第八章 企业项目开发--分布式缓存memcached

注意：本节代码基于《第七章 企业项目开发--本地缓存guava cache》

1、本地缓存的问题

- 本地缓存速度一开始高于分布式缓存，但是随着其缓存数量的增加，所占内存越来越大，系统运行内存越来越小，最后系统会被拖慢（这一点与第二点联系起来）
- 本地缓存存于本机，其缓存数量与大小受本机内存大小限制
- 本地缓存存于本机，其他机器的访问不到这样的缓存

解决方案：分布式缓存

- Jboss cache：缓存还存于本机，但是会同步更新到其他机器（解决了第三个问题，解决不了第一和第二个问题），如果缓存机器数量很多，同步更新很耗时
- memcached：缓存存于其他机器，理论上缓存数量与大小无限（因为集群可伸缩），且不需要同步，所以即使缓存机器数量很多，也无所谓，但是这样就会造成单点故障问题，最简单易行的解决方案是缓存备份，即缓存至少存两份。

2、memcached Java客户端的选用

当下常用的三种memcached Java客户端：

- Memcached Client for Java：memcached官方提供，基于Java BIO实现
- SpyMemcached：基于Java NIO
- XMemcached：基于Java NIO，并发性能优于XMemcached，实际上SpyMemcached性能也很高

三者的实验比较结果：

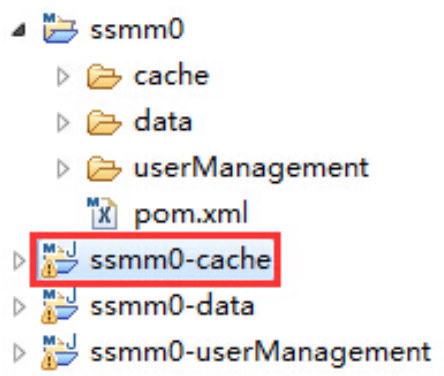
http://xmemcached.googlecode.com/svn/trunk/benchmark/benchmark.html

所以，我们选用XMemcached来实现客户端的编写。

3、代码

在原来的代码结构上，我增加了一个ssmm0-cache模块，专门用于放置分布式缓存相关（memcached、redis、spring cache）的代码。

项目整体结构：



说明：怎样新建maven项目，并加入原来项目，最后引入eclipse，见第一章《第一章 企业项目开发--maven+springmvc+spring+mybatis+velocity整合》

3.1、ssmm0

pom.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
4
5     <modelVersion>4.0.0</modelVersion>
6
7     <groupId>com.xxx</groupId>
8     <artifactId>ssmm0</artifactId>
9     <version>1.0-SNAPSHOT</version>
10
11    <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 -->
            </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>
```

说明：这里给出了完整版，实际上只做了以下5点改动：

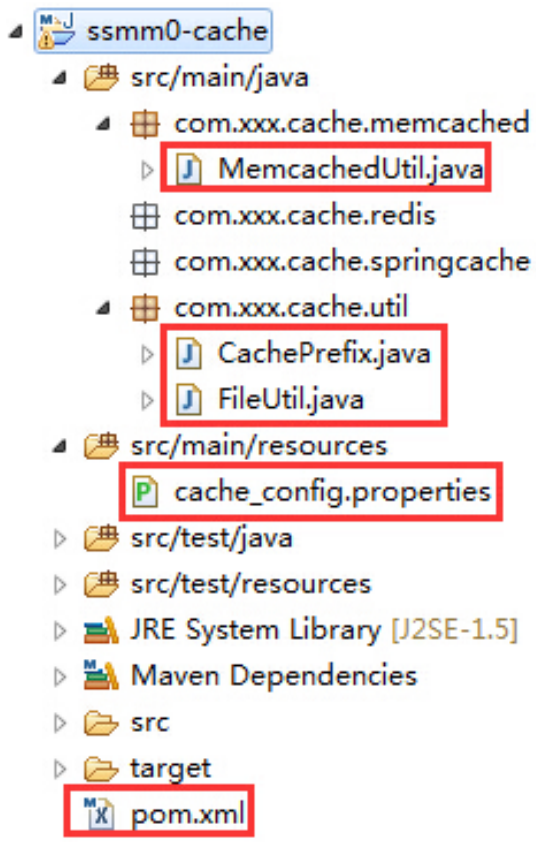
- 新增cache子module
- 引入了commons-lang3的jar包，该jar封装了一些对于字符串的处理方法，eg.isBlank()
- <resources>部分因为要将该pom.xml中的配置读取到ssmm0-cache模块的properties文件中去，所以添加了过滤目录
- 在dev环境下配置了与memcached相关的服务器列表及参数
- 最后一点，只是为了测试方便，将dev设为默认选用的环境，而prod不是，在实际上线之前，一定要改回来

注意：

- 在pom.xml中配置memcached服务器列表时，要以"ip1:port1 ip2:port2..."这样的形式，即每台服务器之间用空格隔开，这与Xmemcached的AddrUtil读取服务器列表的方式有关。

3.2、ssmm0-cache

模块结构：



3.2.1、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>

    <!-- 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>
    </dependencies>
</project>
```

说明：在该pom.xml中引入了xmemcached的jar包，注意该jar依赖于slf4j.jar，如果不是用maven的话，需要手动导入slf4j.jar，但是用maven的话，maven自己会导入相关的依赖包。

3.2.2、cache_config.properties

```
#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}
```

说明：这里需要从根pom.xml（即ssmm0的pom.xml）中读取信息，所以在根pom.xml的资源过滤部分有相关的修改

3.2.3、FileUtil.java

```
package com.xxx.cache.util;

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

import org.apache.commons.lang3.math.NumberUtils;

/**
 * 文件操作工具类
 */
public class FileUtil {

    /**
     * 加载属性文件*.properties
     * @param fileName 不是属性全路径名称，而是相对于类路径的名称
     */
    public static Properties loadProps(String fileName){
        Properties props = null;
        InputStream is = null;

        try {
            is = Thread.currentThread().getContextClassLoader().getResourceAsStream(fileName);//获取类路径下的fileName文件，并且转化为输入流
            if(is != null){
                props = new Properties();
                props.load(is);    //加载属性文件
            }
        } catch (Exception e) {
            e.printStackTrace();
        }finally{
            if(is!=null){
                try {
                    is.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }

        return props;
    }

    /*
     * 从属性文件中获取int型数据
```

```
45      */
46     public static int getInt(Properties props, String key, int defaultValue){
47         int value = defaultValue;
48         if(props.containsKey(key)){                                    //属性文件中是否包含给定键值
49             value = NumberUtils.toInt(props.getProperty(key), defaultValue);//从属性文件中取出给定键值的value,并且转换为int型
50         }
51         return value;
52     }
53 
54     /*
55      * 从属性文件中获取long型数据
56      */
57     public static long getLong(Properties props, String key, long defaultValue){
58         long value = defaultValue;
59         if(props.containsKey(key)){                                    //属性文件中是否包含给定键值
60             value = NumberUtils.toLong(props.getProperty(key), defaultValue);//从属性文件中取出给定键值的value,并且转换为int型
61         }
62         return value;
63     }
64 
65     /*
66      * 从属性文件中获取boolean型数据
67      */
68     public static boolean getBoolean(Properties props, String key, boolean defaultValue){
69         boolean value = defaultValue;
70         if(props.containsKey(key)){                                    //属性文件中是否包含给定键值
71             value = toBoolean(props.getProperty(key), defaultValue);
72         }
73         return value;
74     }
75 
76 
77     public static boolean toBoolean(String str, boolean defaultValue) {
78         if(str == null) {
79             return defaultValue;
80         }
81         return Boolean.parseBoolean(str);
82     }
83 
84     /**
85      * 测试
86      */
87     public static void main(String[] args) {
88         Properties props = FileUtil.loadProps("cache_config.properties");
89         //System.out.println(props);
90         System.out.println(props.getProperty("memcached.servers", "123"));//从属性文件中读取string
91         System.out.println(FileUtil.getInt(props, "httpclient.max.conn.per.route2", 10));//属性文件中没有这个key
92     }
93 }
```

说明：对于该类的介绍与注意点，参看"Java文件相关"系列的《第一章 属性文件操作工具类》

在该类中，添加了一些方法

- 从属性文件中将String转化为long型数据，与String转int一样，使用NumberUtil即可
- 从属性文件中将String转化为Boolean型数据，直接参考NumberUtil的相关源代码进行编写即可
- 从属性文件中读取String，props.getProperties(String key, String defaultValue)即可

注意：

- 如果直接运行该类中的main方法来读取properties文件中由根pom.xml传来的参数，可能读取不到；可以通过后边的MemcachedUtil中的main方法来获取，或者直接通过最后的浏览器访问来获取就好

3.2.4、CachePrefix.java

```
package com.xxx.cache.util;

/**
 * 在该类中定义一些缓存的前缀
 * 方式：
 * 1、定义一个类，在其中添加静态常量
 * 2、使用枚举类（这是最好的方式）
 * 作用：
 * 1、防止缓存键值重复（通常情况下，每一种业务对应一种前缀）
 * 2、可以起到根据前缀分类的作用
 * 后者主要是在memcached中实现类似于redis的实现。
 */
public enum CachePrefix {
    USER_MANAGEMENT,    //人员管理业务类缓存前缀
    HOTEL_MANAGEMENT;    //酒店管理业务类缓存前缀
}
```

说明：在该类中定义一些缓存的前缀

作用：

- 防止缓存键值重复（通常情况下，**每一种业务对应一种前缀**）
- 以起到根据前缀分类的作用（后者主要是在memcached中实现类似于redis的实现），但是其实memcached可以通过使用命名空间来实现分类，具体的参看我的"Java缓存相关"系列中后续的文章

注意：定义常量类有两种方式

- 普通类，里边使用static final常量
- 枚举类（推荐），这里就是使用了枚举类

关于枚举类与普通类做常量类的优缺点对比，查看《effective Java：第二版》第30条，或者参考我"Java高效使用"系列中后续的文章

3.2.5、MemcachedUtil

```
package com.xxx.cache.memcached;

import java.io.IOException;
import java.util.HashMap;
import java.util.Map;
import java.util.Properties;
import java.util.concurrent.TimeoutException;

import org.apache.commons.lang3.StringUtils;

import com.xxx.cache.util.CachePrefix;
import com.xxx.cache.util.FileUtil;


import net.rubyeye.xmemcached.MemcachedClient;
import net.rubyeye.xmemcached.MemcachedClientBuilder;
import net.rubyeye.xmemcached.XMemcachedClientBuilder;
import net.rubyeye.xmemcached.command.BinaryCommandFactory;
import net.rubyeye.xmemcached.exception.MemcachedException;
import net.rubyeye.xmemcached.impl.KetamaMemcachedSessionLocator;
import net.rubyeye.xmemcached.transcoders.CompressionMode;
import net.rubyeye.xmemcached.transcoders.SerializingTranscoder;
import net.rubyeye.xmemcached.utils.AddrUtil;

/**
 * memcached工具类（基于Xmemcached实现）
```

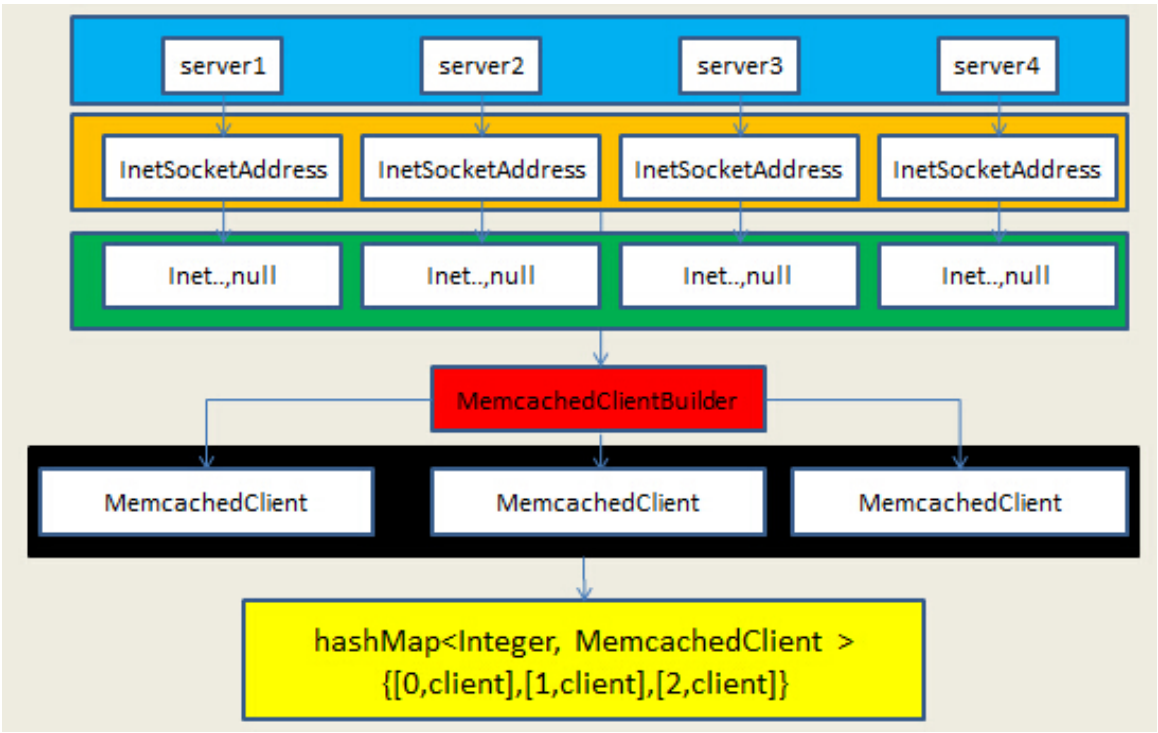
```
 */
public class MemcachedUtil {
    private static Map<Integer, MemcachedClient> clientMap
                        = new HashMap<Integer, MemcachedClient>();//client的集合
    private static int maxClient = 3;
    private static int expireTime = 900;//900s（默认的缓存过期时间）
    private static int maxConnectionPoolSize = 1;//每个客户端池子的连接数
    private static long op_time = 2000L;//操作超时时间

    private static final String KEY_SPLIT = "-";//用于隔开缓存前缀与缓存键值

    /**
     * 构建MemcachedClient的map集合
     */
    static{
        //读取配置文件
        Properties props = FileUtil.loadProps("cache_config.properties");
        String servers = props.getProperty("memcached.servers", "127.0.0.1:11211");//获取memcached servers集合
        maxClient = FileUtil.getInt(props, "", maxClient);
        expireTime = FileUtil.getInt(props, "memcached.expiretime", expireTime);
        maxConnectionPoolSize = FileUtil.getInt(props, "memcached.connection.poolsize", maxConnectionPoolSize);
        op_time = FileUtil.getLong(props, "memcached.op.timeout", op_time);

        if(StringUtils.isNotBlank(servers)){
            MemcachedClientBuilder builder = new XMemcachedClientBuilder(AddrUtil.getAddresses(servers));
            builder.setConnectionPoolSize(1);//这个默认也是1
            builder.setSessionLocator(new KetamaMemcachedSessionLocator(true));//使用一致性hash算法

            SerializingTranscoder transcoder = new SerializingTranscoder(1024*1024);//序列化转换器，指定最大的数据大小1M
            transcoder.setCharset("UTF-8");//默认为UTF-8，这里可去掉
            transcoder.setCompressionThreshold(1024*1024);//单位：字节，压缩边界值，任何一个大于该边界值（这里是：1M）的数据都要进行压缩
            transcoder.setCompressionMode(CompressionMode.GZIP);//压缩算法

            builder.setTranscoder(transcoder);
            builder.setCommandFactory(new BinaryCommandFactory());//命令工厂

            //构建10个MemcachedClient,并放入clientMap
            for(int i=0;i<maxClient;i++){
                try {
                    MemcachedClient client = builder.build();
                    client.setOpTimeout(op_time);//设置操作超时时间，默认为1s
                    clientMap.put(i, client);
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    }

    /**
     * 从MemcachedClient中取出一个MemcachedClient
     */
    public static MemcachedClient getMemcachedClient(){
        /*
         * Math.random()：产生[0,1)之间的小数
         * Math.random()*maxClient：[0~maxClient)之间的小数
         * (int)(Math.random()*maxClient)：[0~maxClient)之间的整数
         */
        return clientMap.get((int)(Math.random()*maxClient));
    }

```

```
88      /**
89       * 设置缓存
90       * @param keyPrefix     缓存的键的前缀
91       * @param key                   缓存的键
92       * @param value                 缓存的值
93       * @param exp                     缓存过期时间
94       */
95      public static void setCacheWithNoReply(CachePrefix keyPrefix,
96                                             String key,
97                                             Object value,
98                                             int exp){
99          try {
100             MemcachedClient client = getMemcachedClient();
101             client.addWithNoReply(keyPrefix+KEY_SPLIT+key, exp, value);
102         } catch (InterruptedException e) {
103             e.printStackTrace();
104         } catch (MemcachedException e) {
105             e.printStackTrace();
106         }
107     }
108
109     /**
110      * 设置缓存
111      * @param exp       缓存过期时间（默认时间）
112      */
113     public static void setCacheWithNoReply(CachePrefix keyPrefix,
114                                            String key,
115                                            Object value){
116         setCacheWithNoReply(keyPrefix, key, value, expireTime);
117     }
118
119     /**
120      * 设置缓存，并返回缓存成功与否
121      * 注意：
122      * 1、设置已经设置过的key-value，将会返回false
123      */
124     public static boolean setCache(CachePrefix keyPrefix,
125                                    String key,
126                                    Object value,
127                                    int exp){
128         boolean setCacheSuccess = false;
129         try {
130             MemcachedClient client = getMemcachedClient();
131             setCacheSuccess = client.add(keyPrefix+KEY_SPLIT+key, exp, value);
132         } catch (TimeoutException e) {
133             e.printStackTrace();
134         } catch (InterruptedException e) {
135             e.printStackTrace();
136         } catch (MemcachedException e) {
137             e.printStackTrace();
138         }
139         return setCacheSuccess;
140     }
141
142     /**
143      * 设置缓存，并返回缓存成功与否（缓存超时时间采用默认）
144      * @param key
145      * @param value
146      */
147     public static boolean setCache(CachePrefix keyPrefix,
148                                    String key,
149                                    Object value){
150         return setCache(keyPrefix, key, value, expireTime);
151     }
152
```

```
153     /**
154      * 获取缓存
155      */
156     public static Object getCache(CachePrefix keyPrefix, String key){
157         Object value = null;
158         try {
159             MemcachedClient client = getMemcachedClient();
160             value = client.get(keyPrefix+KEY_SPLIT+key);
161         } catch (TimeoutException e) {
162             e.printStackTrace();
163         } catch (InterruptedException e) {
164             e.printStackTrace();
165         } catch (MemcachedException e) {
166             e.printStackTrace();
167         }
168         return value;
169     }
170 
171     /**
172      * 删除缓存
173      */
174     public static void removeCacheWithNoReply(CachePrefix keyPrefix, String key){
175         try {
176             MemcachedClient client = getMemcachedClient();
177             client.deleteWithNoReply(keyPrefix+KEY_SPLIT+key);
178         } catch (InterruptedException e) {
179             e.printStackTrace();
180         } catch (MemcachedException e) {
181             e.printStackTrace();
182         }
183     }
184 
185     /**
186      * 删除缓存,并返回删除成功与否
187      */
188     public static boolean removeCache(CachePrefix keyPrefix, String key){
189         boolean removeCacheSuccess = false;
190         try {
191             MemcachedClient client = getMemcachedClient();
192             removeCacheSuccess = client.delete(keyPrefix+KEY_SPLIT+key);
193         } catch (TimeoutException e) {
194             e.printStackTrace();
195         } catch (InterruptedException e) {
196             e.printStackTrace();
197         } catch (MemcachedException e) {
198             e.printStackTrace();
199         }
200         return removeCacheSuccess;
201     }
202 
203     /**
204      * 测试
205      * @param args
206      */
207     public static void main(String[] args) {
208         /*for(int i=0;i<100;i++){
209             System.out.println(Math.random());
210         }*/
211         System.out.println(MemcachedUtil.setCache(CachePrefix.USER_MANAGEMENT,"hello4", "world"));
212         System.out.println(MemcachedUtil.getCache(CachePrefix.USER_MANAGEMENT,"hello4"));
213         /*System.out.println(MemcachedUtil.getCache("hello2"));
214         System.out.println(MemcachedUtil.getCache("hello2"));
215         System.out.println(MemcachedUtil.getCache("hello2"));
216         System.out.println(MemcachedUtil.getCache("hello2"));
217         System.out.println(MemcachedUtil.getCache("hello2"));*/
```

```
218     }
219 }
```

首先给出该类的一个图（不知道该怎么称呼这个图）



说明：上述的图可以根据源代码的追踪画出来；该类是基于XMemcached实现了的一个工具类，主要包含以下6个部分

- 属性默认值的指定
- static静态块中读取属性文件，并与前边的默认值一起来指定最终参数值
- 根据服务器列表构建MemcachedClientBuilder，配置builder的相关属性，包括序列化转化器、二进制协议工厂等
- 通过上述的MemcachedClientBuilder构建指定个数（这里是3个）的MemcachedClient客户端，存于clientMap中
- 提供从clientMap获取MemcachedClient的方法（这里是随机获取）
- 提供缓存的基本操作，增删查操作

注意：

- 我们提供了三个MemcachedClient，那是不是说明同时只能处理三个并发请求呢？不是，Xmemcached基于Java NIO，每一个MemcachedClient都会启动一个reactor线程和一些工作线程，基于IO多路复用技术，一个reactor线程理论上可以接受极高的并发量，甚至可以说成，该reactor线程可以接过所有到达memcached的请求，然后通过事件机制（类似于观察者模式）将这些请求派发给工作线程，进行相应的操作。关于Java NIO、reactor线程模型、事件机制等可以参看《netty权威指南（第2版）》。
- 正如上边所说，每启动一个MemcachedClient，就必须启动一个reactor线程和一些工作线程，这其实是一个昂贵的操作，所以构建多个客户端是比较昂贵的，基于此，XMemcached提供了池化操作，即一个配置参数（setConnectionPoolSize），但是在实际使用中发现，当配置该参数>1的情况下会发生线程死锁现象，所以还是采用多客户端的方式吧！在我们的实际使用中，10个客户端接收百万级请求绝对是轻轻松松的！
- 这里需要注意的是，服务器列表的配置必须用空格隔开，原理查看AddrUtil的源代码
- 关于序列化与反序列化发生的时机，请参看《http://blog.csdn.net/tang9140/article/details/43445511》或者我的"Java缓存相关"的后续文章
- Xmemcached提供了很多的缓存操作API，这些API我会在"Java缓存相关"的后续文章介绍，这里想说，如果你需要提供很多的API方法，那么推荐将上述"说明"中的前5部分写在一个类中（MemcachedFactory），将第6部分（缓存相关操作）写在一个类中（MemcachedUtil），这样会很清晰
- 其他属性的配置，我也会在"Java缓存相关"的后续文章介绍

在这里，需要装一个memcached服务器了。

我们就简要的装一个windows版本的，我发了一个再云盘上，链接：http://pan.baidu.com/s/1dDMlov3，下载后，解压，

两种使用方法：

A、双击解压后的"x86"(32位)或"x64"(64位)文件夹中的memcached.exe，跳出窗口即可。

B、在C:\Windows\System32\cmd.exe右击"以管理员身份运行"-->在命令窗口进入E:\memcached\x86目录中-->"memcached.exe -d install"-->之后去"本地服务"看看是不是已经有memcached server的服务了，如果已经有了，说明安装成功-->之后启动服务，两种方式：

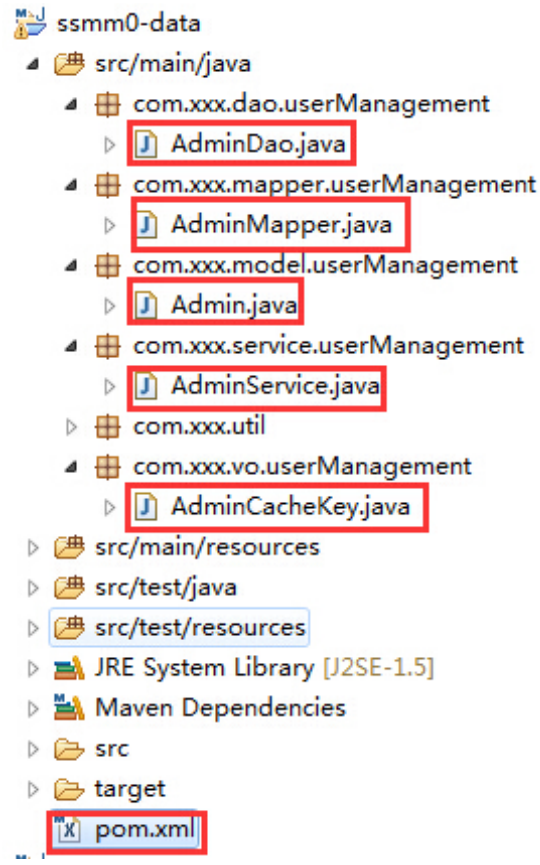
B1、手工在"本地服务部分"启动

B2、命令窗口下"memcached.exe -p 11211 -m 32 -c 1024 -d start"，该方法可以指定参数启动memcached服务，-p表示端口，-m表示分配的内存，-c表示最大的并发连接数

当然，我们在实际使用中，会装在Linux系统上，同时也需要指定一系列参数，例如分配的最大内存、最大并发数等等。

3.3、ssmm0-data

结构：



3.3.1、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>

    <!-- 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-data</artifactId>

    <name>ssmm0-data</name>
    <packaging>jar</packaging><!-- 只是作为其他模块使用的工具 -->

    <!-- 引入实际依赖 -->
    <dependencies>
        <!-- mysql -->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
        </dependency>
        <!-- 数据源 -->
        <dependency>
            <groupId>org.apache.tomcat</groupId>
            <artifactId>tomcat-jdbc</artifactId>
        </dependency>
        <!-- mybatis -->
        <dependency>
            <groupId>org.mybatis</groupId>
            <artifactId>mybatis</artifactId>
        </dependency>
        <dependency>
            <groupId>org.mybatis</groupId>
            <artifactId>mybatis-spring</artifactId>
        </dependency>
```

```
        <!-- servlet --><!-- 为了会用cookie -->
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>javax.servlet-api</artifactId>
        </dependency>
        <!-- bc-加密 -->
        <dependency>
            <groupId>org.bouncycastle</groupId>
            <artifactId>bcprov-jdk15on</artifactId>
        </dependency>
        <!-- cc加密 -->
        <dependency>
            <groupId>commons-codec</groupId>
            <artifactId>commons-codec</artifactId>
        </dependency>
        <!-- guava cache -->
        <dependency>
            <groupId>com.google.guava</groupId>
            <artifactId>guava</artifactId>
            <version>14.0.1</version>
        </dependency>
        <!-- 引入自定义cache模块 -->
        <dependency>
            <groupId>com.xxx.ssmm0</groupId>
            <artifactId>ssmm0-cache</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
    </dependencies>
</project>
```

说明：只引入了上边的ssmm0-cache模块。

3.3.2、Admin

```
package com.xxx.model.userManagement;

import java.io.Serializable;

import com.alibaba.fastjson.JSON;

/**
 * 管理员
 * 这里序列化，是为了向xmemcached中存储，否则会报异常；
 * 当然除了用序列化之外，还可以将admin对象转化为json串，然后进行存储
 */
public class Admin implements Serializable{

    private static final long serialVersionUID = 7149009421720474527L;

    private int id;
    private String username;
    private String password;

    public int getId() {
        return id;
    }

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

```

```
    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    //将json串转为Admin
    public static Admin parseJsonToAdmin(String jsonStr){
        try {
            return JSON.parseObject(jsonStr, Admin.class);
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }

    //将当前实例转化为json串
    public String toJson(){
        return JSON.toJSONString(this);
    }
}
```

说明：这里只添加了让该类实现java.io.Serializable接口，添加了序列号

注意：在实际使用中，把对象存入缓存有两种方式

- 序列化：使用上述的方式，或者使用其他序列化方式
- 将对象转化为json串，在该类中，有两个方法：一个将Admin-->Json，一个将Json-->Admin

这两种方式都可以，只是Java默认的序列化效率低且生成的码流大，但是使用方便，当然第二种方式使用也相当简单。

关于各种序列化的方式以及优缺点对比，查看《netty权威指南（第2版）》，或者查看我的"Java高效使用"系列的后续文章

3.3.3、AdminMapper

```
    /*************memcached*************/

    @Select("SELECT * FROM userinfo WHERE id = #{id}")
    @Results(value = {
            @Result(id = true, column = "id", property = "id"),
            @Result(column = "username", property = "username"),
            @Result(column = "password", property = "password") })
    public Admin selectById(@Param("id") int id);
```

说明：添加了上述按照ID查找用户的方法。

3.3.4、AdminDao

```
    /*****************memcached******************/
    public Admin getUserById(int id){
        return adminMapper.selectById(id);
```

```
4      }
```

说明：添加了上述方法。

3.3.5、AdminService

```
1      /********************memcached*******************/
2      public Admin findAdminById(int id) {
3          //从缓存中获取数据
4          Admin admin = (Admin)MemcachedUtil.getCache(CachePrefix.USER_MANAGEMENT, String.valueOf(id));
5          //若缓存中有，直接返回
6          if(admin != null){
7              return admin;
8          }
9          //若缓存中没有，从数据库查询
10         admin = adminDao.getUserById(id);
11         //若查询出的数据不为null
12         if(admin!=null){
13             //将数据存入缓存
14             MemcachedUtil.setCacheWithNoReply(CachePrefix.USER_MANAGEMENT, String.valueOf(id), admin);
15         }
16         //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）
17         return admin;
18     }
```

说明：添加了上述方法。

注意：

上述方法是缓存使用中最常见的模式，即"从缓存获取-->若没有，从数据库查询，存入缓存-->返回数据"，这就是guava cache的get(Object key)使用一个方法完成的原子操作。

3.4、ssmm0-userManagement

在该模块中，只在一个类中添加了一个方法。

AdminController.java

```
1      /**
2       * 根据id查找Admin
3       */
4      @ResponseBody
5      @RequestMapping("/findAdminById")
6      public Admin findAdminById(@RequestParam(value="id") int id){
7
8          return adminService.findAdminById(id);
9      }
```

说明：下边这个方法就是该模块中唯一添加的一个方法。

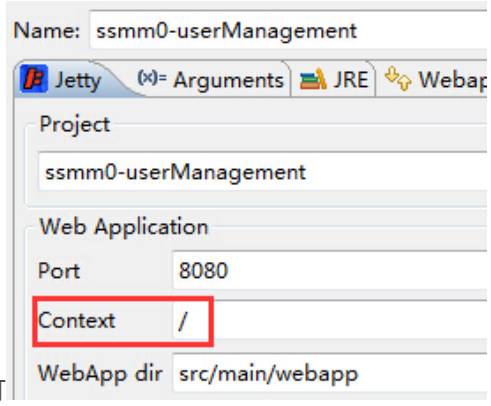
4、测试

在浏览器输入"localhost:8080/ssmm0-userManagement/admin/findAdminById?id=1"，这样就可以测试缓存，具体测试方式看《第七章 企业项目开发--本地缓存guava cache》

这里要说明的是两点：

- 由于在根pom.xml文件中将dev改成了服务器启动后默认使用的环境，所以在之后的测试中，不需要再修改环境了，但是实际上线时，一定要将prod改成默认环境才行

- 我想在上述URL中不输入项目名ssmm0-userManagement也可以访问相关资源，使用如下方式：run as-->run configurations..-->Context参数



改为"/"即可

关于memcached的相关内容和Xmemcached的相关内容，请参看下边链接或者我的"Java缓存相关"的后续文章：

http://code.google.com/p/memcached/wiki/NewStart?tm=6

https://code.google.com/p/xmemcached/wiki/User_Guide_zh