

实验六 MyBatis 基础应用

一、基础实验——MyBatis 框架搭建

（一）实验目的

- 1、掌握 MyBatis 开发环境搭建的基本步骤，了解 MyBatis 的基本原理；
- 2、观察持久化类与数据库表的映射关系，观察相应的 Mybatis 映射文件配置，并能够做简单应用；
- 3、观察 MyBatis 配置文件中的主要元素及属性配置，并能够做简单应用。

（二）基本知识与原理

- 1、MyBatis 是一个基于 Java 的持久层 ORM（Object-Relational Mapping）框架，其提供的持久层框架包括 SQL Maps 和 Data Access Objects（DAO），几乎消除了几乎所有的 JDBC 代码和参数的手工设置以及结果集的检索；
- 2、MyBatis 使用简单的 XML 或注解用于配置和原始映射，将接口和 Java 的 POJOs 映射成数据库中的记录；

（三）实验内容及步骤

- 1、登录 <https://dev.mysql.com/downloads/connector/j/> 站点，下载并安装 MySQL 数据库；
- 2、在 MySQL 中创建一个名称为 mybatisdb 的数据库，并在该数据库中创建一个名称为 user 的数据表，表结构如表 6-1 所示：

表 6-1 user 数据表

字段名称	类型	中文含义
uid	INTEGER(11), Primary key, Not Null	用户编号
uname	VARCHAR(20)	用户名
usex	VARCHAR(20)	用户性别

- 3、在表 customer 中添加若干条记录；
- 4、登录 <http://downloads.mysql.com/archives/c-j/> 站点，下载 MySQL JDBC 驱动；

- 5、在 Eclipse 中新建工程 mybatis-prj1，并添加 MySQL 驱动程序库文件到工程中；
- 6、登录 <https://github.com/mybatis/mybatis-3/releases> 站点，下载 MyBatis 发布版，将核心 jar 包、依赖 jar 包添加到工程 mybatis-prj1 中，如图 6-1 所示；

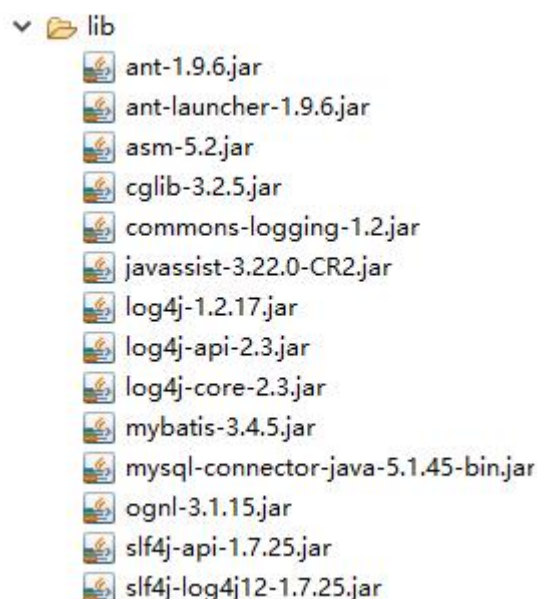


图 6-1 MyBatis 相关 JAR 包

- 7、创建一个名为 cn.edu.zjut.po 包，在该包中创建持久化类 MyUser，类中声明的属性与数据表 user 的字段一致；
- 8、创建映射文件：创建一个名为 cn.edu.zjut.mapper 的包，在该包中创建映射文件 UserMapper.xml，并在其中配置操作数据库的 SQL 语句，具体代码如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper
PUBLIC "-//Mybatis.org/DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="cn.edu.zjut.mapper.UserMapper">
    <select id="selectUserById" parameterType="Integer"
        resultType="cn.edu.zjut.po.MyUser">
        Select * from user where uid = #{uid}
    </select>
    <select id="selectAllUser" resultType="cn.edu.zjut.po.MyUser">
        Select * from user
    </select>
    <insert id="addUser" parameterType="cn.edu.zjut.po.MyUser">
        Insert into user (uname,usex) values(#{uname},#{usex})
    </insert>
    <update id="updateUser" parameterType="cn.edu.zjut.po.MyUser">
        update user set uname=#{uname}, usex=#{usex} where uid=#{uid}
    </update>
    <delete id="deleteUser" parameterType="Integer">
```

```
        delete from user where uid = #{uid}
    </delete>
</mapper>
```

- 9、创建配置文件：创建 MyBatis 的核心配置文件 mybatis-config.xml，在该文件中配置了数据库环境和映射文件的位置，具体代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE configuration
PUBLIC "-//mybatis.org/DTD Config 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
    <environments default="development">
        <environment id="development">
            <transactionManager type="JDBC"/>
            <dataSource type="POOLED">
                <property name="driver" value="com.mysql.jdbc.Driver"/>
                <property name="url"
                    value="jdbc:mysql://localhost:3306/mybatisdb"/>
                <property name="username" value="root"/>
                <property name="password" value="root"/>
            </dataSource>
        </environment>
    </environments>
    <mappers>
        <mapper resource="cn/edu/zjut/mapper/UserMapper.xml"/>
    </mappers>
</configuration>
```

- 10、创建测试类：创建一个名为 cn.edu.zjut.test 的包，在该包中创建测试类 MyBatisTest，在其中使用输入流读取配置文件，然后根据配置信息构建 SqlSessionFactory 对象，再通过 SqlSessionFactory 对象创建 SqlSession 对象，并使用 SqlSession 对象的方法执行数据库操作，部分代码如下：

```
package cn.edu.zjut.test;
import java.io.IOException;
import java.io.InputStream;
import java.util.List;
.....

public class MyBatisTest {
    Public static void main(String[] args){
        try {
            InputStream config=Resources.
                getResourceAsStream("mybatis-config.xml");
            SqlSessionFactory ssf=
                new SqlSessionFactoryBuilder().build(config);
```

```

        SqlSession ss=ssf.openSession();
        //查询一个用户
        MyUser mu =
            ss.selectOne("cn.edu.zjut.mapper.UserMapper.selectUserById", 1)
        System.out.println(mu);
        //添加一个用户
        MyUser addmu=new MyUser();
        addmu.setUname("张三");
        addmu.setUsex("男");
        ss.insert("cn.edu.zjut.mapper.UserMapper.addUser", addmu);
        //修改一个用户
        .....
        //删除一个用户
        .....
        //查询所有用户
        .....
        ss.commit();
        ss.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
}

```

11、运行以上测试类，并记录运行结果。

（四）实验要求

1、填写并上交实验报告，报告中应包括：

- （1）运行结果截图；
- （2）简述 MyBatis 的工作原理；
- （3）简述 MyBatis 和 Hibernate 的异同点和优缺点；
- （4）碰到的问题及解决方案或思考；
- （5）实验收获及总结。

2、上交程序源代码，代码中应有相关注释。

二、提高实验——映射器

（一）实验目的

- 1、进一步熟悉 MyBatis 应用的开发方法；
- 2、掌握 MyBatis 映射器的作用，熟悉映射文件中主要元素及其属性的含义和作用，并能进行正确应用；
- 3、掌握 MyBatis 映射文件中的输入参数映射和输出结果映射。

（二）基本知识 with 原理

- 1、映射器是 MyBatis 最复杂且最重要的组件，由一个接口加上 XML 文件（SQL 映射文件）组成；
- 2、SQL 映射文件的常用配置元素如表 6-2 所示：

表 6-2 SQL 映射文件的常用配置元素

元素名称	描 述	备 注
select	查询语句，最常用、最复杂的元素之一	可以自定义参数，返回结果集等
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，代表更新的行数
delete	删除语句	执行后返回一个整数，代表删除的行数
sql	定义一部分 SQL，在多个位置被引用	例如一张表，列名一次定义，可以在多个 SQL 语句中使用
resultMap	用来描述从数据库结果集中来加载对象，是最复杂、最强大的元素	提供映射规则

- 3、映射文件中<select>元素的常用属性如表 6-3 所示：

表 6-3 <select>元素的常用属性

属性名称	描 述
id	它和 Mapper 的命名空间组合起来使用，是唯一标识符，供 MyBatis 调用
parameterType	表示传入 SQL 语句的参数类型的全限定名或别名。是个可选属性，MyBatis 能推断出具体传入语句的参数。
resultType	SQL 语句执行后返回的类型（全限定名或者别名）。如果是集合类型，返回的是集合元素的类型。返回时可以使用 resultType 或 resultMap 之一
resultMap	它是映射集的引用，与< resultMap>元素一起使用。返回时可以使用 resultType 或 resultMap 之一
flushCache	它的作用是在调用 SQL 语句后，是否要求 MyBatis 清空之前查询本地缓存和二级缓存。默认值为 false。如果设置为 true，则任何时候只要 SQL 语句被调用，都将清空本地缓存和二级缓存
useCache	启动二级缓存的开关。默认值为 true，表示将查询结果存入二级缓存中
timeout	用于设置超时参数，单位是秒。超时将抛出异常。
fetchSize	获取记录的总条数设定

statementType	告诉 MyBatis 使用哪个 JDBC 的 Statement 工作，取值为 STATEMENT（Statement）、PREPARED（PreparedStatement）、CALLABLE（CallableStatement）
resultSetType	这是针对 JDBC 的 ResultSet 接口而言，其值可设置为 FORWARD_ONLY（只允许向前访问）、SCROLL_SENSITIVE（双向滚动，但不及时更新）、SCROLL_INSENSITIVE（双向滚动，及时更新）

4、<resultMap>元素表示结果映射集，是 MyBatis 中最重要也是最强大的元素。主要用来定义映射规则、级联的更新以及定义类型转化器等。<resultMap>元素包含了一些子元素，结构如下：

```
<resultMap type="" id="">
  <constructor><!-- 类在实例化时，用来注入结果到构造方法 -->
    <idArg/><!-- ID 参数，结果为 ID -->
    <arg/><!-- 注入到构造方法的一个普通结果 -->
  </constructor>
  <id/><!-- 用于表示哪个列是主键 -->
  <result/><!-- 注入到字段或 JavaBean 属性的普通结果 -->
  <association property=""/><!-- 用于一对一关联 -->
  <collection property=""/><!-- 用于一对多、多对多关联 -->
  <discriminator javaType=""><!-- 使用结果值来决定使用哪个结果映射 -->
    <case value=""/> <!-- 基于某些值的结果映射 -->
  </discriminator>
</resultMap>
```

其中，<resultMap>元素的 type 属性表示需要的 POJO，id 属性是 resultMap 的唯一标识。子元素<constructor>用于配置构造方法（当 POJO 未定义无参数的构造方法时使用）。子元素<id>用于表示哪个列是主键。子元素<result>用于表示 POJO 和数据表普通列的映射关系。子元素<association>、<collection> 和<discriminator>在级联的情况下使用。

（三）实验内容及步骤

1、<select>元素的应用：在实际开发中，查询 SQL 语句经常需要多个参数，例如多条件查询。当传入多个参数时，MyBatis 允许 Map 接口通过键值对传递多个参数，对应 SQL 文件的代码片段如下：

```
<select id="selectAllUser" resultType="cn.edu.zjut.po.MyUser"
  parameterType="map">
  Select * from user
  where uname like concat('%',#{u_name},'%')
  and usex = #{u_sex}
```

```
</select>
```

- 2、修改测试代码，将查询所有用户信息的代码片段修改为“查询陈姓男性用户信息”，并记录运行结果，代码片段如下所示：

```
public class MyBatisTest {  
    Public static void main(String[] args){  
        try {  
            .....  
            Map<String, Object> map=new HashMap<>();  
            map.put("u_name", "陈");  
            map.put("u_sex", "男");  
            List<MyUser> list=ss.selectList  
                ("cn.edu.zjut.mapper.UserMapper.selectAllUser",map);  
            for(MyUser myUser : list){  
                System.out.println(myUser);  
            }  
            .....  
        } catch (IOException e) {  
            .....  
        }  
    }  
}
```

- 3、创建映射器 UserDao，新建名为 cn.edu.zjut.dao 的包，在该包中创建 UserDao 接口，具体代码如下：

```
Package cn.edu.zjut.dao;  
import java.util.List;  
import org.apache.ibatis.annotation.Mapper;  
import cn.edu.zjut.po.MyUser;  
  
public interface UserDao {  
    public MyUser SelectUserById(Integer id);  
    Public List<MyUser> selectAllUser(Map<String, Object> param);  
    Public int addUser(MyUser user);  
    Public int updateUser(MyUser user);  
    Public int deleteUser(MyUser user);  
}
```

- 4、修改测试代码，使用映射器 UserDao 实现“查询陈姓男性用户信息”，并记录运行结果，代码片段如下所示：

```
public class MyBatisTest {  
    Public static void main(String[] args){  
        try {  
            .....  
            Map<String, Object> map=new HashMap<>();
```

```

        map.put("u_name", "陈");
        map.put("u_sex", "男");
        UserDao userDao = ss.getMapper(UserDao.class);
        List<MyUser> list = userDao.selectAllUser(map);
        for(MyUser myUser : list){
            System.out.println(myUser);
        }
        .....
    } catch (IOException e) {
        .....
    }
}
}

```

- 5、使用 Map 不能限定其传递的数据类型，若 SQL 语句较为复杂，参数很多，也会造成 Map 使用很不方便，此时可以使用 JavaBean 传递多个参数。创建一个名为 cn.edu.zjut.pojo 的包，在包中创建一个 POJO 类 SelectUserParam，代码片段如下：

```

package cn.edu.zjut.pojo;
public class SelectUserParam {
    private String u_name;
    private String u_sex;
    //省略 setters/getters 方法
}

```

- 6、修改映射文件 UserMapper.xml 中的“查询陈姓男性用户信息”的代码，代码片段如下：

```

<select id="selectAllUser" resultType="cn.edu.zjut.po.MyUser"
        parameterType="cn.edu.zjut.pojo.selectUserParam">
    Select * from user
    where uname like concat('%',#{u_name},'%')
    and usex = #{u_sex}
</select>

```

- 7、修改测试代码中“查询陈姓男性用户信息”的部分代码，并记录运行结果，代码片段如下所示：

```

SelectUserParam su = new SelectUserParam();
su.setU_name("陈");
su.setU_sex("男");
List<MyUser> list=ss.selectList
    ("cn.edu.zjut.mapper.UserMapper.selectAllUser",su);
for(MyUser myUser : list){
    System.out.println(myUser);
}

```

- 8、同样，在任何 select 语句中都可以使用 Map 存储结果，对应 SQL 文件的代码

片段如下：

```
<select id="selectAllUser" resultType="map">
    Select * from user
</select>
```

- 9、修改测试代码，将查询所有用户信息的返回结果保存到 **Map** 中，并记录运行结果。其中，**Map** 的 **key** 是 **select** 语句查询的字段名（必须完全一致），而 **Map** 的 **value** 是查询返回结果中字段对应的值，一条记录映射到一个 **Map** 对象中。代码片段如下所示：

```
List<Map<String, Object>> lmp=ss.selectList
    ("cn.edu.zjut.mapper.UserMapper.selectAllUser");
for(MyUser myUser : list){
    System.out.println(myUser);
}
```

- 10、在 **select** 语句中也可以使用 **POJO** 存储结果，此时可以使用 **resultType** 属性进行自动映射，也可以使用 **resultMap** 属性进行更为复杂的映射或级联。此时，需要在 **cn.edu.zjut.pojo** 包中创建一个 **POJO** 类 **MapUser**，代码片段如下：

```
package cn.edu.zjut.pojo;
public class MapUser {
    private Integer m_uid;
    private String m_uname;
    private String m_usex;
    //省略 setters/getters 方法

    public String toString(){
        return "User [uid=" + m_uid + ", uname=" + m_uname +
            ", usex=" + m_usex + "];"
    }
}
```

- 11、在 SQL 映射文件 **UserMapper.xml** 中配置 **<resultMap>** 元素，其属性 **type** 引用 **POJO** 类，代码片段如下：

```
<resultMap type="cn.edu.zjut.pojo.MapUser" id="myResult">
    <!-- property 是 com.pojo.MapUser 类中的属性-->
    <!-- column 是查询结果的列名，可以来自不同的表 -->
    <id property="m_uid" column="uid"/>
    <result property="m_uname" column="uname"/>
    <result property="m_usex" column="usex"/>
</resultMap>
```

- 12、修改映射文件 **UserMapper.xml**，在 **<select>** 元素中使用 **resultMap** 属性，代码片段如下：

```
<select id="selectResultMap" resultMap="myResult">
    Select * from user
```

```
</select>
```

- 13、修改测试代码，使用 id 为 selectResultMap 的 select 操作完成查询，并记录运行结果；
- 14、<insert>、<update>和<delete>元素用于映射插入、更新和删除语句，MyBatis 执行完此类语句后，将返回一个整数表示其影响的行数。尝试使用 resultMap 属性和 parameterMap 属性，完成增删改操作，并记录运行结果。

（四）实验要求

- 1、填写并上交实验报告，报告中应包括：
 - （1）运行结果截图；
 - （2）结合实验过程，总结 MyBatis 实现查询时返回的结果集中常见的存储方式；
 - （3）碰到的问题及解决方案或思考；
 - （4）实验收获及总结。
- 2、上交程序源代码，代码中应有相关注释。

三、扩展实验——级联查询

（一）实验目的

- 1、掌握 MyBatis 关联关系映射的基本概念，理解关联的方向和数量，重点理解双向一对多/多对一的关联关系，及其在实际应用中的体现；
- 2、学习 MyBatis 框架处理一对多/多对一关联关系的方法，掌握相应 MyBatis 映射文件的配置方案；
- 3、能在实际应用中通过 MyBatis 建立正确的一对多/多对一关联关系映射，并进行数据库访问，特别是级联查询的相关实现。

（二）基本知识与原理

- 1、MyBatis 中，通常通过<resultMap>元素的子元素<association>处理级联关系。在<association>元素中，通常使用以下属性：
 - property：指定映射到实体类的对象属性。
 - column：指定表中对应的字段（即查询返回的列名）。
 - javaType：指定映射到实体对象属性的类型。
 - select：指定引入嵌套查询的子 SQL 语句，用于关联映射中的嵌套查询。

(三) 实验内容及步骤

- 1、一对一级联查询：在数据库中创建两张数据表：身份证表 **idcard**，个人信息表 **person**，创建代码如下：

```
CREATE TABLE `idcard` (  
  `id` tinyint(2) NOT NULL AUTO_INCREMENT,  
  `code` varchar(18) COLLATE utf8_unicode_ci DEFAULT NULL,  
  PRIMARY KEY (`id`)  
);  
  
CREATE TABLE `person` (  
  `id` tinyint(2) NOT NULL,  
  `name` varchar(20) COLLATE utf_unicode_ci DEFAULT NULL,  
  `age` innt(11) DEFAULT NULL,  
  `idcard_id` tinyint(2) DEFAULT NULL,  
  PRIMARY KEY (`id`),  
  KEY `idcard_id` (`idcard_id`),  
  CONSTRAINT `idcard_id` FOREIGN KEY (`idcard_id`) REFERENCES `idcard`  
  (`id`)  
);
```

- 2、创建对应的持久化类 **IDcard** 和 **Person**;
- 3、在 **MyBatis** 的核心配置文件 **mybatis-config.xml** 中打开延迟加载开关，代码片段如下：

```
<!-- 在使用 MyBatis 嵌套查询方式进行关联查询时，使用 MyBatis 的延迟加载可以在一定程度上提高查询效率 -->  
<settings>  
  <!-- 打开延迟加载的开关 -->  
  <setting name="lazyLoadingEnabled" value="true"/>  
  <!-- 将积极加载改为按需加载 -->  
  <setting name="aggressiveLazyLoading" value="false"/>  
</settings>
```

- 4、创建映射文件 **IDCardMapper.xml**，具体代码如下：

```
<?xml version="1.0" encoding="UTF-8" ?>  
<!DOCTYPE mapper  
  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  
  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  
<mapper namespace="cn.edu.zjut.dao.IDCardDao">  
  <select id="selectCodeById" parameterType="Integer"  
    resultType="cn.edu.zjut.po.IDcard">
```

```
        Select * from idcard where id=#{id}
    </select>
</mapper>
```

5、创建映射文件 **PersonMapper.xml**，并在其中以 3 中方式实现“根据 id 查询个人信息”的功能，具体代码如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper SYSTEM "http://mybatis.org/dtd/mybatis-3-mapper.dtd"
PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN">
<mapper namespace="cn.edu.zjut.dao.PersonDao">
<!-- 一对一 根据 id 查询个人信息：第一种方法（嵌套查询） -->
<resultMap id="cardAndPerson1" type="cn.edu.zjut.po.Person">
    <id column="id" property="id"/>
    <result column="name" property="name"/>
    <result column="age" property="age"/>
    <!-- 一对一关联查询 -->
    <association column="idcard_id" property="card"
        select="cn.edu.zjut.dao.IDCardDao.selectCodeById"
        javaType="cn.edu.zjut.po.IDcard"/>
</resultMap>
<select id="selectPersonById1" resultMap="cardAndPerson1"
    parameterType="Integer">
    Select * from person where id=#{id}
</select>
<!-- 一对一 根据 id 查询个人信息：第二种方法（嵌套结果） -->
<resultMap id="cardAndPerson2" type="cn.edu.zjut.po.Person">
    <id column="id" property="id"/>
    <result column="name" property="name"/>
    <result column="age" property="age"/>
    <!-- 一对一关联查询 -->
    <association property="card" javaType="cn.edu.zjut.po.IDcard">
        <id column="idcard_id" property="id"/>
        <result column="code" property="code"/>
    </association>
</resultMap>
<select id="selectPersonById2" resultMap="cardAndPerson2"
    parameterType="Integer">
    Select p.*,ic.code from person p, idcard ic
    where p.idcard_id = ic.id and p.id=#{id}
</select>
<!-- 一对一 根据 id 查询个人信息：第三种方法（使用 POJO 存储结果） -->
<select id="selectPersonById3" parameterType="Integer"
    resultType="cn.edu.zjut.pojo.SelectPersonById">
```

```
        Select p.*,ic.code from person p, idcard ic
        where p.idcard_id = ic.id and p.id=#{id}
    </select>
</mapper>
```

6、创建 POJO 类 `cn.edu.zjut.pojo.SelectPersonById`，代码片段如下：

```
package cn.edu.zjut.pojo;

public class SelectPersonById {
    private Integer id;
    private String name;
    private Integer age;
    private String code;
    //省略 setter 和 getter 方法
    @Override
    public String toString() {
        return "Person [id=" + id + ",name=" + name + ",
                age=" + age + ",code=" + code + "]";
    }
}
```

7、创建数据操作接口 `IDCardDao`，接口的具体代码如下：

```
package cn.edu.zjut.dao;

import org.apache.ibatis.annotations.Mapper;
import cn.edu.zjut.po.IDcard;

public interface IDCardDao {
    public IDcard selectCodeById(Integer i);
}
```

8、创建数据操作接口 `PersonDao`，接口的具体代码如下：

```
package cn.edu.zjut.dao;

import org.apache.ibatis.annotations.Mapper;
import cn.edu.zjut.po.Person;
import cn.edu.zjut.pojo.SelectPersonById;

public interface PersonDao {
    public Person selectPersonById1(Integer id);
    public Person selectPersonById2(Integer id);
    public SelectPersonById selectPersonById3(Integer id);
}
```

9、创建测试类 `TestOneToOne` 并记录运行结果，代码片段如下：

```
package cn.edu.zjut.controller;

public class TestOneToOne {
```

```
private PersonDao personDao;  
Person p1 = personDao.selectPersonById1(1);  
System.out.println(p1);  
System.out.println("=====");  
Person p2 = personDao.selectPersonById2(1);  
System.out.println(p2);  
System.out.println("=====");  
SelectPersonById p3 = personDao.selectPersonById3(1);  
System.out.println(p3);  
}
```

10、尝试以用户和订单之间的一对多关系为例,实现一对多级联查询的处理过程,并记录过程。

(四) 实验要求

1、填写并上交实验报告,报告中应包括:

- (1) 运行结果截图;
- (2) 碰到的问题及解决方案或思考;
- (3) 实验收获及总结。

2、上交程序源代码,代码中应有相关注释。