"redis.conf.minEvictableIdleTimeMillis", 60000);//默认：30min(一个空闲连接至少连续保持30min中空闲才会被空闲扫描)
 51         /*
 52          * 配置redis参数
 53          */
 54         JedisPoolConfig config = new JedisPoolConfig();
 55         config.setLifo(lifo);//(last in, first out)是否启用后进先出，默认true
 56         /*
 57          * 即原来的maxActive,能够同时建立的最大连接个数（就是最多分配多少个ShardJedis实例），
 58          * 默认8个，若设置为-1，表示为不限制，
 59          * 如果pool中已经分配了maxActive个jedis实例，则此时pool的状态就成exhausted了
 60          *
 61          * 这里最多可以生产64个shardJedis实例
 62          */
 63         config.setMaxTotal(maxTotal);
 64         config.setBlockWhenExhausted(blockWhenExhausted);//连接耗尽时是否阻塞， false报异常,true阻塞直到超时,
默认true，达到maxWait时抛出JedisConnectionException
 65         config.setMaxWaitMillis(maxWaitMillis);//获取连接时的最大等待毫秒数(如果设置为阻塞时
BlockWhenExhausted),如果超时就抛异常，小于零:阻塞不确定的时间， 默认-1
 66
 67         config.setTestOnBorrow(testOnBorrow);//使用连接时，先检测连接是否成功,若为true，则获取到的shardJedis连
接都是可用的，默认false
 68         config.setTestOnReturn(testOnReturn);//归还连接时，检测连接是否成功
 69
 70         /*
 71          * 空闲状态
 72          */
 73         config.setMaxIdle(maxIdle);//空闲连接数（即状态为idle的ShardJedis实例）大于maxIdle时，将进行回收,默认8
个
 74         config.setMinIdle(minIdle);//空闲连接数小于minIdle时，创建新的连接,默认0
 75         /*
 76          * 在空闲时检查有效性，默认false,如果为true，表示有一个idle object evitor线程对idle object进行扫描，
 77          * 如果validate失败，此object会被从pool中drop掉；这一项只有在timeBetweenEvictionRunsMillis大于0时才有意
义
 78          */
 79         config.setTestWhileIdle(testWhileIdle);
 80         config.setTimeBetweenEvictionRunsMillis(timeBetweenEvictionRunsMillis);//表示idle object evitor两次
扫描之间要sleep的毫秒数；
 81         config.setNumTestsPerEvictionRun(numTestsPerEvictionRun);//表示idle object evitor每次扫描的最多的对象
数；
 82         //表示一个对象至少停留在idle状态的最短时间，然后才能被idle object evitor扫描并驱逐；这一项只有在
timeBetweenEvictionRunsMillis大于0时才有意义；
 83         config.setMinEvictableIdleTimeMillis(minEvictableIdleTimeMillis);
 84
 85         /*
 86          * 构建JedisShardInfo集合
 87          */
 88         List<JedisShardInfo> jedisList = new ArrayList<JedisShardInfo>(1);//我这里只有一台机器，所以传入参数
1，否则默认为10，浪费空间
 89         for(String server : serverArray){
 90             String[] hostAndPort = server.split(":");
 91             /*
 92              * 这句代码中我没有判断hostAndPort是不是长度为2，而且端口如果没有指定或指定错误的话，就直接转到6379
```

```
93              * 实际中，我们在配置服务器的时候就一定要注意配置格式正确：host:port
94              */
95             JedisShardInfo shardInfo = new JedisShardInfo(hostAndPort[0],
96                                                           NumberUtils.toInt(hostAndPort[1], 6379),
97                                                           timeout);
98             jedisList.add(shardInfo);
99         }
100        /*
101         * 创建ShardJedisPool
102         */
103        jedisPool = new ShardedJedisPool(config, jedisList);//构建jedis池
104    }
105
106    /**
107     * 如果有多个ShardJedisPool，则需要写一个hash算法从hashmap中选一个pool返回
108     */
109    public static ShardedJedisPool getJedisPool() {
110        return jedisPool;
111    }
112 }
```

注意：

- 这里只有一个ShardJedisPool，如果你有很多业务，而且不想这些业务都共用几台redis服务器的话，你可以创建多个ShardJedisPool,每个pool中放置不同的服务器即可；这时候多个ShardJedisPool可以放置在一个hashmap中，key由自己指定（写在一个Enum类中去），key的名称一般与业务挂钩就好。如果没说清，看下图：（在我当前的程序中由于只有一个业务简单，值设置了一个ShardJedisPool，所以没有下图的hashmap）



- 配置参数较多（这里列出几乎所有的配置参数）
  - testOnBorrow：使用连接时，先检测连接是否成功,若为true，则获取到的shardJedis连接都是可用的，默认false；在实际使用中，直接使用默认值，因为虽然该参数配置为true可以保证获取到的连接一定可用，但是由于每次获取连接都要进行测试，所以效率会变低；考虑到获取到的连接不可用的概率很低，综合考虑下，将该值设为false还是比较合适的。
  - testOnReturn：同上，在我们下面的程序中可以看到每个缓存操作方法的流程都是"获取连接-->进行缓存操作-->归还连接"
  - 注意这里jedis的版本是2.6.1，一些配置属性可能在其他版本看不到（eg.maxTotal），而其他版本的一些属性可能在该版本中没有（eg.maxActive）。
- jedis参数介绍参考：http://blog.csdn.net/huahuagongzi99999/article/details/13631579，但是由于版本不同，请注意上边这一条所说的。

RedisBaseUtil：

```
1  package com.xxx.cache.redis;
2
3  import redis.clients.jedis.ShardedJedis;
4  import redis.clients.jedis.ShardedJedisPool;
5
6  /**
7   * 获取ShardJedis与归还实例
8   */
9  public class RedisBaseUtil {
10     /**
```

```
     * 从ShardJedisPool中获取ShardJedis
     */
    public static ShardedJedis getJedis(){
        ShardedJedisPool jedisPool = RedisFactory.getJedisPool();//获取连接池
        if(jedisPool == null){
            return null;
        }
        return jedisPool.getResource();
    }

    /**
     * 归还jedis实例到连接池中
     */
    public static void returnJedis(ShardedJedis jedis, boolean broken){
        if(jedis==null){//如果传入的jedis是null的话，不需要归还
            return;
        }
        ShardedJedisPool jedisPool = RedisFactory.getJedisPool();//获取连接池
        if(jedisPool == null){//如果连接池为null的话，不需要归还
            return;
        }
        if(broken){//如果为true的话，表示是因为发生了异常才归还
            jedisPool.returnBrokenResource(jedis);
            return;
        }
        jedisPool.returnResource(jedis);//缓存正常操作结束之后，归还jedis
    }
}
```

注意：

- returnBrokenResource：操作被打断，即没有正常结束缓存操作，连接归还
- returnResource：缓存正常操作结束后，连接归还

RedisStringUtil：

```
package com.xxx.cache.redis;

import com.xxx.cache.util.CachePrefix;

import redis.clients.jedis.ShardedJedis;

/**
 * 字符串缓存操作类或者JavaBean缓存操作类
 * key String, value String-->看下边的注意点2
 * key byte[], value byte[]-->key.getBytes[], value 序列化为byte[]，通常需要自己写一个序列化工具
 * 注意：这一点与memcached不一样，memcached可以key String, value Object
 * 1、memcached直接加序列化器就可以，或者在业务层中将Object-->String
 * 2、redis执行此接口，一般只会采用后者Object-->String
 */
public class RedisStringUtil extends RedisBaseUtil{
    private static final String KEY_SPLIT = "-";//用于隔开缓存前缀与缓存键值
    /**
     * 设置缓存
     * 类似于memcached的set，不管是否已经有相同的key，都成功
     * 实际上只是set(String, String)
     */
    public static void set(CachePrefix keyPrefix, String key, String value){
        boolean broken = false;//标记：该操作是否被异常打断而没有正常结束
        ShardedJedis jedis = null;
        try {
            jedis = getJedis();//获取jedis实例
```

```
            if(jedis==null){
                broken = true;
                return;
            }
            jedis.set(keyPrefix+KEY_SPLIT+key, value);//set(String,String),value除了string以外，还可以是byte[]
        } catch (Exception e) {
            broken = true;
        }finally{
            returnJedis(jedis, broken);
        }
    }

    /**
     * 设置缓存，并指定缓存过期时间，单位是秒
     */
    public static void setex(CachePrefix keyPrefix, String key, String value, int expire){
        boolean broken = false;//该操作是否被异常打断而没有正常结束
        ShardedJedis jedis = null;
        try {
            jedis = getJedis();//获取jedis实例
            if(jedis==null){
                broken = true;
                return;
            }
            jedis.setex(keyPrefix+KEY_SPLIT+key, expire, value);
        } catch (Exception e) {
            broken = true;
        }finally{
            returnJedis(jedis, broken);
        }
    }

    /**
     * 设置缓存，如果设置的key不存在，直接设置，如果key已经存在了，则什么操作都不做，直接返回
     * 类似于memcached的add
     */
    public static boolean setnx(CachePrefix keyPrefix, String key, String value){
        boolean broken = false;//该操作是否被异常打断而没有正常结束
        ShardedJedis jedis = null;
        try {
            jedis = getJedis();//获取jedis实例
            if(jedis==null){
                broken = true;
                return false;
            }
            long setCount = jedis.setnx(keyPrefix+KEY_SPLIT+key, value);
            if(setCount == 1){
                return true;
            }
            return false;
        } catch (Exception e) {
            broken = true;
        }finally{
            returnJedis(jedis, broken);
        }
        return false;
    }

    /**
     * 根据key获取缓存
     * @param key
     * @return String
     */
    public static String get(CachePrefix keyPrefix, String key){
        boolean broken = false;//该操作是否被异常打断而没有正常结束
```

```
        ShardedJedis jedis = null;
        try {
            jedis = getJedis();//获取jedis实例
            if(jedis==null){
                broken = true;
                return null;
            }
            return jedis.get(keyPrefix+KEY_SPLIT+key);
        } catch (Exception e) {
            broken = true;
        }finally{
            returnJedis(jedis, broken);
        }
        return null;
    }

    /**
     * 删除缓存
     */
    public static void delete(CachePrefix keyPrefix, String key){
        boolean broken = false;//该操作是否被异常打断而没有正常结束
        ShardedJedis jedis = null;
        try {
            jedis = getJedis();//获取jedis实例
            if(jedis==null){
                broken = true;
                return;
            }
            jedis.del(keyPrefix+KEY_SPLIT+key);
        } catch (Exception e) {
            broken = true;
        }finally{
            returnJedis(jedis, broken);
        }
    }

    /**
     * 更新缓存过期时间，单位：秒
     * 从运行该方法开始，为相应的key-value设置缓存过期时间expire
     * 类似于memcached中的touch命令
     */
    public static void setExpire(CachePrefix keyPrefix, String key, int expire){
        boolean broken = false;
        ShardedJedis jedis = null;
        try {
            jedis = getJedis();
            if(jedis==null){
                broken = true;
                return;
            }
            jedis.expire(keyPrefix+KEY_SPLIT+key, expire);
        } catch (Exception e) {
            broken = true;
        }finally{
            returnJedis(jedis, broken);
        }
    }

    /**
     * 测试
     */
    public static void main(String[] args) {
        //System.out.println(RedisStringUtil.get("hello"));
        //RedisStringUtil.delete("hello");
        //RedisStringUtil.setex("hello1", "word1", 1);
```

```
        //RedisStringUtil.setExpire("hello1", 20);
        //System.out.println(RedisStringUtil.get("hello1"));
    }

}
```

注意：

- 只有set(string,string)和set(byte[],byte[])，前者类似于Xmemcached的文本协议，后者类似于Xmemcached的二进制协议
- set-->Xmemcached的set，redis上是否已经有与将要存放的key相同的key，都会操作成功
- setnx-->Xmemcached的add，redis上有与将要存放的key相同的key，操作失败
- expire-->Xmemcached的touch，该方法会在方法执行的时候，为还存在的key-value重新指定缓存过期时间

附：

这里需要安装一个redis服务器，redis在实际使用中是安装在Linux上的，我们为了方便，使用windows版本的（我这里使用了redis2.6-win32），如果是64bit的，可以使用redis2.8。

redis2.6(32bit)的文件下载链接：http://pan.baidu.com/s/1hri1erq

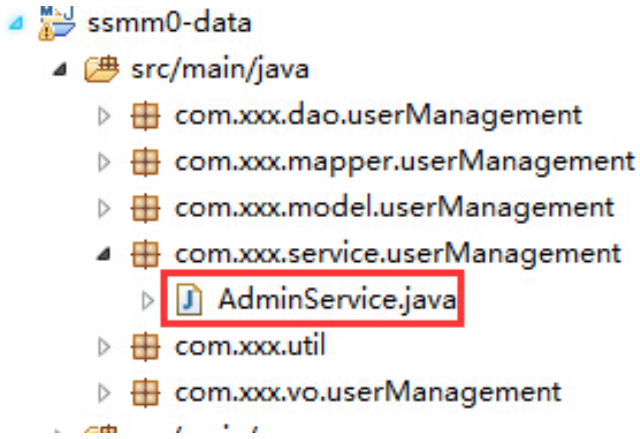
安装方式如下：

下载后解压，此时如果直接双击"redis-server.exe"，可能会报内存警告，所以先修改redis.conf文件，添加如下配置，如果你下载的上边的链接，可能已经配置了。

```
315  # maxmemory <bytes>
316  #maxmemory 31457280
317  maxmemory 30M
```

之后，以管理员身份运行cmd.exe，并在命令窗口中进入redis-server.exe所在目录下，执行"redis-server.exe redis.conf"即可。

2.3、ssmm0-data



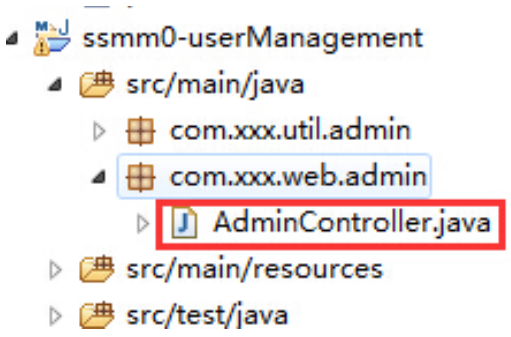
AdminService：

```
    /**********************redis*******************/
    public Admin findAdminByIdFromRedis(int id) {
        //从缓存中获取数据
        String adminStr = RedisStringUtil.get(CachePrefix.USER_MANAGEMENT, String.valueOf(id));
        //若缓存中有，直接返回
        if(StringUtils.isNoneBlank(adminStr)){
            return Admin.parseJsonToAdmin(adminStr);
        }
        //若缓存中没有，从数据库查询
        Admin admin = adminDao.getUserById(id);
        //若查询出的数据不为null
        if(admin!=null){
            //将数据存入缓存
            RedisStringUtil.set(CachePrefix.USER_MANAGEMENT, String.valueOf(id), admin.toJson());
        }
        //返回从数据库查询的admin（当然也可能数据库中也没有，就是null）
        return admin;
```

```
    }
```

说明：只添加了如上方法，相当于将上一节的memcached缓存换成了redis

2.4、ssmm0-userManagement



AdminController：

```
    /************************redis*****************************/
    /**
     * 根据id查找Admin
     */
    @ResponseBody
    @RequestMapping("/findAdminByIdFromRedis")
    public Admin findAdminByIdFromRedis(@RequestParam(value="id") int id){

        return adminService.findAdminByIdFromRedis(id);
    }
```

说明：只添加了如上方法。

3、测试

首先对ssmm0整个项目"clean compile"，然后通过浏览器访问，进行整体测试，测试方法与上一章《第八章 企业项目开发--分布式缓存memcached》完全相同

在以上的代码中，我只写了redis的String类型数据结构的缓存操作，其余的四种下一篇再说。