 ---------------------------------------------------

计划课程安排：

FUNDAMENTAL(7天)      该阶段已结束

OOP(11天)                        该阶段已结束

JAVASE01(10天)              该阶段已结束

JAVASE02(12天)               该阶段已结束

WEBBASIC(11天)        该阶段已结束

JQUERY(2天)                    该阶段已结束

DATABASE(5天)            该阶段已结束

JDBC(2天)                         该阶段已结束

SERVLETJSP(14天)        该阶段已结束

SPRINGMYBATIS01(7天) 该阶段已结束

AJAX(0.5天) 该阶段已结束

JSON(0.5天)                       该阶段已结束

SPRINGMYBATIS02(12天) （该阶段已被取消）

LINUX(2天)                        （该阶段已被取消）

NGINX(2天)                      （该阶段已被取消）

REDIS(2天)                       （该阶段已被取消）

T-STORE(5天)                             （该阶段已被取消）

预计合计：99天

预计剩余：3天 ←当前处于已毕业待就业阶段

注1：上述信息随时可能发生更改，仅供参考。

注2：被取消表示若按照正常流程就业则直接上班，不用再继续到来机构上课。

反之，则需要继续到机构上完全部课程。

---------------------------------------------------

****2019年6月24日****

****Java就业班-第95天****

**7:01实训室2-早自习**

上周六课程笔记：

### 作业

1. 创建名为`tedu\_ums`的数据库；

CREATE DATABASE tedu\_ums;

2. 在该数据库中创建名为`t\_user`的数据表，数据表中至少包括：id、用户名、密码、年龄、手机号码、邮箱这些字段；

```

CREATE TABLE t\_user (

id INT AUTO\_INCREMENT COMMENT '用户数据的id',

username VARCHAR(50) UNIQUE NOT NULL COMMENT '用户名',

password VARCHAR(50) NOT NULL COMMENT '密码',

age INT COMMENT '年龄',

phone VARCHAR(20) COMMENT '手机号码',

email VARCHAR(50) COMMENT '邮箱',

PRIMARY KEY(id)

) DEFAULT CHARSET=UTF8;

```

3. 向该表中插入不少于10条数据；

INSERT INTO t\_user (username,password,phone,age,email)

VALUES ('Henry','1234','13800138001',18,'henry@qq.com');

4. 删除指定id的数据，例如删除id=3的用户的数据；

DELETE FROM t\_user WHERE id=3;

5. 将所有用户的密码全部修改为`8888`；

UPDATE t\_user SET password='8888';

6. 将指定用户的密码修改为`123456`，例如将id=6的用户的密码改为`123456`；

UPDATE t\_user SET password='123456' WHERE id=6;

7. 统计当前有多少条用户数据；

SELECT COUNT(\*) FROM t\_user;

8. 查询指定id的用户数据，例如查询id=5的用户数据；

SELECT id,username,password,age,phone,email FROM t\_user WHERE id=5;

9. 查询用户名和密码是指定值的用户数据，例如查询用户名为`root`且密码是`1234`的用户数据；

SELECT id,username,password,age,phone,email FROM t\_user WHERE username='root' AND password='1234';

10. 删除多个指定id的数据，例如一次性删除id=4,id=2,id=8的这3条数据；

DELETE FROM t\_user WHERE id IN (2,4,8);

11. 查询年龄第2大的用户数据。

SELECT id,username,password,age,phone,email FROM t\_user ORDER BY age DESC LIMIT 1,1;

### 1. MyBatis

#### 1.1. 作用

简化持久层开发，当需要开发某个增删改查功能时，程序员只需要定义好该功能对应的抽象方法，及该抽象方法的功能对应的SQL语句即可。

#### 1.2. 创建项目

与创建SpringMVC项目的步骤相同，另外，增加添加依赖：

<!-- MyBatis -->

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis</artifactId>

<version>3.5.1</version>

</dependency>

<!-- MyBatis整合Spring -->

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis-spring</artifactId>

<version>2.0.1</version>

</dependency>

<!-- Spring-JDBC，与spring-webmvc使用相同版本 -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-jdbc</artifactId>

<version>4.3.8.RELEASE</version>

</dependency>

<!-- MySQL连接 -->

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>8.0.16</version>

</dependency>

<!-- 数据库连接池 -->

<dependency>

<groupId>commons-dbcp</groupId>

<artifactId>commons-dbcp</artifactId>

<version>1.4</version>

</dependency>

### 1.3. 配置数据库连接

在\*\*src/main/resources\*\*下创建`db.properties`文件，用于配置数据库连接：

url=jdbc:mysql://localhost:3306/tedu\_ums?useUnicode=true&characterEncoding=utf-8&serverTimezone=Asia/Shanghai

driver=com.mysql.cj.jdbc.Driver

username=root

password=root

initialSize=2

maxActive=10

关于`driver`属性的配置：在\*\*mysql-connector-java\*\*的jar包中，展开，找到\*\*META-INF\*\*下的\*\*services\*\*下的\*\*java.sql.Driver\*\*文件，该文件中记录的就是正确的驱动类，如果没有该文件，则使用`com.mysql.jdbc.Driver`。

然后，可以在Spring的配置文件中，通过`<util:properties>`节点读取以上配置文件：

<!-- 读db.properties配置 -->

<util:properties id="dbConfig"

location="classpath:db.properties" />

在实际连接时，应该使用`BasicDataSource`对象获取连接对象，在使用之前需要为`BasicDataSource`的相关属性注入值：

<!-- 配置数据源 -->

<bean class="org.apache.commons.dbcp.BasicDataSource">

<property name="url"

value="#{dbConfig.url}" />

<property name="driverClassName"

value="#{dbConfig.driver}" />

<property name="username"

value="#{dbConfig.username}" />

<property name="password"

value="#{dbConfig.password}" />

<property name="initialSize"

value="#{dbConfig.initialSize}" />

<property name="maxActive"

value="#{dbConfig.maxActive}" />

</bean>

以上配置中，各`<property>`节点的`name`属性的值来自`BasicDataSource`中的属性名称，在Spring表达式例如`#{dbConfig.driver}`中的`driver`来自\*\*db.properties\*\*文件中各等于号左侧的属性名。

完成后，就可以编写单元测试，尝试获取连接对象：

public class Tests {

@Test

public void getConnection() throws SQLException {

ClassPathXmlApplicationContext ac

= new ClassPathXmlApplicationContext(

"spring.xml");

DataSource dataSource

= ac.getBean("dataSource", DataSource.class);

Connection conn = dataSource.getConnection();

System.out.println(conn);

ac.close();

}

}

### 1.4. 接口与抽象方法

在开发持久层功能时，需要为对应的功能定义抽象方法，这些抽象方法应该存在于接口中，所以，先创建对应的接口文件`cn.tedu.mybatis.UserMapper`：

public interface UserMapper {

}

然后，在接口中添加抽象方法，关于抽象方法的设计原则：

1. 如果方法对应的功能是执行增、删、改，可以使用`Integer`作为返回值类型，表示“受影响的行数”，如果不关心返回值，也可以使用`void`，如果方法对应的功能是查询，返回值可以根据实际使用需求来设计；

2. 方法的名称可以自定义，但绝不可以重载；

3. 方法的参数按需设计。

以“增加新用户数据为例”，可以设计为：

Integer addnew(User user);

则还应该在项目补充创建`cn.tedu.mybatis.User`类。

然后，还需要在Spring的配置文件中，配置`org.mybatis.spring.mapper.MapperScannerConfigurer`类，以设置接口文件的位置：

<!-- MapperScannerConfigurer -->

<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">

<property name="basePackage"

value="cn.tedu.mybatis" />

</bean>

### 1.5. 配置SQL

从`http://doc.tedu.cn/config/mybatis-mapper.zip`下载配置SQL语句的文件的压缩包，解压后得到\*\*SomeMapper.xml\*\*文件，将其重命名为\*\*UserMapper.xml\*\*。

在\*\*src/main/resources/\*\*下创建名为\*\*mappers\*\*的文件夹，然后将\*\*UserMapper.xml\*\*粘贴到这个文件夹中。

然后编辑`UserMapper.xml`，首先，添加`<mapper>`根节点，并配置`namespace`属性，取值是该XML文件对应的JAVA接口文件：

<!-- namespace：对应的接口 -->

<mapper namespace="cn.tedu.mybatis.UserMapper">

</mapper>

然后，添加子级节点，以配置与接口中抽象方法对应的SQL语句，应该根据所需要执行的操作类型来决定使用`<insert>`、`<delete>`、`<update>`、`<select>`节点中的某一种，子点的`id`属性就是对应的抽象方法的名称，然后，在节点内部配置所需要执行的SQL语句：

<!-- id：抽象方法的名称 -->

<insert id="addnew">

INSERT INTO t\_user (

username,password,

age,phone,

email

) VALUES (

#{username},#{password},

#{age},#{phone},

#{email}

)

</insert>

如果配置的是`<select>`节点，还必须配置`resultType`属性，用于表示返回值类型，如果返回值类型是`List`集合，则该属性配置值是`List`集合中的元素类型。

最后，还需要在Spring的配置文件中，配置`org.mybatis.spring.SqlSessionFactoryBean`类，用于指定：XML文件的位置，指定数据源：

<!-- SqlSessionFactoryBean -->

<bean class="org.mybatis.spring.SqlSessionFactoryBean">

<!-- XML文件在哪里 -->

<property name="mapperLocations"

value="classpath:mappers/\*.xml" />

<!-- 用哪个数据源 -->

<property name="dataSource"

ref="dataSource" />

</bean>

验证以上功能是否完成，可以编写并执行单元测试：

@Test

public void addnew() {

ClassPathXmlApplicationContext ac

= new ClassPathXmlApplicationContext(

"spring.xml");

UserMapper userMapper

= ac.getBean("userMapper", UserMapper.class);

User user = new User();

user.setUsername("MyBatis");

user.setPassword("888888");

Integer rows = userMapper.addnew(user);

System.out.println("rows=" + rows);

ac.close();

}

又是一晚没睡，第一个到达教室。我似乎是又生病了。

不行，顶不住，请假回宿舍休息。

**23:30宿舍-休息**

这一觉睡足了，开始通宵搞学习吧。

-----------------------------------------------------------------

****2019年6月25日****

****Java就业班-第96天****

**8:08实训室2-早自习**

正常到教室，今天还是正常上课吧。虽说今天也是通宵了，但精神状态似乎还可以。说出来我自己的都不信，我居然真的通宵复习了一晚上……最后卡在了动态代理的内容上，那么现在接着研究吧。

先把昨天没上课的课程笔记补一下：

### 1. 查询时，需要查询结果中的列名与返回值类型中的属性名保持一致

假设在`t\_user`表中添加了新的名为`is\_delete`的字段：

alter table t\_user add column is\_delete int;

则对应的`User`类中也应该添加新的属性，以与对应：

public class User {

private Integer id;

private String username;

private String password;

private Integer age;

private String phone;

private String email;

private Integer isDelete;

当查询数据时，需要自定义别名，使得查询结果中的列名与返回值类型中的属性名保持一致：

<select id="findById"

resultType="cn.tedu.mybatis.User">

SELECT

id,username,

password,age,

phone,email,

is\_delete AS isDelete

FROM

t\_user

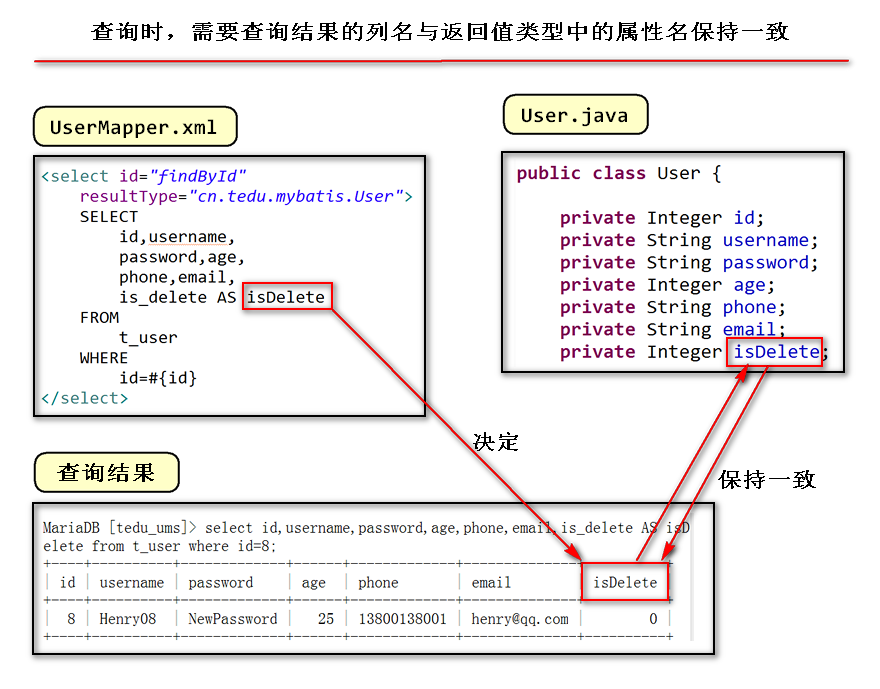
WHERE

id=#{id}

</select>

其对应关系如图所示：

![](01.png)



### 2. 简单的关联表查询数据

假设需要创建“部门信息表”：

CREATE TABLE t\_department (

id INT AUTO\_INCREMENT COMMENT '部门id',

name VARCHAR(30) UNIQUE NOT NULL COMMENT '部门名称',

PRIMARY KEY(id)

) DEFAULT CHARSET=utf8;

然后，添加一些模拟数据：

INSERT INTO t\_department (name) VALUES ('软件研发部'),('人力资源部'),('财务部'),('销售部');

并且，每个用户都归属于某个部门：

ALTER TABLE t\_user ADD COLUMN department\_id INT;

最后，为用户分配部门：

UPDATE t\_user SET department\_id=1 WHERE id IN (21,28,32);

UPDATE t\_user SET department\_id=2 WHERE id IN (24,25,26);

UPDATE t\_user SET department\_id=3 WHERE id IN (22,27,30);

UPDATE t\_user SET department\_id=4 WHERE id IN (23,29,31);

假设存在需求“查询某用户的信息，并显示该用户的部门的名称”，必须通过关联查询才可以得到所需要的结果，需要执行的SQL语句大致是：

select

\*

from

t\_user

left join

t\_department

on

t\_user.department\_id=t\_department.id

where

t\_user.id=25;

如果需要使用MyBatis开发该功能，首先，目前并没有某个类型可以封装查询结果！因为创建的`User`类是与`t\_user`表相对应的，例如`User`这样的类称之为\*\*实体类(entity)\*\*，实体类肯定不满足多表查询需求的，所以，对于这种情况，需要自行另创建\*\*VO类(value object)\*\*，这种类的设计应该与查询结果或查询需求相对应：

public class UserVO {

private Integer id;

private String username;

private String password;

private Integer age;

private String phone;

private String email;

private Integer isDelete;

private Integer departmentId;

private String departmentName;

}

> VO类的设计方式与实体类基本一致，只是这2种类的定位不同而已，实体类是与数据表对应的，VO类是与查询结果对应的。

然后，可以将此前的`findById(Integer id)`的返回值类型修改为`UserVO`：

UserVO findById(Integer id);

且映射的配置：

<select id="findById"

resultType="cn.tedu.mybatis.UserVO">

SELECT

t\_user.id,username,

password,age,

phone,email,

is\_delete AS isDelete,

department\_id AS departmentId,

name AS departmentName

FROM

t\_user

LEFT JOIN

t\_department

ON

t\_user.department\_id=t\_department.id

WHERE

t\_user.id=#{id}

</select>

### 3. 使用<resultMap>处理1对多的关联查询

假设存在需求“查询某部门信息，且显示出该部门所有的员工”，需要执行的SQL语句大致是：

select

\*

from

t\_department

left join

t\_user

on

t\_department.id=t\_user.department\_id

where

t\_department.id=?

这类查询可能查到多条结果，取决于该部门的员工数量，但是，查询的需求却是查询“某1个”部门的信息！在设计查询的方法时，就需要某1个类型可以封装多条查询结果，可以：

public class DepartmentVO {

private Integer id;

private String name;

private List<User> users;

}

则对应的抽象方法可以是：

public interface DepartmentMapper {

DepartmentVO findById(Integer id);

}

此次查询必然可能出现多个结果，MyBatis并不知道如何将多个结果封装到1个对象中，所以，按照此前的查询做法，必然会出错！例如：

Caused by: org.apache.ibatis.exceptions.TooManyResultsException: Expected one result (or null) to be returned by selectOne(), but found: 3

在这种情况下，需要自定义`<resultMap>`节点，用于指导MyBatis如何将结果进行封装：

<!-- id：自定义名称 -->

<!-- type：返回结果的类型 -->

<resultMap id="DepartmentVO\_Map"

type="cn.tedu.mybatis.DepartmentVO">

<!-- id节点：用于配置主键 -->

<!-- result节点：用于配置非主键 -->

<!-- column：查询结果的列名 -->

<!-- property：返回结果类型中的属性名 -->

<!-- 无论哪个节点，都是用于告之MyBatis将查询结果中哪一列的数据放到返回类型中的哪个属性中 -->

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

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

<!-- collection节点：用于配置1对多关系 -->

<!-- ofType：集合中的元素类型 -->

<collection property="users"

ofType="cn.tedu.mybatis.User">

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

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

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

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

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

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

<result column="is\_delete" property="isDelete"/>

</collection>

</resultMap>

<select id="findById"

resultMap="DepartmentVO\_Map">

SELECT

t\_department.id, name,

t\_user.id AS uid, username,

password, age,

phone, email,

is\_delete

FROM

t\_department

LEFT JOIN

t\_user

ON

t\_department.id=t\_user.department\_id

WHERE

t\_department.id=#{id}

</select>

查询结果例如：

DepartmentVO [

id=1, name=软件研发部,

users=[

User [id=21, username=Mike01, password=8888, age=18, phone=13800138001, email=Mike@qq.com, isDelete=0],

User [id=28, username=Mike08, password=8888, age=25, phone=13800138001, email=Mike@qq.com, isDelete=0],

User [id=32, username=Mike11, password=8888, age=26, phone=13800138001, email=Mike@qq.com, isDelete=0]

]

]

### 4. 动态SQL

动态SQL指的是可以在配置SQL语句添加一些特殊的标签，例如`<if>`、`<foreach>`等，可以根据参数的不同，最终生成不同的SQL语句，则称之为动态SQL。

例如：根据若干个id删除数据，大致的SQL语句是：

delete from t\_user where id in (1,3,5,7,9)

在实际应用中，以上`IN`关键字后侧的括号中的值是不确定的，不光是值本身，值的数量也是不确定，并且各值之间需要使用逗号进行分隔，当值的数量不确定时，逗号的数量也是无法确定的！

首先，需要在`UserMapper.java`接口中添加抽象方法：

Integer deleteByIds(List<Integer> ids);

在设计参数时，可以使用`List`集合类型，也可以使用数组类型。

然后，在`UserMapper.xml`中配置以上抽象方法的映射：

<delete id="deleteByIds">

DELETE FROM t\_user

WHERE id IN (

<foreach collection="list"

item="id" separator=",">

#{id}

</foreach>

)

</delete>

在配置`<foreach>`节点时：

- `collection`：需要被遍历的集合或数据，如果抽象方法只有1个参数时，如果参数的类型是`List`集合，则取值为`list`，如果参数类型是数组，则取值为`array`；如果抽象方法有多个参数，则该属性取值为`@Param("xx")`注解中使用的名称。

- `item`：遍历过程中，集合中的元素的名称，在`<foreach>`子级位置，可以使用`#{item值}`表示被遍历到的元素的值。

- `separator`：分隔符。

- `open`和`close`：遍历生成的SQL语句部分的最左侧字符和最右侧字符。

练习1：根据若干个id查询用户列表，例如查询id=26和id=29和id=30的用户的数据：

List<User> findByIds(Integer[] ids);

<select id="findByIds"

resultType="cn.tedu.mybatis.User">

SELECT

id, username,

password, age,

phone, email,

is\_delete AS isDelete

FROM

t\_user

WHERE

id IN

<foreach collection="array"

item="id" separator=","

open="(" close=")">

#{id}

</foreach>

</select>

练习2：将多个用户的密码改成某个值，例如将id=25、id=27、id=28的用户的密码都改成`123456`，该功能中，id值是由用户指定的，新密码也是由用户指定的：

Integer updatePasswordByIds(

@Param("ids") Integer[] ids,

@Param("password") String password);

<update id="updatePasswordByIds">

UPDATE

t\_user

SET

password=#{password}

WHERE

id IN

<foreach collection="ids"

item="id" separator=","

open="(" close=")">

#{id}

</foreach>

</update>

### 5. #{}与${}占位符

在MyBatis中，配置SQL语句时，可以使用`#{}`与`${}`这2种占位符。

使用`#{}`占位符，可以用于占位某些值，也就是在SQL中写值的位置，都可以使用这种占位符(此前在学习JDBC时使用?的位置)；而`${}`可以表示SQL语句的任何部分！

在使用`#{}`对某个值进行占位时，框架对整个SQL语句是有预编译处理的，无需考虑该值的数据类型的问题；而使用`${}`占位时，框架的处理方式其实就是非常单纯的字符串拼接，需要考虑数据类型的问题，如果占位的值中包括字符串类型的值，则必须使用`''`框住值！

由于`#{}`只能对某个值进行占位，SQL语句本身是相对固定的，所以，这种做法实现的功能的局限性就非常明显，由于是预编译的，没有SQL注入风险，且工作效率较高！而`${}`可以随意占位，功能可以非常灵活，但是，不是预编译的，有SQL注入风险，工作效率较低。

**8:38实训室2-早自习**

今天晨讲的学员没来，应该是迟到了，所以改为由项目经理进行讲解什么是权限管理。

**9:13实训室2-正课**

上午的课程内容是JSON。而下午则是AJAX，似乎是打算只用今天一天的时间讲解完这两项内容。

**17:50实训室2-下课时间**

今日课程笔记：

### 1. @ResponseBody注解

可以在处理请求的方法之前添加`@ResponseBody`注解，表示该方法返回的内容会直接作用响应正文给到客户端，而并不再执行转发或重定向！

目前，更推荐使用这样的方式进行响应，而不再是转发或重定向，因为这样做的好处在于：服务器端不必再关心客户端的显示问题，可以更好的实现服务器端与客户端的分离，并且，这种方式响应时，响应的正文长度更短，则响应速度更快，产生的流量消耗更小，用户体验更好！

### 2. 关于响应正文的数据格式

客户端提交请求时，也许希望从服务器端得到一些数据，例如：用户打开“个人中心”时则需要从服务器获取当前登录的用户的详情，用户打开“收藏列表”时则需要从服务器获取此前收藏的相关内容的信息，所以，服务器需要向客户端响应数据，那这些数据必须是有规则的组织起来的！

假设需要响应数据有用户名、年龄、手机号码、邮箱，首先，决不可能响应为：

root2513800138001root@163.com

这些数据必须能够分开，并且保证客户端通过某种技术可以只获取其中的某个部分，所以，就需要组织数据的规则，早期的解决方案是使用XML组织这些数据：

<user>

<username>root</username>

<age>25</age>

<phone>13800138001</phone>

<email>root@163.com</email>

</user>

目前比较流行的组织数据的方式是JSON格式：

{

"username":"root",

"age":25,

"phone":"13800138001",

"email":"root@163.com"

}

### 3. JSONO类型

JSON数据中，整个数据是一个\*\*JSON对象\*\*，对象中可以有多组\*\*属性与值\*\*的配置，各组之间使用逗号(`,`)进行分隔，而\*\*属性名称\*\*与\*\*属性值\*\*之间使用冒号(`:`)进行分隔，\*\*属性名称\*\*是字符串类型的数据，需使用一对引号框住，\*\*属性值\*\*可以是任意数据类型，如果是数值型或布尔型，可以直接写，不需要使用引号，如果是字符串类型，则需要使用引号框住。

JSON数据是在JavaScript语言中可以直接识别的数据类型，无需使用任何解析技术：

<script type="text/javascript">

var data = {

"username":"root",

"age":25,

"phone":"13800138001",

"email":"root@163.com"

};

console.log("username=" + data.username);

console.log("age=" + data.age);

console.log("phone=" + data.phone);

console.log("email=" + data.email);

</script>

在JSON数据中，每个属性的值还可以是另一个JSON对象，例如：

{

"username":"root",

"age":25,

"phone":"13800138001",

"email":"root@163.com",

"department":{

"id":3,

"name":"RD"

}

}

以上`department`属性的值就是一个JSON对象，该对象中有`id`和`name`这2个属性，在对象内部组织属性与值的方式与此前完全相同！

另外，在JSON中也有\*\*数组\*\*的概念，每个属性的值都可以是一个数组，例如：

{

"username":"root",

"age":25,

"phone":"13800138001",

"email":"root@163.com",

"department":{

"id":3,

"name":"RD"

},

"skill":["Java","MySQL","SSM"]

}

可以通过下标获取数组元素的值：

console.log("skill-1=" + data.skill[0]);

console.log("skill-2=" + data.skill[1]);

console.log("skill-3=" + data.skill[2]);

在JSON的数组其实也是相对于JavaScript这门语言的数组，所以，某个属性是数组类型，对于JavaScript语言来说，就是一个数组，也是有`length`属性的，所以，在操作数组类型的数据时，也可以进行循环或遍历：

for (var i = 0; i < data.skill.length; i++) {

console.log("skill[" + i + "]=" + data.skill[i]);

}

另外，数组的各元素的类型也可以是JSON对象，在JSON中还可以有数组……

在实际使用时，JSON数据往往是来自服务器端的响应的，但是，服务器响应的数据传递到客户端时，一定是字符串类型的，则无法直接在JavaScript访问相关属性，可以通过`JSON.parse(str)`将字符串转换为JSON对象，例如：

var str = '{'

+ '"username":"root",'

+ '"age":25,'

+ '"phone":"13800138001",'

+ '"email":"root@163.com",'

+ '"department":{'

+ ' "id":3,'

+ ' "name":"RD"'

+ '},'

+ '"skill":["Java","MySQL","SSM"]'

+ '}';

var data = JSON.parse(str);

综上，可以看到，相对XML语法而言，JSON的优势有：简洁，使用的字符数量相对较少，传输速度较快，用户体验较好，解析简单，不需要使用额外的解析技术。

### 4. 服务器端向客户端响应JSON格式的数据

当服务器处理请求的方法添加了`@ResponseBody`注解后，表示响应正文，且返回值类型是`String`时，会将方法返回值中的字符串响应给客户端，但是，默认的响应头中的配置是：

Content-Type: text/html;charset=ISO-8859-1

所以，默认的响应方式是不支持中文的！

总的来看，自行组织JSON数据格式来进行响应是不可取的！

正确的做法是自定义某个类型，用于表示所需要响应的数据内容：

public class JsonResult {

private Integer state;

}

当创建以上类型的对象，为`state`属性赋值为`0`，则表示将响应的JSON数据为：

{"state":0}

也可以在以上类中添加更多的属性：

public class JsonResult {

private Integer state;

private String message;

}

当创建对象并为属性赋值后，表示将响应的JSON数据例如：

{"state":0,"message":"xxx"}

在实际应用中，也应该创建这样的类，用于统一约定响应结果的属性及格式。

具体的设计通常是：

public class JsonResult<T> {

// 操作状态，即成功或失败

private Integer state;

// 操作失败时的提示消息

private String message;

// 响应给客户端的数据

private T data;

}

使用这种响应结果时，必须添加Jackson依赖：

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-databind</artifactId>

<version>2.9.8</version>

</dependency>

在使用之前，还应该在Spring的配置文件中配置注解驱动：

<!-- 注解驱动 -->

<mvc:annotation-driven />

该框架的作用就是将返回的对象组织成JSON格式，并将响应头中的`Content-Type`进行设置：

Content-Type: application/json;charset=UTF-8

在应用时，将控制器中处理请求的方法的返回值设置为`JsonResult`类型，并返回该类型的对象即可，最终，客户端就会收到与返回对象相匹配的JSON字符串，例如：

{"state":1,"message":"注册成功，用户名＝小李同学","data":null}

之所以将Jackson依赖添加到项目中就可以直接使用，是因为：当响应正文时，SpringMVC框架会根据匹配的转换器(Converter)将方法的返回值转换为响应的正文，并决定响应头的相关信息，在SpringMVC使用Converter时有一定的优先级，例如当方法的返回值是`String`时，优先级是较高的，会自动使用`StringHttpMessageConverter`转换器，当添加了Jackson依赖时，凡是返回值类型是SpringMVC默认不支持的，就会自动应用Jackson中的转换器，而Jackson中的转换器主要就设置了响应头中的`Content-Type`及将返回的对象转换为JSON格式。

### 5. AJAX

Ajax即：Asynchronous JavaScript And XML（异步JavaScript和XML）。

在JavaScript中，通过AJAX可以发出异步请求，即由子线程发出请求，且将由子线程获取响应结果，则主线程可以不参与，表现为浏览器中显示的页面内容可以不发生变化。

基于jQuery的AJAX访问，可以调用jQuery中的`$.ajax()`函数，该函数的参数是一个JSON对象，需要至少为该JSON对象配置5个属性：

<script type="text/javascript">

$("#btn-reg").click(function(){

// url：将请求提交到哪里去

// data：请求参数

// type：请求方式，例如GET或POST等

// dataType：服务器端响应的数据类型，取值可以是json、text、xml等，取决于服务器端响应时的响应头中的Content-Type

// success：响应成功时(Http响应码是200)的回调(callback)函数，函数中的参数就是服务器端响应的JSON数据形成的JSON对象

$.ajax({

"url":"user/handle\_register.do",

"data":"username=Hello&password=123456",

"type":"post",

"dataType":"json",

"success":function(obj) {

if (obj.state == 0) {

alert(obj.message);

} else {

alert("注册成功！");

}

}

});

});

</script>

**20:59实训室2-晚自习**

只用了今天一天的时间，讲师就把JSON和AJAX部分的内容全部讲完了，明天开始就是利用所学的知识开始尝试编写实例项目。JSON和AJAX单独拿出来讲其实内容也确实不多，不过也得看是讲到个什么程度了，我目前是只知道其分别是干什么用的，具体怎么写还真的不怎么熟悉。

项目经理也强调道，自今天起，机构再没有新的知识内容教学了，而后天的招聘会将是近期最后一次招聘会，前来招人的企业要求也比上周五的要求低很多，争取全员应聘上，全员尽早毕业。（如果可能的话吧。）

P·S:计划表有所更新。

今天我的主要目的就是弄明白的AOP是什么，为此从今天凌晨开始一步一步的复习、学习，从最基本的反射→注解→代理模式→静态代理→动态代理。争取在今天剩下的时间之内彻底理解动态代理的工作原理与其作用。

**23:58宿舍-休息**

回到宿舍，今天也算是结束了。很抱歉昨天我因为身体不适（周日一晚没睡+感冒），临时请假鸽了一天的日记。这几天压力还是有的，精力集中起来也比之前要容易了许多，主要还是因为感受到了自身基础水平严重下滑的原因吧。

JDK的动态代理实现我已经整明白了，并且对CGLIB的动态代理也有了一定的了解。由于我还不能很严谨的将过程原理口述出来，所以这里就暂时摘抄面试宝典里面的一段内容来表示我已经理解到的情况吧：

##动静态代理的区别，什么场景使用？

--静态代理通常只代理一个类，动态代理是代理一个接口下的多个实现类。

--静态代理事先知道要代理的是什么，而动态代理不知道要代理什么东西，只有在运行时才知道。

--动态代理是实现 JDK 里的 InvocationHandler 接口的 invoke 方法，但注意的是代理的是接口，也就是你的

业务类必须要实现接口，通过 Proxy 里的 newProxyInstance 得到代理对象。

--还有一种动态代理 CGLIB，代理的是类，不需要业务类继承接口，通过派生的子类来实现代理。通过在运行

时，动态修改字节码达到修改类的目的。

--AOP 编程就是基于动态代理实现的，比如著名的 Spring 框架、Hibernate 框架等等都是动态代理的使用例子。

我自认为攻克了这一难点会更方便我学习并理解Spring框架的工作流程及使用方法，所以应该算是有些进步了吧。哪怕是只有那么一点点……

再说一下就业方面的事情，班主任这两天有时不时在群里发一些企业的招聘需求，如果感觉自己合适的话，班主任会把有意向的学员的简历直接发送到对方公司负责的HR手上。所以这一点比较好的地方就是与个人海投不一样，不会被筛选条件直接屏蔽掉，基本上保障了学员的简历能够得到有效的投递。

项目经理方面则是在班上反复一件事情，那就是找准自己的目标，因为有部分学员觉得自己的水平不足，还不足以胜任实际工作，那么这些天就得加紧学习。这么说是因为想告诉学员，技术永远是衡量程序员各方面的唯一标准，作为机构的员工顺带澄清一件事就是说，并不是强制要求各位学员尽早就业，只不过是因为近期会有各种招聘消息以及招聘会的举行，希望每个学员能够明确自己的立场。如果想要尽快就业，那就抓紧这些机会以及利用上所有可用的时间加紧复习。如果想要继续学习，那么就保持自我不要被当前的环境所影响。

…………

不得不说，到了现在这个情况，每个学员也都开始真正努力了，而我也不再是班上的四大逼王……因为班上厉害的大佬也都开始浮出水面了。

还有明天一天复习的时间，后天的招聘会我必须得找到工作才行……

今日感想总结：今天我算是体会到了一把什么叫做废寝忘食，今天一整天就只吃了一段饭，其他时间都是在搞学习。顺便一提刚才点的外卖真特么难吃……

今日评分：9/10分

今天就到这里，明天继续投稿。

感谢支持！

================================================================

****Acer程序员群：790482850****

****up主的java学习日记word文件及全部代码（随日记同步更新）：****

Address：github.com/wt62635