# Mybatis简介

# MyBatis入门

## 开发环境准备

## MyBatis基本构成

1、MyBatis核心组件

1. SqlSessionFactoryBuilder：根据配置信息或代码来生成SqlSessionFactory的实例
2. SqlSessionFactory：工厂，生产SqlSession(会话)
3. SqlSession：可以发送SQL来执行并返回结果，也可以获取Mapper接口的实现
4. SQL Mapper：由Java接口和XML文件(或注解)构成，需要给出对应的SQL和映射规则，负责执行SQL并返回结果

* Java接口+XML文件
* Java接口+注解

### 构建SqlSessionFactory

1、使用XML方式构建

1. 获取数据库连接实例的数据源(DataSource)
2. 决定事务范围和控制方式的事务管理器(TransactionManager)
3. 映射器(SQL Mapper)

2、使用代码方式构建

1. 构建Configuration对象
2. 往该对象中注册构建SqlSessionFactory所需要的信息

3、两种方式本质相同

SqlSessionFactory sqlSessionFactory=  
new SqlSessionFactoryBuilder.build(configuration);

SqlSessionFactory sqlSessionFactory=  
new SqlSessionFactoryBuilder.build("\*.xml");

4、推荐采用XML构建方式

### 创建SqlSession

1、惯例

SqlSession sqlSession=null;

try{

sqlSession=sqlSessionFactory.openSession();

//do something

sqlSession.commit();

}catch(Exception ex){

sqlSession.rollback();

}finally{

if(sqlSession!=null){

sqlSession.close();

}

}

### 映射器

1、映射器由Java接口和XML文件(或者Java接口和注解)组成

1. 定义参数类型
2. 描述缓存
3. 描述SQL语句
4. 定义查询结果和POJO的映射关系

2、XML文件中mapper的namespace属性值就是Java接口的全限定名

### 使用方式

1、

XXXDAO xDAO=sqlSession.getMapper(XXXDAO.class);

xDAO.<function>(<args>)

2、

Role role=  
sqlSession.selectOne("<全限定类名>.<方法名>",<args>);

3、SqlSessionTemplate的方法

1. update(String statement,<可能有其他参数>)
2. delete(String statement,<可能有其他参数>)
3. insert(String statement,<可能有其他参数>)
4. select(String statement,<可能有其他参数>)
5. selectOne(String statement,<可能有其他参数>)
6. selectList(String statement,<可能有其他参数>)
7. selectMap(String statement,<可能有其他参数>)
8. selectCursor(String statement,<可能有其他参数>)
9. rollback(String statement,<可能有其他参数>)
10. commit(String statement,<可能有其他参数>)

## 生命周期

1、SqlSessionFactoryBuilder

1. SqlSessionFactoryBuilder利用XML(提取到流对象)或者Java编码(Configuration)对象来构建SqlSessionFactory
2. 通过它可以构建多个SessionFactory
3. 它的作用就是一个构建器，一旦构建了SqlSessionFactory，它的作用就已经完结，失去了存在的意义
4. 它的生命周期只能存在于方法的局部，它的作用就是生成SqlSessionFactory对象

2、SqlSessionFactory

1. SqlSessionFactory的作用是创建SqlSession，而SqlSession就是一个会话，相当于JDBC中的Connection对象
2. 每次应用程序需要访问数据库，我们就要通过SqlSessionFactory创建SqlSession，所以SqlSessionFactory应该在Mybatis应用的整个生命周期中
3. 每个数据库只对应一个SqlSessionFactory

3、SqlSession

* SqlSession是一个会话，相当于JDBC的一个Connection对象，它的生命周期应该是在请求数据库处理事务的过程中
* 它是一个线程不安全的对象
* 每次创建它后，都必须及时关闭它，避免浪费资源

4、Mapper

* Mapper是一个接口，没有任何实现类，其作用是发送SQL，然后返回需要的结果，或者执行SQL从而修改数据库的数据
* 它应该在一个SqlSession事务方法之内，是一个方法级别的东西

# 配置

1、整个MyBatis配置XML文件的层次结构

<?xml version="1.0" encoding="utf-8"?>

<configuration>

<properties/> <!--属性-->

<settings/> <!--设置-->

<typeAliases/> <!--类型别名-->

<typeHandlers/> <!--类型处理器-->

<objectFactory/> <!--对象工厂-->

<plugins/> <!--插件-->

<enviroments> <!--配置环境-->

<environment> <!--环境变量-->

<transactionManager/> <!--事务管理器-->

<dataSource/> <!--数据源-->

</environment>

</enviroments>

<databaseIdProvider/> <!--数据库厂商标识-->

<mappers/> <!--映射器-->

</configuration>

## properties元素

1、properties是一个配置属性的元素，让我们能在配置文件的上下文中使用它

2、Mybatis提供3种配置方式

1. property子元素
2. properties配置文件
3. 程序参数传递

### property子元素

1、示例

<!--定义-->

<properties>

<property name="driver" value="com.mysql.jdbc.Driver"/>

<property name="url" value="jdbc:mysql://localhost:3306/mybatis"/>

<property name="username" vlaue="root"/>

<property name="password" value="123456"/>

</properties>

<!--使用-->

<dataSource type="POOLED">

<property name="driver" value="${driver}"/>

<property name="url" value="${url}"/>

<property name="username" vlaue="${username}"/>

<property name="password" value="${password}"/>

### properties配置文件

1、示例

#数据库配置文件:jdbc.properties

driver=com.mysql.jdbc.Driver

url=jdbc:mysql://localhost:3306/mybatis

username=root

password=123456

<properties resource="jdbc.properties"/>

### 程序参数传递

### 优先级

1、优先级

1. properties元素体内指定的属性先被读取
2. 根据properties元素中的resource属性读取类路径下属性文件，或者根据url属性指定的路径读取属性文件，并覆盖已读取的同名属性
3. 读取作为方法参数传递的属性，并覆盖已读取的同名属性

2、首选properties文件

## 设置

1、属性介绍

1. mapUnderscoreToCamelCase：是否开启自动驼峰命名规则(camel case)映射，即从经典数据库列名A\_COLUMN到经典的Java属性名aColumn的类似映射，默认false

## 别名

1、别名(typeAliases)是一个指代的名称，用一个简短的名称去指代一个类全限定名，而这个别名可以在MyBatis上下文中使用

2、MyBatis别名不区分大小写

3、一个typeAliases的实例是在解析配置文件时生成的，然后长期保存在Configuration对象中

### 系统定义别名

1、基本所有的数据类型，以及常用的容器类型，以及String等都有系统定义的别名，例如

1. \_byte:byte
2. byte:Byte
3. list:List
4. ...

### 自定义别名

1、为单个类定义别名，示例

<typeAliases>

<typeAliases alias="role" type="com.learn.chapter2.po.Role"/>

</typeAliases>

2、扫描包名的形式自定义别名

<typeAliases>

<package name="com.learn.chapter2.po"/>

</typeAliases>

可以添加@Alias指定别名，若没有注解，则默认将类名第一个字母变成小写

## typehandler类型处理器

1、**MyBatis在预处理语句(PreparedStatement)中设置一个参数时，或者从结果集(ResultSet)中取出一个值时，都会用注册了typeHandler进行处理**

2、typeHandler常用的配置为Java类型(javaType)、JDBC类型(jdbcType)。TypeHandler的作用就是将参数从javaType转化为jdbcType，或者从数据库取出结果时把jdbcType转化为javaType

### 系统定义的typeHandler

1、数据库类型

* 串数据类型

1. CHAR：1-255个字符的定长串，长度在创建时指定，否则假定为CHAR(1)
2. ENUM：接受最多64K字节?个串组成的一个预定义集合的某个串
3. LONGTEXT：与TEXT相同，最大长度4GB字节?
4. MEDIUMTEXT：与TEXT相同，最大长度为16K字节?
5. SET：接受最多64个串组成的一个预定义集合的零个或多个串
6. TEXT：最大长度为64K字节?的变长文本
7. TINYTEXT：与TEXT相同，最大长度为255字节
8. VARCHAR：长度可变，最多不超过255字节

* 数值数据类型

1. BIT：位字段，1-64位
2. BIGINT：整数值，-9223372036854775808~9223372036854775807，如果是UNSIGNED，则为0~18446744073709551615
3. BOOLEAN(BOOL)：布尔标志，0或者1，主要用于开关标志
4. DECIMAL(DEC)：精度可变的浮点值
5. DOUBLE：双精度浮点值
6. FLOAT：单精度浮点值
7. INT(INTEGER)：整数值，-2147483648~2147483647，若为UNSIGNED，则为0~4294967295
8. MEDIUMINT：整数值，-8388608~8088607，若为UNSIGNED，则为0~16777215
9. REAL：4字节浮点数
10. SMALLINT：整数值，支持-32768~32767，若为UNSIGNED，则0~65535
11. TINYINT：整数值，-128-127，若为UNSIGNED，则0~255

* 日期和时间数据类型

1. DATE：表示1000-01-01 9999-12-31的日期，格式为YYYY-MM-DD
2. DATETIME：DATE和TIME的组合
3. TIMESTAMP：功能和DATETIME相同(范围较小)
4. TIME：格式为HH:MM:SS
5. YEAR：用2位数字表示，范围70(1970)~69(2069)，用四位数表示，范围1901~2155

* 二进制数据类型

1. BLOB：Blob最大长度64KB
2. MEDIUMBLOB：Blob最大长度16MB
3. LONGBLOB：Blob最大长度4GB
4. TINYBLOB：Blob最大长度255字节

2、系统注册的typeHandler

|  |  |  |
| --- | --- | --- |
| 类型处理器 | Java类型 | JDBC类型 |
| BooleanTypeHandler | java.lang.Boolean,boolean | 数据库兼容的BOOLEAN |
| ByteTypeHandler | java.lang.Byte,byte | 数据库兼容的NUMERIC/BYTE |
| ShortTypeHandler | java.lang.Short,short | 数据库兼容的NUMERIC/SHORT INTEGER |
| IntegerTypeHandler | java.lang.Integer,int | 数据库兼容的NUMERIC或INTEGER |
| LongTypeHandler | java.lang.Long,long | 数据库兼容的NUMERIC或LONG INTEGER |
| FloatTypeHandler | java.lang.Float,float | 数据库兼容的NUMERIC或FLOAT |
| DoubleTypeHandler | java.lang.Double,double | 数据库兼容的NUMERIC或DOUBLE |
| BigDecimalTypeHandler | java.math.BigDecimal | 数据库兼容的NUMERIC或DECIMAL |
| StringTypeHandler | java.lang.String | CHAR，VARCHAR |
| ClobTypeHandler | java.lang.String | CLOB，LONGVARCHAR |
| NStringTypeHandler | java.lang.String | NVARCHAR，NCHAR |
| NClobTypeHandler | java.lang.String | NCLOB |
| ByteArrayTypeHandler | byte[] | 数据库兼容的字节流类型 |
| BlobTypeHandler | byte[] | BLOB，LONGVARBINARY |
| DateTypeHandler | java.util.Date | TIMESTAMP |
| DateOnlyTypeHandler | java.util.Date | DATE |
| TimeOnlyTypeHandler | java.util.Date | TIME |
| SqlTimestampTypeHandler | java.sql.Timestamp | TIMESTAMP |
| SqlDateTypeHandler | java.sql.Date | DATE |
| SqlTimeTypeHandler | java.sql.Time | Time |
| ObjectTypeHandler | Any | OTHER或未指定类型 |
| EnumTypeHandler | Enumeration Type | VARCHAR或任何兼容字符串类型，存储枚举的名称(而不是索引) |
| EnumOrdinal | Enumeration Type | 任何兼容的NUMERIC或DOUBLE类型，存储枚举的索引(而不是名称) |

### 自定义typeHandler

## ObjectFactory

1、当Mybatis在构建一个结果返回的时候，都会使用ObjectFactory(对象工厂)去构建POJO，在MyBatis中可以定制自己的对象工厂

2、一般来说使用默认的ObjectFactory：org.apache.ibatis.reflection.factory.DefaultObjectFactory来提供服务

3、示例

<objectFactory type="com.learn.chapter3.objectFactory.MyObjectFactory">

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

</objectFactory>

## 插件

1、后面介绍

## environments配置环境

### 概述

1、配置环境可以注册多个数据源(dateSource)，每一个数据源分为两大部分：一个是数据库源的配置，另外一个是数据库事务(transactionManager)的配置

<environments default="development">

<environment id="development">

<transactionManager type="JDBC"/>

<property name="autoCommit" value="false"/>

</transactionManager>

<dataSource type="POOLED">

<property name="driver" value="com.mysql.jdbc.Driver"/>

<property name="url" value="jdbc:mysql://localhost:3306/mybatis"/>

<property name="username" value="root"/>

<property name="password" value="hcflh19930101"/>

</dataSource>

</environment>

</environments>

* <environments/>中的default：在缺省的情况下，将启用哪个数据源配置
* <environment/>配置一个数据源，id设置这个数据源的标识，以便在MyBatis上下文中使用它
* <transactionManager/>：配置数据库事务，type属性有三种配置方式

1. JDBC，采用JDBC方式管理事务
2. MANAGED，采用容器方式管理事务，在JNDI数据源中常用
3. 自定义，使用自定义数据库事务管理办法，适用于特殊应用

* <dateSource/>：配置数据源连接信息，type属性提供对数据源连接方式的配置，提供以下几种配置方式

1. UNPOOLED，非连接池数据库
2. POOLED，连接池数据库
3. JNDI，JNDI数据源
4. 自定义数据源

### 数据库事务

1、数据库事务是交由SqlSession去控制的，可以通过SqlSession的提交(commit)或者回滚(rollback)

2、大部分工作环境下，我们都会使用Spring框架来控制它

### 数据源

1、MyBatis内部提供了3中数据源的实现方式

1. UNPOOLED，非连接池，由org.apache.ibatis.datasource.unpooled.UnpooledDataSource实现
2. POOLED，连接池，由  
   org.apache.ibatis.datasource.pooled.PooledDataSource实现
3. JNDI，org.apache.ibatis.datasource.jndi.JndiDataSourceFactory获取数据源

## databaseIdProvider数据库厂商标识

1、在相同数据库厂商的环境下，数据库厂商标识没有什么意义

2、MyBatis可能会运行在不同厂商的数据库中，它为此提供一个数据库标识，并提供自定义，它的作用在于指定SQL到对应的数据库厂商提供的数据库中运行

## 引入映射器的方法

1、映射器是MyBatis最复杂、最核心的组件

2、引入映射器由以下几种方法

1. 用文件路径引入映射器

<mappers>

<mapper resource="com/learn/chapter3/mapper/roleMapper.xml"/>

</mappers>

1. 用包名引入映射器

<mappers>

<package name="com.learn.chapter3.mapper"/>

</mappers>

1. 用类注册引入映射器

<mappers>

<mapper class="com.learn.chapter3.mapper.UserMapper"/>

<mapper class="com.learn.chapter3.mapper.RoleMapper"/>

</mappers>

1. 用userMapper.xml引入映射器

<mappers>

<mapper url="file:///var/mappers/com/learn/chapter3/mapper/userMapper.xml"/>

<mapper url="file:///var/mappers/com/learn/chapter3/mapper/roleMapper.xml"/>

</mappers>

# 映射器

## 映射器的主要元素

1、映射器可以使用元素列表

|  |  |  |
| --- | --- | --- |
| 元素名称 | 描述 | 备注 |
| select | 查询语句、最常用、最复杂的元素 | 可自定义参数、返回结果集 |
| insert | 插入语句 | 执行后返回一个整数，代表插入的行数 |
| update | 更新语句 | 执行后返回一个整数，代表更新的条数 |
| delete | 删除语句 | 执行后返回一个整数，代表删除的条数 |
| parameterMap | 定义参数映射关系 | 即将被删除的元素，不建议使用 |
| sql | 允许定义一部分SQL，然后在各个地方引用 | 例如，一张表列名，我们可以一次定义，在多个SQL语句中引用 |
| resultMap | 用来描述从数据库结果集中来加载对象，它是最复杂、最强大的元素 | 提供映射规则 |
| cache | 给定命名空间的缓存配置 |  |
| cache-ref | 其他命名空间缓存配置的引用 |  |

## select元素

1、主要的元素

1. id：它和Mapper命名空间组合起来是唯一的，供MyBatis调用
2. parameterType：可以给出类的全命名，也可以给出类的别名，但使用别名必须是MyBatis内部定义或字节
3. resultType：定义类的全路径，在允许自动匹配的情况下，结果集将通过JavaBean的规范映射，或者使用系统别名(3.3)，不能与resultMap同时使用
4. resultMap：它是映射集的引用，执行强大的映射功能，resultMap给予我们自定义映射规则的机会，该元素引用<resultMap>元素

### 自动映射

1、当参数autoMappingBehavior不为NONE时，MyBatis会提供自动映射的功能，只要返回的SQL列名和JavaBean属性名一致，MyBatis就会帮助我们回填这些字段而无需任何配置

2、但实际上，大部分的数据库规范都要求每个单词用下划线分隔，而Java则用驼峰命名法来命名，于是使用列别名就可以使得MyBatis自动映射，或者直接在配置文件中开启驼峰命名方式

3、自动映射可以在settings元素中配置autoMappingBehavior属性值来设置其策略

1. NONE，取消自动映射
2. PARTIAL，只会自动映射，没有定义嵌套结果集映射的结果集，其为默认值
3. FULL，自动映射任意复杂的结果集

4、如果数据库是规范命名，即每一个单词都用下划线分隔，POJO采用驼峰命名法，那么可以将mapUnderscoreToCamelCase设为true

### 传递多个参数

1、使用MyBatis提供的Map接口作为参数来实现它，不推荐使用

2、使用注解方式传递参数

* MyBatis的参数注解@Param(org.apache.ibatis.annotations.Param)来实现想要的功能
* 为接口的参数标注注解，示例如下

public List<Role> findRole(@Param("roleName") String rolename);

3、使用JavaBean传递参数

1. MyBatis允许组织一个JavaBean，通过简单的setter和getter方法设置参数，这样就可以提高我们的可读性
2. 这样在SQL语句中通过#{<属性名>}就会映射到对该JavaBean入参的getter方法上

4、选择建议

* 当参数数量<5个时，采用@Param
* 当参数数量>=5个时，采用JavaBean方式

### 使用resultMap映射结果集

1、示例

<resultMap id="roleResultMap" type="com.learn.chapter4.pojo.Role">

<id property="id" column="id"/>

<result property="roleName" column="role\_name"/>

<result property="note" column="note"/>

</resultMap>

<select parameterType="long" id="getRole" resultMap="roleResultMap">

SELECT id, role\_name, note from t\_role where id=#{id}

</select>

* resultMap的id属性：定义了唯一标识，供其他地方引用，type属性定义它对应哪个JavaBean
* id元素：指明哪个属性为主键
* result元素：定义普通列的映射关系

## insert元素

### 概述

1、insert元素，相对于select元素要简单许多，MyBatis会在执行插入之后返回一个整数，以表示你进行操作后插入的记录数

2、主要元素

1. id：它和Mapper命名空间组合起来唯一，作为唯一标识提供给MyBatis
2. parameterType：可以给出类的全名，也可以是一个别名，但别名必须是MyBatis内部定义或者自定义的别名
3. keyProperty：表示哪个列作为属性的主键，不能和keyColumn同时使用
4. keyColumn：指明第几列是主键，不能和keyProperty同时使用，只接受整形参数

### 主键回填和自定义

1、MySQL里面的主键需要根据一些特殊的规则去生成，在插入后我们往往需要获得这个主键，以便未来的操作，MyBatis提供了实现的方法

2、使用keyProperty属性指定哪个是主键字段，同时使用useGeneratedKeys属性告诉MyBatis这个主键是否能使用数据库内置策略生成

3、注意点：若要MyBatis自动回填，那么数据库列名必须与POJO属性名一致，例如列名为id，则必须有setId()方法，列名为role\_id，则必须有setRole\_id()方法，否则回填失败，目前还没有找到映射方法

4、insert后需要用SqlSession的对象执行commit才会刷新到数据库中，否则数据库中不会有插入的数据，但是自增量会一直递增

5、主键回填注解用法

@Insert({

"INSERT INTO call\_crm\_user(" +

"account " +

") " +

"VALUES( " +

"#{crmUser.account ,jdbcType=VARCHAR} " +

") "

})

@Options(useGeneratedKeys = true, keyProperty = "crmUser.id")

Integer insertCrmUser(@Param("crmUser") CrmUser crmUser);

## update元素和delete元素

1、update元素和delete元素在执行完毕后会返回一个整数，标出执行后影响的记录条数

2、update和delete后需要用SqlSession的对象执行commit才会刷新到数据库中，否则数据库中不会有更新

## 参数

1、通过指定JdbcType、JavaType可以明确使用哪个typeHandler去处理参数，或者制定一些特殊的东西，但是需要强调的是：定义参数属性的时候，MyBatis不允许换行

### 配置参数

1、参数配置示例

#{age,javaType=int,jdbcType=NUMERIC}

#{age,javaType=int,jdbcType=NUMERIC,typeHandler=MyTypeHandler}

#{price,javaType=double,jdbcType=NUMERIC,numericScale=2}

## sql元素

## resultMap结果映射集

## 级联

## 缓存

# 动态SQL

## 概述

1、MyBatis的动态SQL包括以下几种元素

|  |  |  |
| --- | --- | --- |
| 元素 | 作用 | 备注 |
| if | 判断语句 | 单条件分支判断 |
| choose(when,otherwise) | 相当于Java的case when语句 | 多条件分支判断 |
| trim(there,set) | 辅助元素 | 用于处理SQL拼装问题 |
| foreach | 循环语句 | 在in语句等列举条件常用 |

## if元素

1、if元素是最常用的判断语句，相当于Java的if语句，常与test属性联合使用

2、示例

<select id="findRoles" parameterType="string" resultMap=roleResultMap">

<select role\_no,role\_name,note from t\_role where 1=1

<if test="roleName != null and roleName !=' '">

and role\_name like concat('%',#{roleName},'%')

</if>

</select>

## choose、when、otherwise元素

1、在需要第三种选择或者更多选择时，需要switch...case...default语句，在映射器的动态语句中choose、when、otherwise元素承担了这个功能

2、示例

<select id="findRoles" parameterType="role" resultMap="roleResultMap">

<select role\_no, role\_name, note from t\_role

where 1 = 1

<choose>

<when test="roleNo != null and roleNo !=' '">

AND role\_no = #{roleNo}

</when>

<when test="roleName != null and roleName !=' '">

AND role\_name like concat('%',#{roleName},'%')

</when>

<otherwise>

AND note is not null

</otherwise>

</choose>

</select>

* 当角色编号不为空，则只用角色编号作为条件查询
* 当角色编号为空，角色名不为空，则用角色名作为条件进行模糊查询
* 当角色编号和角色名都为空，则要求角色备注不为空

## trim、where、set元素

1、前两个例子中，有where 1=1这个片段，如果没有这个片段，则最后组合起来的SQL可能是有语法错误的

2、<未完成>

## foreach元素

1、foreach元素是一个循环语句，作用是遍历集合，可以很好地支持数组和List、Set接口的集合，对此提供遍历功能

2、示例

<select id="findUserBySex" resultType="user">

select \* from t\_user where sex in

<foreach item="sex" index="index" collection="sexList" open="("

separator="," close=")">

#{sex}

</foreach>

</select>

* collection配置的"sexList"是传进来的参数名称，它可以使数组、List、Set等集合
* item配置的是循环中当前的元素
* index配置的是当前元素在集合的位置下标
* open和close配置的是以什么符号将这些元素包装起来
* separator是各个元素的间隔符

## test的属性

1、test的属性用于条件判断的语句中

## bind元素

1、bind元素的作用是通过OGNL表达式去定义一个上下文变量，这样更方便我们使用

2、当我们进行模糊查询时，如果是MySQL数据库，我们常用到concat"%"和参数相连接，而在Oracle数据库中，用连接符号"||"，这样SQL就需要提供两种形式去实现，如果有了bind元素，就可以不必使用数据库的语言，只要使用MyBatis的语言即可与所需参数相连

<select id="findRole" resultType="com.learn.chapter5.RoleBean">

<bind name="pattern" value="'%'+\_parameter+'%'/>

SELECT id,role\_name AS roleName WHERE role\_name LIKE #{pattern}

</select>

* "\_parameter"代表的是传递进来的参数，和通配符连接后，赋值给pattern，就可以在select语句中使用这个变量进行模糊查询

# MyBatis的解析和运行原理

## 涉及的技术难点简介

1、为什么要用代理模式

1. 一方面可以控制访问真正的服务对象，提供额外服务
2. 另一方面有机会通过重写一些类来满足特定的需要

2、动态代理分为两种

1. JDK反射机制提供的代理，必须提供接口
2. CGLIB代理，不需要提供接口，可以代理任意类

### 反射技术

1、反射的最大好处是配置性大大提高，就如同Spring IOC容器一样，我们可以给很多配置设置参数，大大提高了Java的灵活性和可配置性，降低模块之间的耦合度

### JDK动态代理

1、JDK动态代理，由JDK的java.lang.reflect.\*包提供支持，需要完成以下步骤

1. 编写服务类和接口，这个是真正的服务提供者，在JDK代理中接口是必须的
2. 编写代理类，提供绑定和代理方法

### CGLIB动态代理

1、JDK提供的动态代理存在一个缺陷