mybatis实战2

实现mybatis分页

在实际的项目中，分页是肯定需要的。而且是物理分页，不是内存分页。对于物理分页方案，不同的数据库，有不同的实现方法,对于 Mysql 来说 就是利用 limit offset,pagesize 方式来实现的。oracle 是通过 rownum 来实现的，如果你熟悉相关数据库的操作，是一样的很好扩展，本文以 Mysql 为例子来讲述。

动态SQL

Mybatis 的动态 SQL 语句是基于 OGNL 表达式的。可以方便的在 SQL 语句中实现某些逻辑. 总体说来 Mybatis 动态 SQL 语句主要有以下几类:

1. if 语句 (简单的条件判断)
2. choose (when,otherwize) ,相当于java 语言中的 switch ,与 jstl 中的choose 很类似.
3. trim (对包含的内容加上 prefix,或者 suffix 等，前缀，后缀)
4. where (主要是用来简化 sql 语句中 where 条件判断的，能智能的处理 and or ,不必担心多余导致语法错误)
5. set (主要用于更新时)
6. foreach (在实现 mybatis in 语句查询时特别有用)

具体看网站

这里只负责粘贴代码

这是接口中定义的

```
public Blog dynamicIfTest(Blog blog);

public  Blog dynamicChooseTest(Blog blog);

public Blog dynamicTrimTest(Blog blog);

public Blog dynamicWhereTest(Blog blog);

public void dynamicSetTest(Blog blog);

public List<Blog> dynamicForeachTest(List<Integer> ids);

public List<Blog> dynamicForeach2Test(int [] ids);

public List<Blog> dynamicForeach3Test(Map<String,Object> params);
```

这是在相应的xml中定义的

```
<select id="dynamicIfTest" parameterType="Blog" resultType="Blog">
SELECT  * from t_blog where 1=1
<if test="title != null">
and title = #{title}
</if>
    <if test="content !=null">
and content=#{content}
</if>
    <if test="owner !=null">
and owner =#{owner}
</if>

</select>


<select id="dynamicChooseTest" parameterType="Blog" resultType="Blog">
SELECT  * FROM t_blog WHERE  1=1
<choose>
        <when test="title !=null">
and title=#{title}
</when>
        <when test="content !=null">
and content=#{content}
</when>
       <otherwise>
and owner="owner1"
</otherwise>
    </choose>
```

```
</select>

<select id="dynamicTrimTest" parameterType="Blog" resultType="Blog">
SELECT * from t_blog
<trim prefix="where" prefixOverrides="and |or">
        <if test="title !=null">
title=#{title}
</if>
        <if test="content !=null">
and content=#{content}
</if>
        <if test="owner !=null">
or owner =#{owner}
</if>
    </trim>
</select>

<select id="dynamicWhereTest" parameterType="Blog" resultType="Blog">
SELECT  * FROM t_blog
<where>
        <if test="title !=null">
AND title=#{title}
</if>
        <if test="content !=null">
and content=#{content}
</if>
    </where>
</select>


<update id="dynamicSetTest" parameterType="Blog">
UPDATE  t_blog
<set>
    <if test="title !=null">
title=#{title},
</if>
        <if test="content != null">
content = #{content},
</if>
        <if test="owner !=null">
owner=#{owner}
</if>
    </set>
where id=#{id}
</update>

<select id="dynamicForeachTest" resultType="Blog">
SELECT  *  FROM  t_blog WHERE  id in
<foreach collection="list" index="index" item="item" open="(" separator= "," close=")">
#{item}
</foreach>
</select>

<select id="dynamicForeach2Test" resultType="Blog">
SELECT * FROM t_blog WHERE  id in
<foreach collection="array" index="index" item="item" open="(" separator="," close=")">
#{item}
</foreach>
</select>


<select id="dynamicForeach3Test" resultType="Blog">
SELECT  * FROM  t_blog WHERE  id IN
<foreach collection="ids" index="index" item="item" open="(" separator="," close=")">
#{item}
</foreach>
</select>
```

相关的测试类中

```
void TestdynamicIfTest() {
System.out.println("TestdynamicTest 测试");
Blog blog = new Blog();
blog.setContent("content1");
blog.setTitle("title1");
```

```
System.out.println(mapper.dynamicIfTest(blog));
}

void  TestdynamicChooseTest() {
System.out.println("TestdynamicChooseTest 测试");
Blog blog = new Blog();
System.out.println(mapper.dynamicChooseTest(blog));
}


void TestdynamicTrimTest(){
System.out.println("TestdynamicTrimTest 测试");
Blog blog = new Blog();
blog.setTitle("title1");
System.out.println(mapper.dynamicTrimTest(blog));
}

void TestdynamicWhereTest(){
System.out.println("TestdynamicWhereTest 测试");
Blog blog = new Blog();
blog.setContent("content1");
System.out.println(mapper.dynamicWhereTest(blog));
}

void TestdynamicSetTest() {
System.out.println("TestdynamicSetTest 测试");
Blog blog = new Blog();
blog.setId(1);
blog.setTitle("title1");
blog.setContent("content1");
mapper.dynamicSetTest(blog);
}

public void TestdynamicForeachTest() {
System.out.println("TestdynamicForeachTest 测试");
List<Integer> ids = new ArrayList<Integer>();
ids.add(1);
List<Blog> blogs = mapper.dynamicForeachTest(ids);
for (Blog blog : blogs)
System.out.println(blog);

}

public  void TestdynamicForeach2Test(){
System.out.println("TestdynamicForeach2Test 测试");
int [] array=new int[10];
array[1]=1;
List<Blog> blogs=  mapper.dynamicForeach2Test(array);
for (Blog blog:blogs){
System.out.println(blog);
     }
}

public  void TestdynamicForeach3Test(){
System.out.println("TestdynamicForeach3Test 测试");
Map<String, Object> ids = new HashMap<String, Object>();
int [] array=new int[10];
array[1]=1;
ids.put("ids",array);
List<Blog> blogs = mapper.dynamicForeach3Test(ids);
for (Blog blog:blogs){
System.out.println(blog);
     }

}
```

代码生成工具的使用

Mybatis 应用程序，需要大量的配置文件，对于一个成百上千的数据库表来说，完全手工配置，这是一个很恐怖的工作量. 所以 Mybatis 官方也推出了一个 Mybatis 代码生成工具的 jar 包. 今天花了一点时间，按照 Mybatis generator 的 doc 文档参考，初步配置出了一个可以使用的版本，我把源代码也提供下载,Mybatis 代码生成工具，主要有一下功能:

1. 生成 pojo 与 数据库结构对应
2. 如果有主键，能匹配主键
3. 如果没有主键，可以用其他字段去匹配
4. 动态 select,update,delete 方法
5. 自动生成接口(也就是以前的 dao 层)
6. 自动生成 sql mapper，增删改查各种语句配置，包括动态 where 语句配置
7. 生成 Example 例子供参考

注意： 这个网站上的列子是无法使用的 我使用的是另外一种方法

首先 注意 先在maven中加上这个mybatis代码生成工具

```
<build>
  <finalName>Demo</finalName>

  <plugins>
    <plugin>
      <groupId>org.mybatis.generator</groupId>
      <artifactId>mybatis-generator-maven-plugin</artifactId>
      <version>1.3.2</version>
    </plugin>
  </plugins>
</build>
```

然后
这个代码生成工具最重要的就是一个generatorConfig.xml的配置文件的

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE generatorConfiguration PUBLIC
        "-//mybatis.org//DTD MyBatis Generator Configuration 1.0//EN"
        "http://mybatis.org/dtd/mybatis-generator-config_1_0.dtd" >
<generatorConfiguration>
<!-- 配置mysql 驱动jar包路径.用了绝对路径 -->
<classPathEntry location="C:\Users\han\Desktop\Test\mysql\mysql-connector-java-5.1.22-bin.jar" />

    <context id="mysqlTables" targetRuntime="MyBatis3">
        <commentGenerator>
            <property name="suppressAllComments" value="true" />
            <property name="suppressDate" value="true" />
        </commentGenerator>

<!-- 数据库连接-->
<jdbcConnection driverClass="com.mysql.jdbc.Driver"
connectionURL="jdbc:mysql://localhost:3306/spitter"
userId="root"
password="liuziye"
/>

        <javaTypeResolver>
            <property name="forceBigDecimals" value="false"/>
        </javaTypeResolver>



<!--数据表对应的Model层-->
<javaModelGenerator targetPackage="com.mybatis.generator" targetProject="MAVEN">
            <property name="enableSubPackages" value="true"/>
            <property name="trimStrings" value="true"/>
        </javaModelGenerator>


<!--sql mapper隐射配置文件-->
<sqlMapGenerator targetPackage="mybatis" targetProject="MAVEN">
             <property name="enableSubPackages" value="true"/>
         </sqlMapGenerator>


<!-- 在ibatis2 中是dao层，但在mybatis3中，其实就是mapper接口 -->
<javaClientGenerator type="XMLMAPPER" targetPackage="com.mybatis.inter" targetProject="MAVEN">
            <property name="enableSubPackages" value="true"/>
        </javaClientGenerator>

<!-- 要对那些数据表进行生成操作，必须要有一个. -->
<table   schema="mybatis"
tableName="category" domainObjectName="Category"
enableCountByExample="false" enableUpdateByExample="false"
```

```
enableDeleteByExample="false" enableSelectByExample="false"
selectByExampleQueryId="false"
/>

    </context>
</generatorConfiguration>
```

配置好这个之后就可以了 然后直接在idea的maven的plugin中使用mybatis自动生成工具 然后在对应的target中就生成成功了

SqlsessionFactoryBean的使用
只粘贴代码

```
@Repository
public class UserDAOImpl extends SqlSessionDaoSupport implements UserDao{


public List<Article> getUserArticles(int userid) {

return this.getSqlSession().selectList("com.mybatis.inter.IUserOperation.getUserArticles",userid);
    }

@Autowired
    @Override
public void setSqlSessionFactory(SqlSessionFactory sqlSessionFactory) {
super.setSqlSessionFactory(sqlSessionFactory);
    }


public static void main(String[] args) {
ApplicationContext context = new ClassPathXmlApplicationContext("Spring/applicationContext.xml");
UserDao userDao= (UserDao) context.getBean("userDAOImpl");
System.out.println(userDao.getUserArticles(1));
    }
}
```

mybatis补充

1 是打印**SQL**语句
注意 要加上对应的**jar**包才能进行工作

这几个jar包分别为：

```
<dependency>
  <groupId>log4j</groupId>
  <artifactId>log4j</artifactId>
  <version>1.2.15</version>
</dependency>

<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-api</artifactId>
  <version>1.5.8</version>
</dependency>

<!-- https://mvnrepository.com/artifact/org.slf4j/slf4j-log4j12 -->
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-log4j12</artifactId>
  <version>1.5.8</version>
</dependency>
```

然后就在**resources**根目录下 写入**log4j.properties**文件的具体日志输出内容就可以了

```
log4j.rootCategory=info, stdout , R

log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=[QC] %p [%t] %C.%M(%L) | %m%n

log4j.appender.R=org.apache.log4j.DailyRollingFileAppender
log4j.appender.R.File=D:/my_log.log
log4j.appender.R.layout=org.apache.log4j.PatternLayout
log4j.appender.R.layout.ConversionPattern=%d-[TS] %p %t %c - %m%n

log4j.logger.com.ibatis=debug
log4j.logger.com.ibatis.common.jdbc.SimpleDataSource=debug
log4j.logger.com.ibatis.common.jdbc.ScriptRunner=debug
log4j.logger.com.ibatis.sqlmap.engine.impl.SqlMapClientDelegate=debug

log4j.logger.java.sql.Connection=debug
log4j.logger.java.sql.Statement=debug
log4j.logger.java.sql.PreparedStatement=debug,stdout
```

传递多个参数的方法

在用 Mybatis 做查询的时候，通常会传递多个参数，一般来说，这种情况下有两种解决办法：

- 利用 hashMap 去做。
- 利用 Mybatis 自身的多个参数传递方式去做。

hashMap的方法

在Sql映射文件中

```
<select id="dynamicHashMultParamter"  parameterType="map" resultType="Blog">
SELECT  * from t_blog  WHERE  id=#{id} and title = #{title}
</select>
```

接口

```
public List<Blog> dynamicHashMultParamter(Map<String, Object> params);
```

具体测试方法

```
void TestdynamicHashMultParamter() {
System.out.println("TestdynamicHashMultParamter 测试");
Map<String, Object> params = new HashMap<String, Object>();
params.put("id",2);
params.put("title", "title2");
System.out.println(mapper.dynamicHashMultParamter(params)
    );
}
```

自带的多个参数传递

sql映射文件中

```
<select id="dynamicHashMultParamter2"   resultType="Blog">
SELECT  * from t_blog  WHERE  id=#{id} and title = #{title}
</select>
```

接口中

```
public List<Blog> dynamicHashMultParamter2( int id,String title);
```

具体测试方法

```
void  TestdynamicHashMultParamter2() {
System.out.println("TestdynamicHashMultParamter2 测试");
System.out.println(mapper.dynamicHashMultParamter2(2,"title2"));
}
```

缓存机制

一种是mybatis自带的缓存
这种缓存开启十分简单
只需要在Sql映射文件中进行如下配置

```
<cache eviction="FIFO" flushInterval="60000" size="512" readOnly="true" />
```

就可以了

另外一种是ehcahe的缓存

首先是引用相关的jar包 在maven中进行配置 需要注意 这里的网站中推荐使用的mybatis-ehcahe的1.0.2版本有bug 换成1.0.1就好了

```
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-api</artifactId>
  <version>1.5.8</version>
</dependency>

<dependency>
  <groupId>org.mybatis.caches</groupId>
  <artifactId>mybatis-ehcache</artifactId>
  <version>1.0.1</version>
</dependency>

<dependency>
  <groupId>net.sf.ehcache</groupId>
  <artifactId>ehcache-core</artifactId>
  <version>2.6.5</version>
</dependency>
```

然后需要在resources下建立如下的ehcahe.xml

```
<ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://ehcache.org/ehcache.xsd">


    <cache name="default"
maxElementsInMemory="10000"
eternal="false"
timeToIdleSeconds="3600"
timeToLiveSeconds="10"
overflowToDisk="true"
diskPersistent="true"
diskExpiryThreadIntervalSeconds="120"
maxElementsOnDisk="10000"
/>
</ehcache>
```

这里还有一份其他人配置的 注意 如果在Spring集成中使用时name属性不能是default 估计是与Spring内部的机制有关

```
<ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="http://ehca
che.org/ehcache.xsd">
    <diskStore path="E:\test" />
<!--
    name：Cache的唯一标识
    maxElementsInMemory：内存中最大缓存对象数
    maxElementsOnDisk：磁盘中最大缓存对象数，若是0表示无穷大
    eternal：Element是否永久有效，一但设置了，timeout将不起作用
    overflowToDisk：配置此属性，当内存中Element数量达到maxElementsInMemory时，Ehcache将会Element写到磁盘中
    timeToIdleSeconds：设置Element在失效前的允许闲置时间。仅当element不是永久有效时使用，可选属性，默认值是0，
也就是可闲置时间无穷大
    timeToLiveSeconds：设置Element在失效前允许存活时间。最大时间介于创建时间和失效时间之间。仅当element不是永
久有效时使用，默认是0.，也就是element存活时间无穷大
    diskPersistent：是否缓存虚拟机重启期数据
    diskExpiryThreadIntervalSeconds：磁盘失效线程运行时间间隔，默认是120秒
    diskSpoolBufferSizeMB：这个参数设置DiskStore（磁盘缓存）的缓存区大小。默认是30MB。每个Cache都应该有自己的
一个缓冲区
    memoryStoreEvictionPolicy：当达到maxElementsInMemory限制时，Ehcache将会根据指定的策略去清理内存。默认策略
是LRU（最近最少使用）。你可以设置为FIFO（先进先出）或是LFU（较少使用）
-->
<defaultCache maxElementsInMemory="1000" maxElementsOnDisk="3600" eternal="false" overflowToDisk="true" t
imeToIdleSeconds="3600" timeToLiveSeconds="86400" diskExpiryThreadIntervalSeconds="120" memoryStoreEvicti
```

```
onPolicy="LRU"/>

    <cache name="article" maxElementsInMemory="1000" timeToIdleSeconds="3600" timeToLiveSeconds="86400" e
ternal="false" overflowToDisk="true" />
</ehcache>
```

另外在mybatis的SQL 映射文件加入

```
<cache type="org.mybatis.caches.ehcache.LoggingEhcache" >
    <property name="timeToIdleSeconds" value="3600"/><!--1 hour-->
<property name="timeToLiveSeconds" value="3600"/><!--1 hour-->
<property name="maxEntriesLocalHeap" value="1000"/>
    <property name="maxEntriesLocalDisk" value="10000000"/>
    <property name="memoryStoreEvictionPolicy" value="LRU"/>
</cache>
```

如果不配置ehcahe.xml的话也是可以的 只需要在sql映射文件中配置就好 因为在ehcahe中默认有一个ehcahe.xml 但是如果sql映射文件中没有配置的话 是真的起不到作用的

其实在SQL映射文件中是不需要再配置一次的

```
<cache type="org.mybatis.caches.ehcache.EhcacheCache"     >

</cache>
```

这样就能达到效果 毕竟在ehcahe.xml中已经配置了 只需要用ehcahe的就好

注意这里的配置都是全局缓存

如果在实现某些功能的时候不能用缓存 比如mybatis的分页功能 是可以禁止掉的

```
<select id="getUserArticles" parameterType="int" resultMap="resultUserArticleList" useCache="false">
select USER.id ,USER.username,user.userAddress,article.id aid,article.title,article.content
    FROM  USER , article WHERE  user.id=article.userid and user.id=#{id}
</select>
```