《SSM 框架》实训指导书

院系名称：经管信息学院

课程代码：31091220

总学时数：80

适用专业：大数据技术与应用专业

编 制 人：孙 检

编制日期：2020 年 7 月

审 核 人：

审 定 人：

一、项目目的与要求

《SSM 框架》主要目的是让学生通过这门实践技能课程的学习了解和掌握 spring+springMVC+Mybatis 的基本方法，通过不断上机项目练习达到解决实际的问题。因此，在本学期特设置此课程设计，集中一段时间使学生综合运用所学习的　　知识及以前所学习的计算机方面的知识，按照企业流程，完成一个相对具体、综合的网站，全面巩固学生的知识，培养学生解决实际问题的能力，从而达到学以致用的目的。

二、项目内容

（一）实例项目

以电 SSM 设计的实例指导学生如何独立完成设计和制作。让学生在机房实际操作，按照给定的实例完成实例中站点的创建和设计制作。

（二）自建站点项目

让学生自行选择站点的主题，从规划站点到上传文件一步一步完成整个项目的创建、调试和上传工作。

（三）总结

对学生的全部作品进行考核，并选择典型的案例对项目的结果进行考核。

二、参考课时

标题	项目内容	项目课时
预备	项目内容概述	3
项目一	基本知识回顾	3
项目四	Spring 操作	3
项目五	Mybatis 操作	9
项目六	Springmvc 操作	9
项目七	Ssm 合成	16
总计		60

三、项目材料准备

（一）软件准备

Eclipse 3.6 或以上版本，jdk1.8 或以上版本。

（二）硬件准备

教师用机：Windows 7 及以上版本。

学生用机：Windows 7 及以上版本。

四、综合项目考核办法：

系统文档	20 分
编写代码	30 分
程序调试	10 分
项目出勤	20 分
技术含量	10 分
美工设计	10 分

目　　录

项目一 基本知识回顾

一、 项目目的和要求

温习 的课程重点难点，使学生对 各方面的操作知识系统的由“片”的认识转向“面”的认识。

二、 项目内容

的基本操作：创建项目，类，执行等。

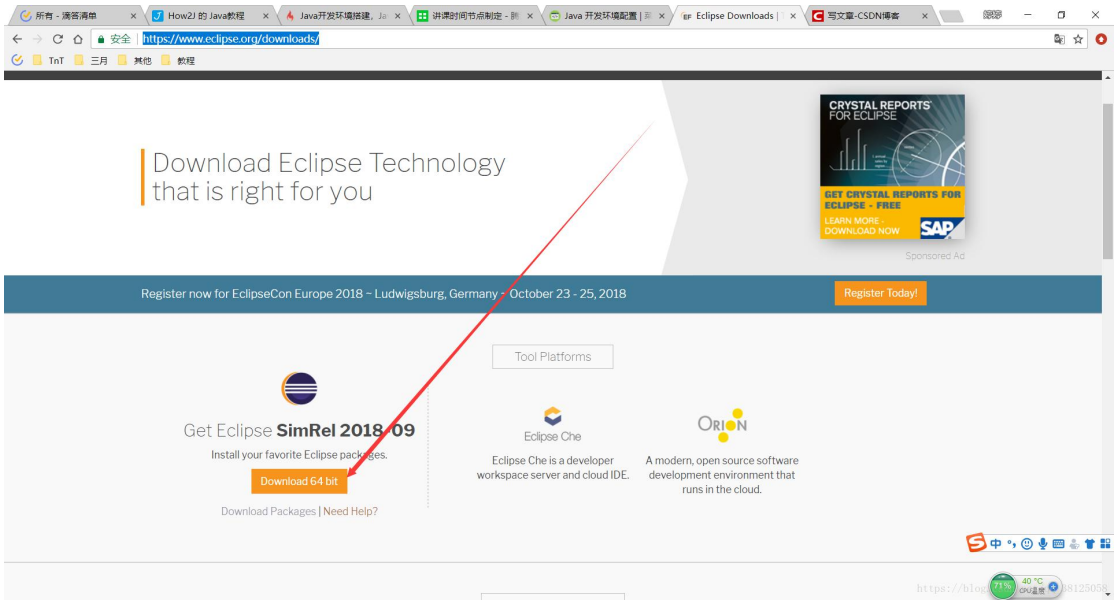
三、 项目准备

中文版。

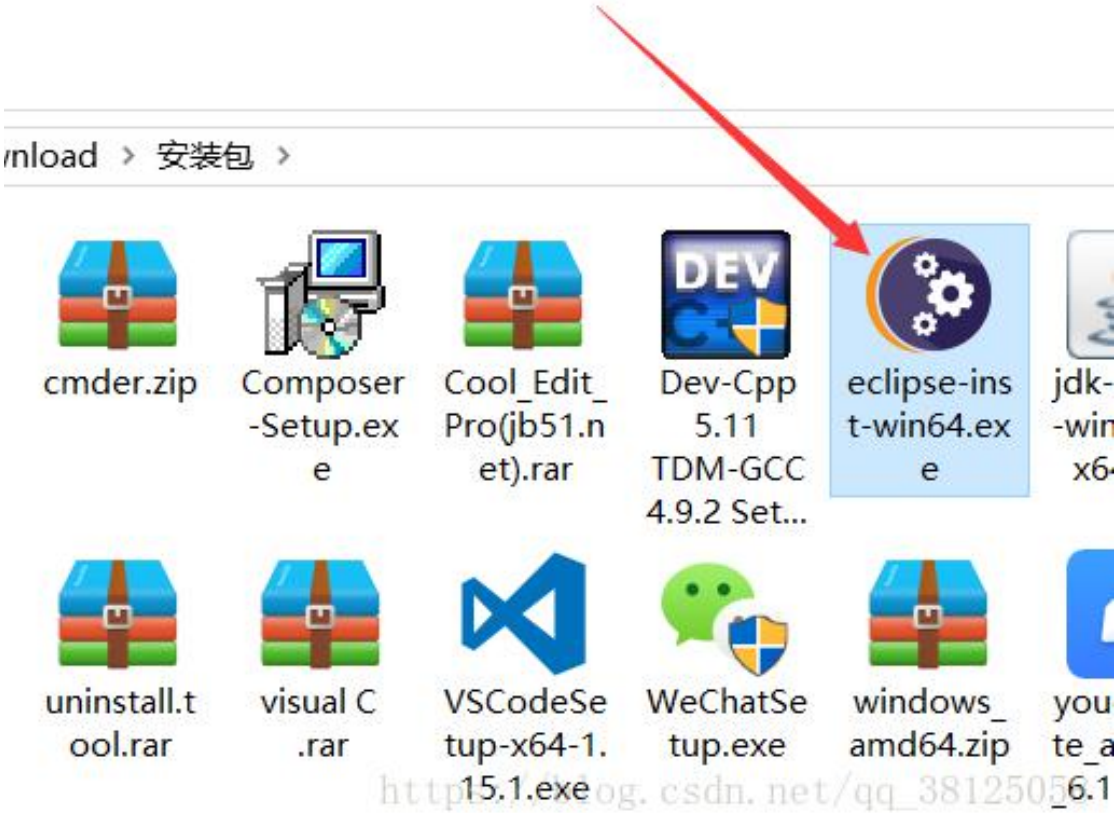
四、 项目步骤

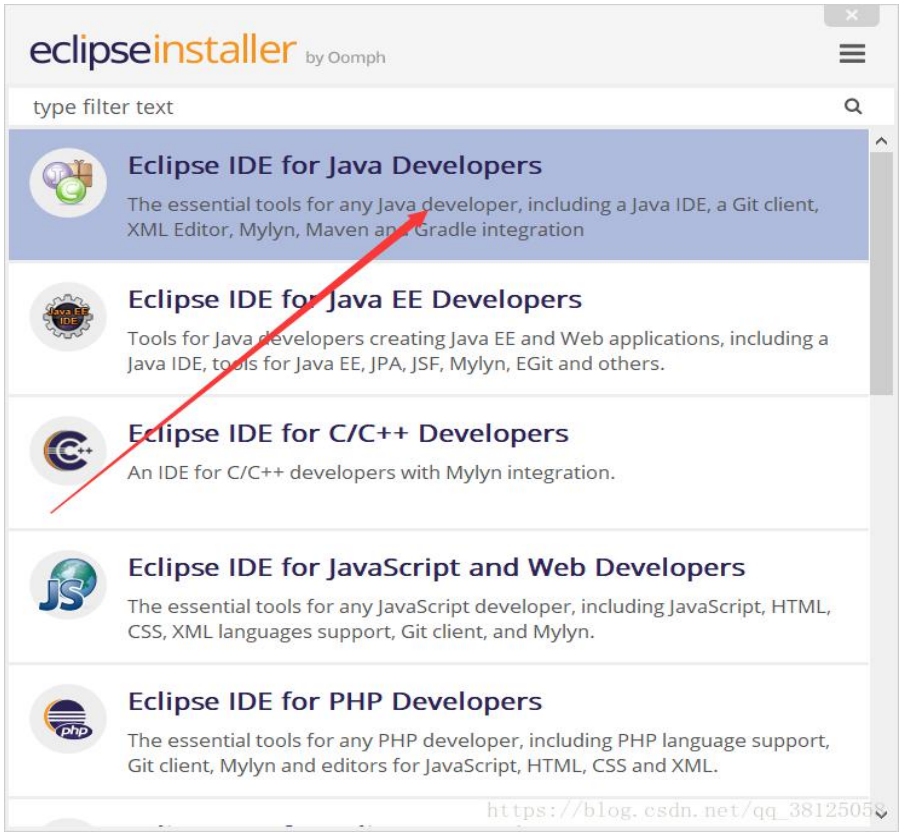
各项目指导教师按照所代学生的情况不同选择性地按下列步骤温习 eclipse 的重点难点知识。

下载地址：https://www.eclipse.org/downloads/

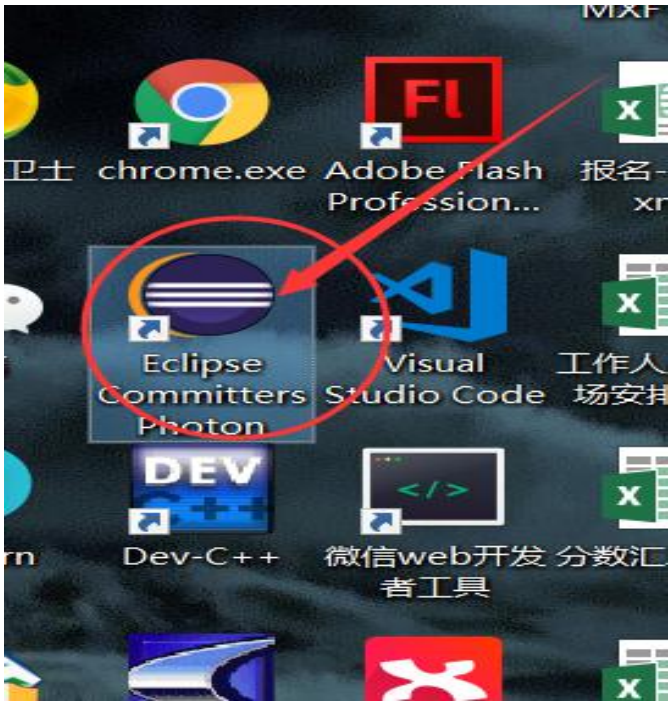


下载完成之后，双击运行，下载安装。

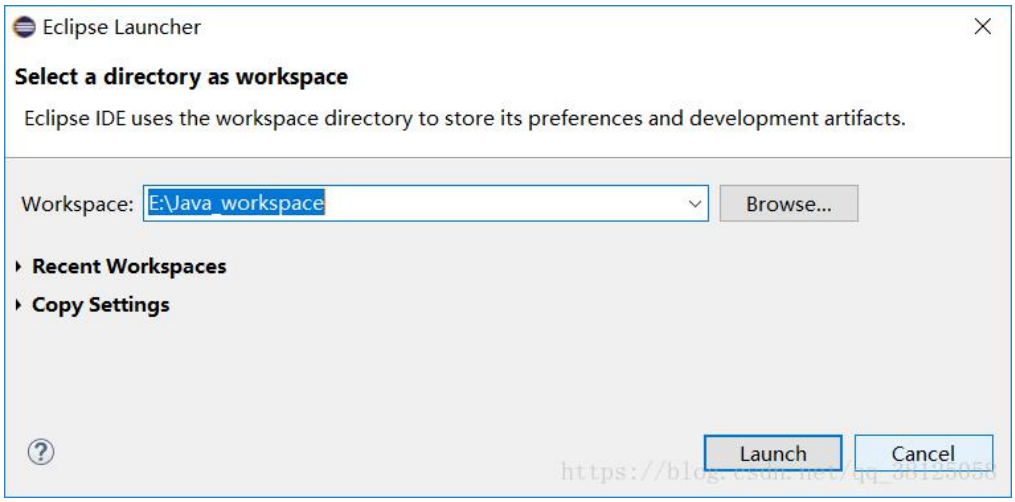


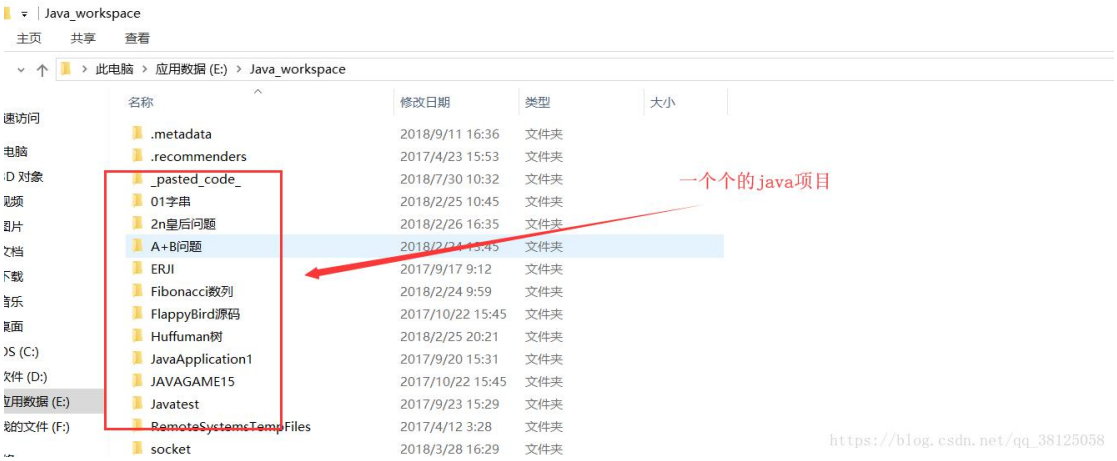


安装完成之后，桌面上就会多了这么一个图标，双击运行



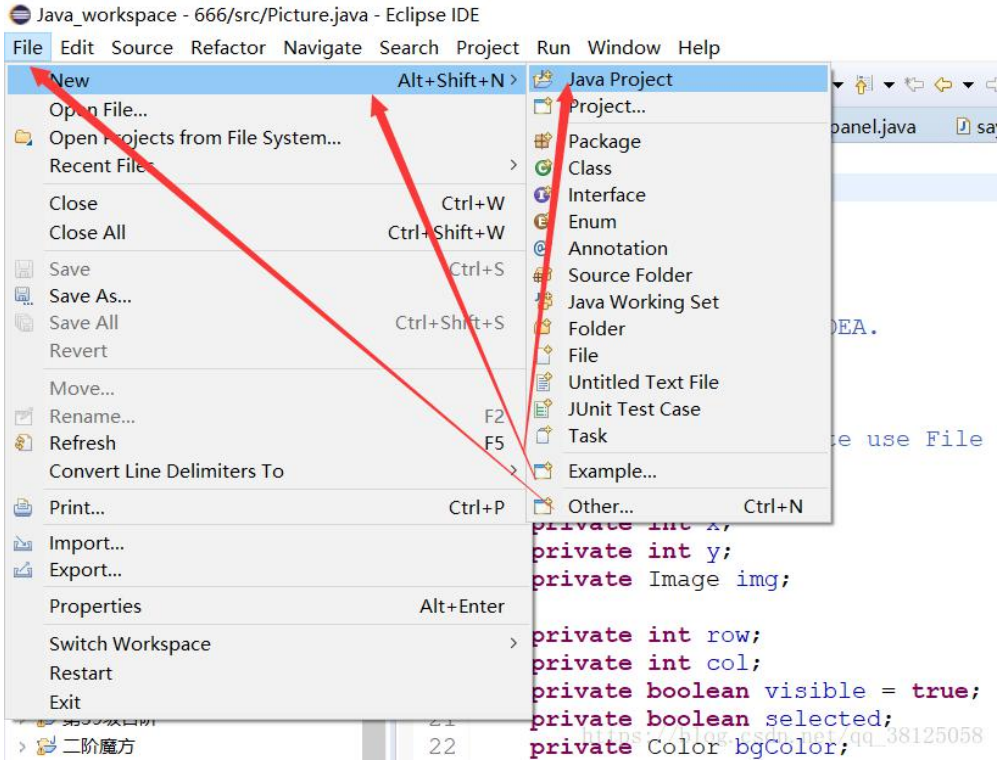
记得设置自己的工作空间，以后自己所有写的代码都会存放到这里。

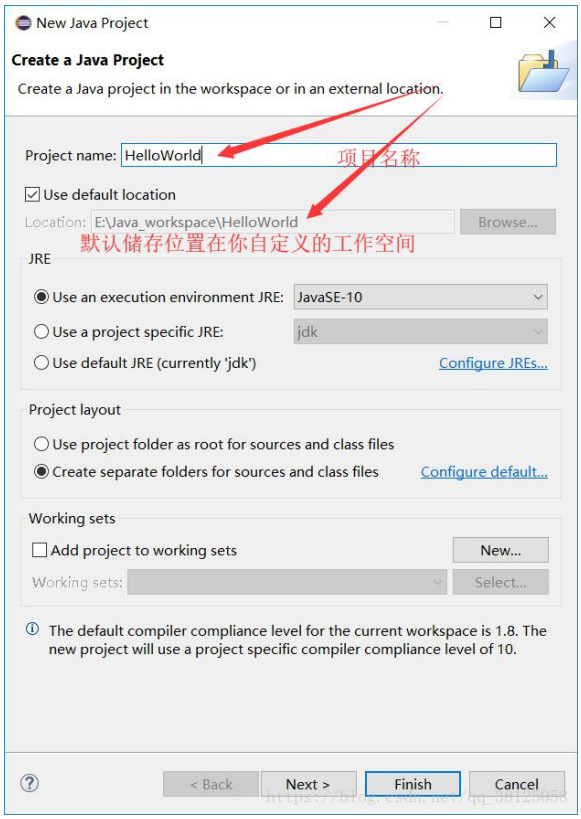


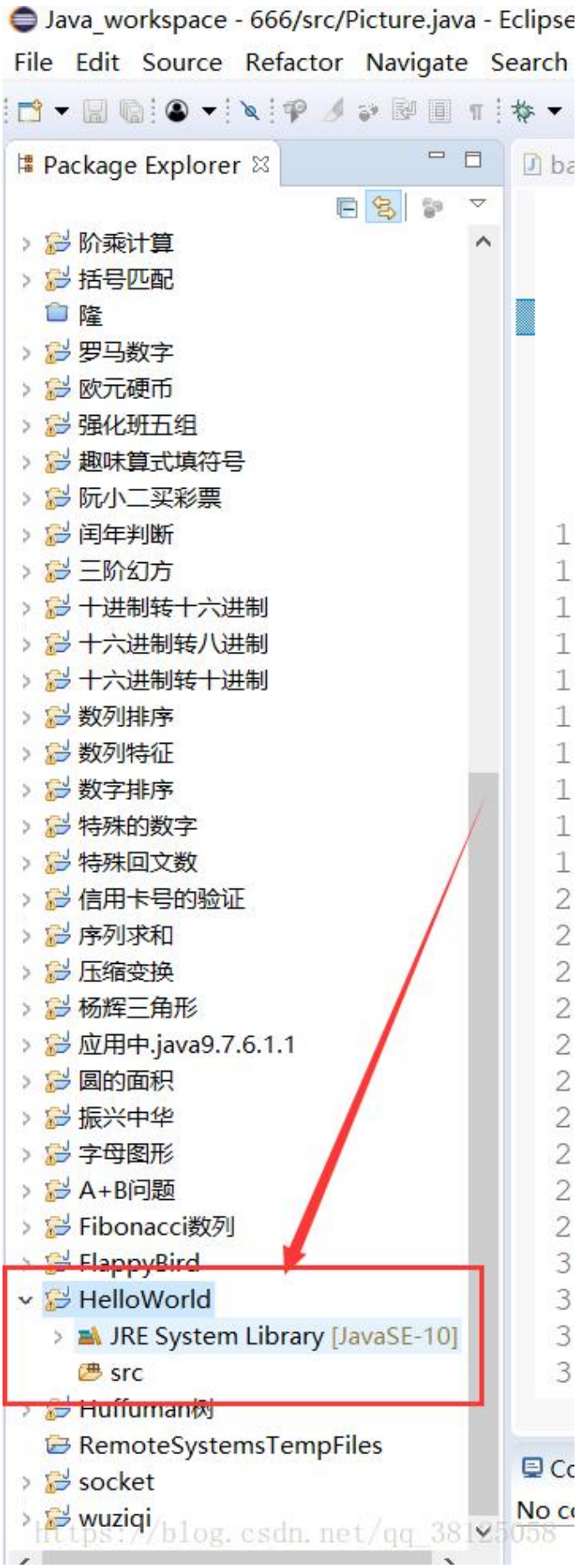


使用：

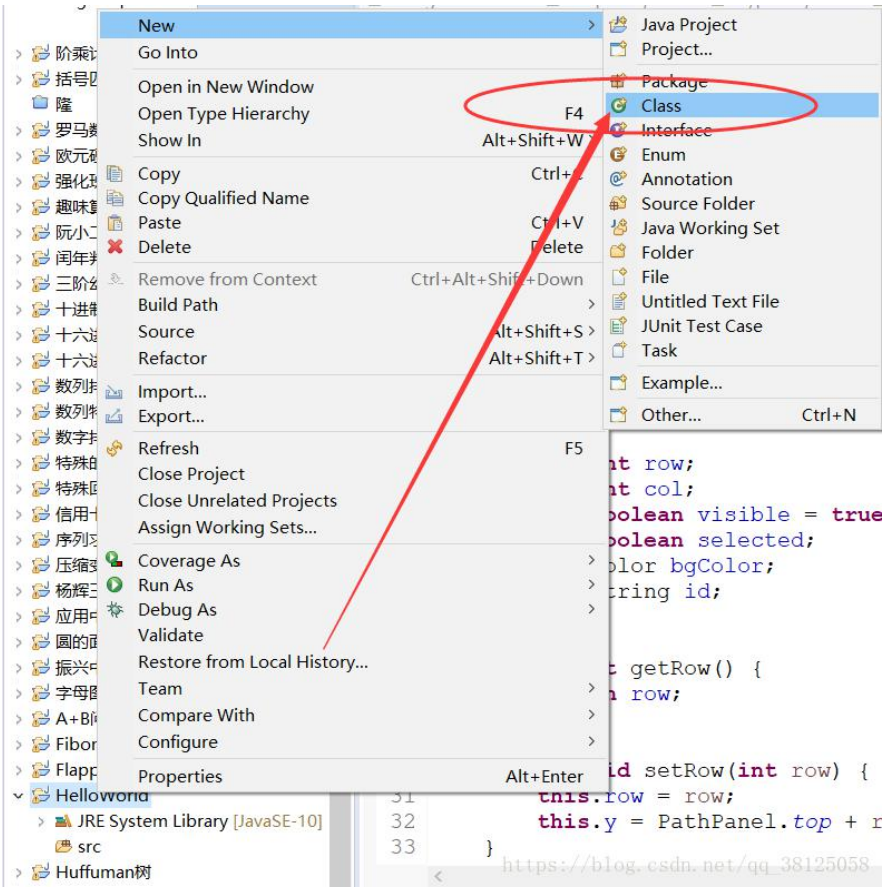
一、新建 java 项目

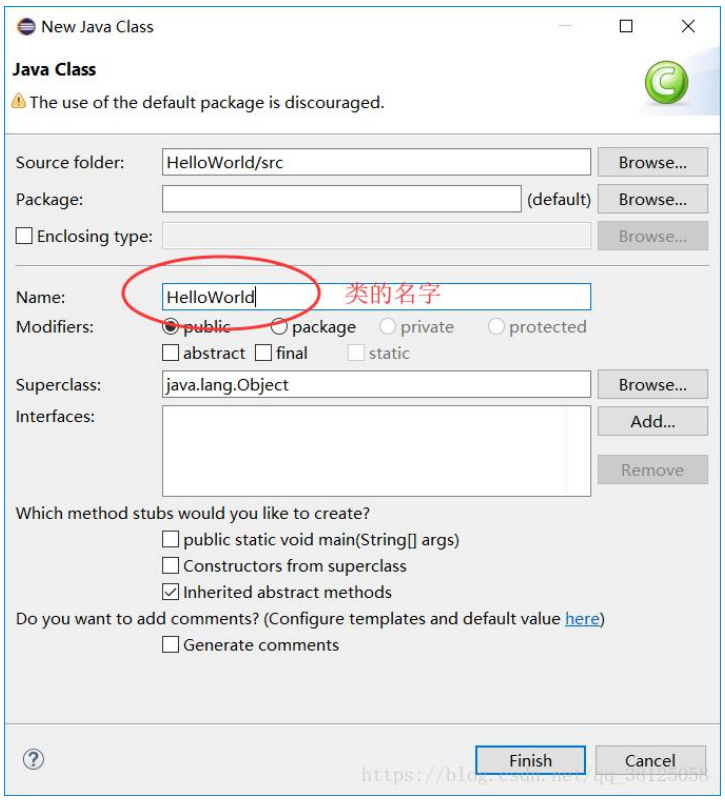




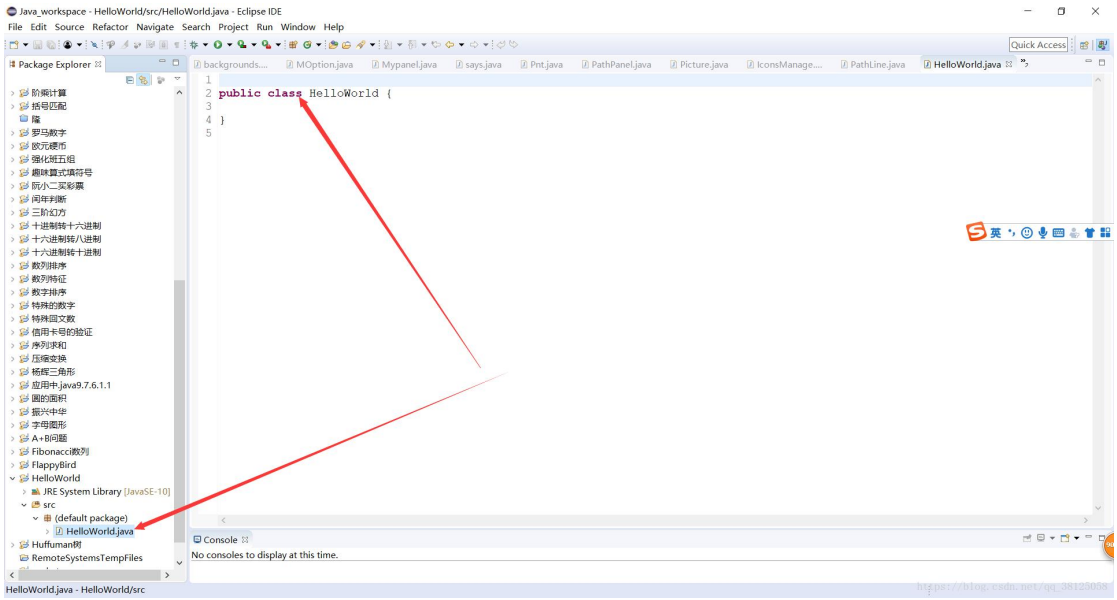

Java_workspace - 666/src/Picture.java - Eclipse
File Edit Source Refactor Navigate Search
Package Explorer
阶乘计算
括号匹配
隆
罗马数字
欧元硬币
强化班五组
趣味算式填符号
阮小二买彩票
闰年判断
三阶幻方
十进制转十六进制
十六进制转八进制
十六进制转十进制
数列排序
数列特征
数字排序
特殊的数字
特殊回文数
信用卡号的验证
序列求和
压缩变换
杨辉三角形
应用中.java9.7.6.1.1
圆的面积
振兴中华
字母图形
A+B问题
Fibonacci数列
FlappyBird
HelloWorld
JRE System Library [JavaSE-10]
src
Huffuman树
RemoteSystemsTempFiles
socket
wuziqi
No c
https://blog.csdn.net/qq_38125058

二、选中刚刚建立的 java 项目，右键点击，新建类



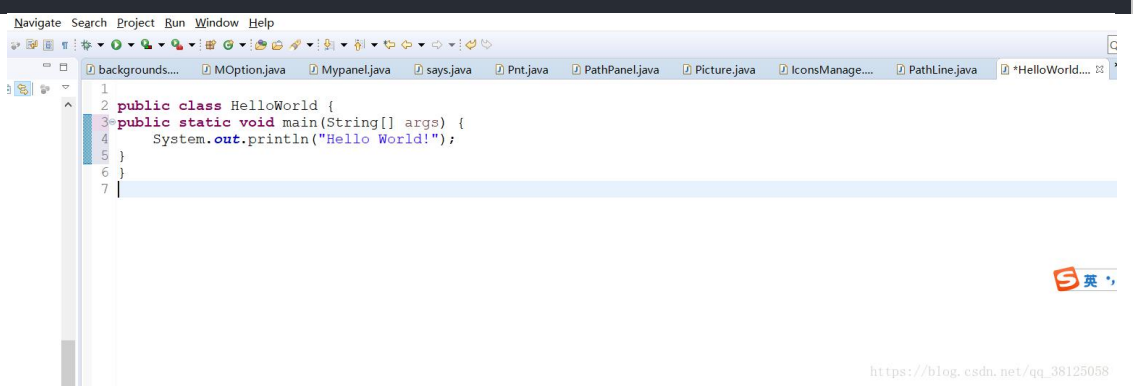


点击 finish，会发现 src 目录下多了一个.java 文件，即我们刚刚建立的 HelloWord 类

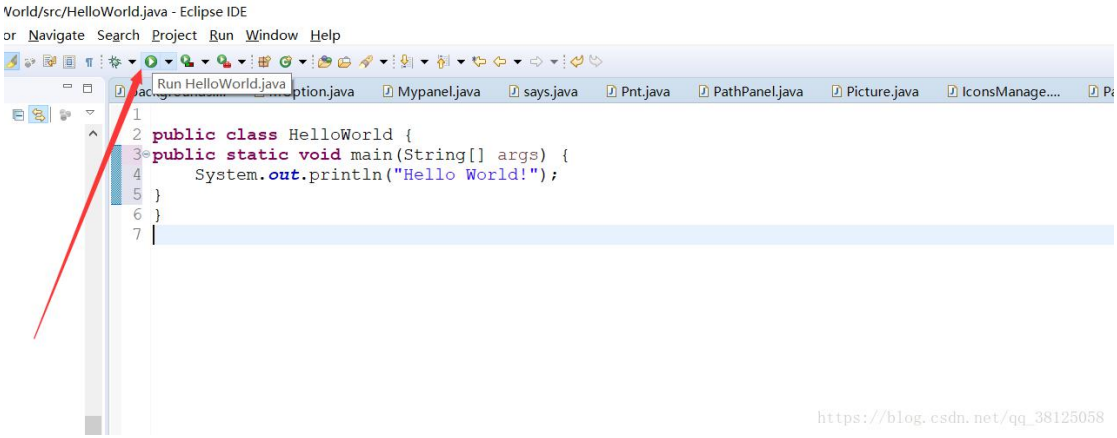


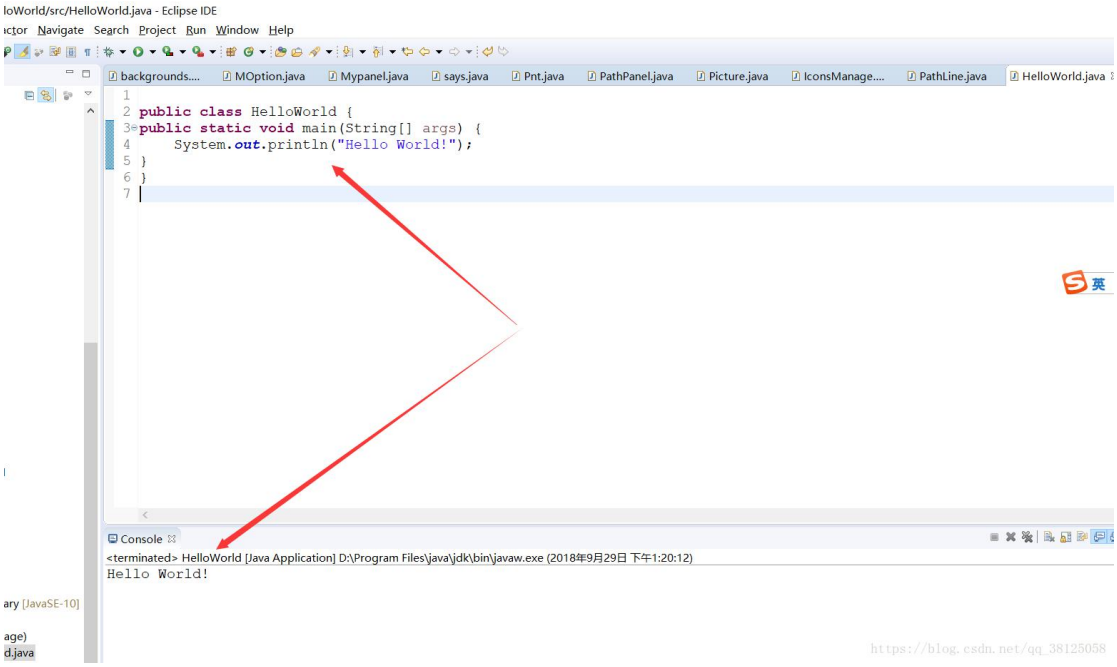
输入如下代码：

```
public static void main(String[] args) {

        System.out.println("Hello World!");

}
```



点击**执行**按钮，如下，就可以看到我们的执行结果了

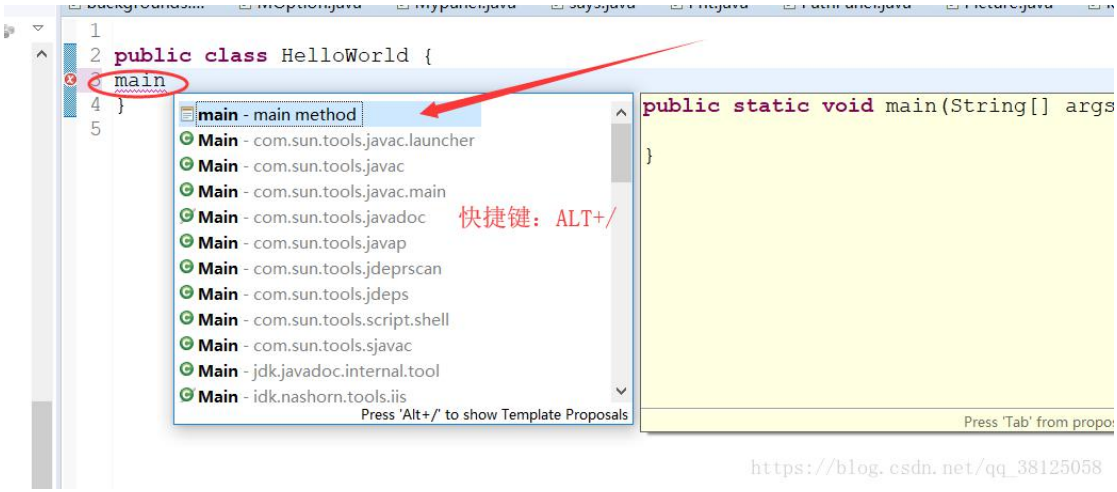


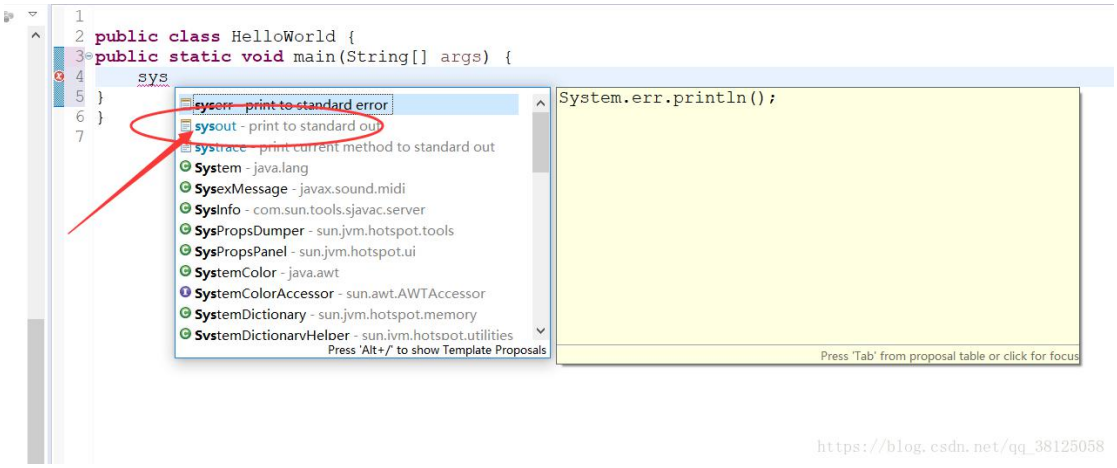


介绍几个快捷键：

1、代码提示：alt+/

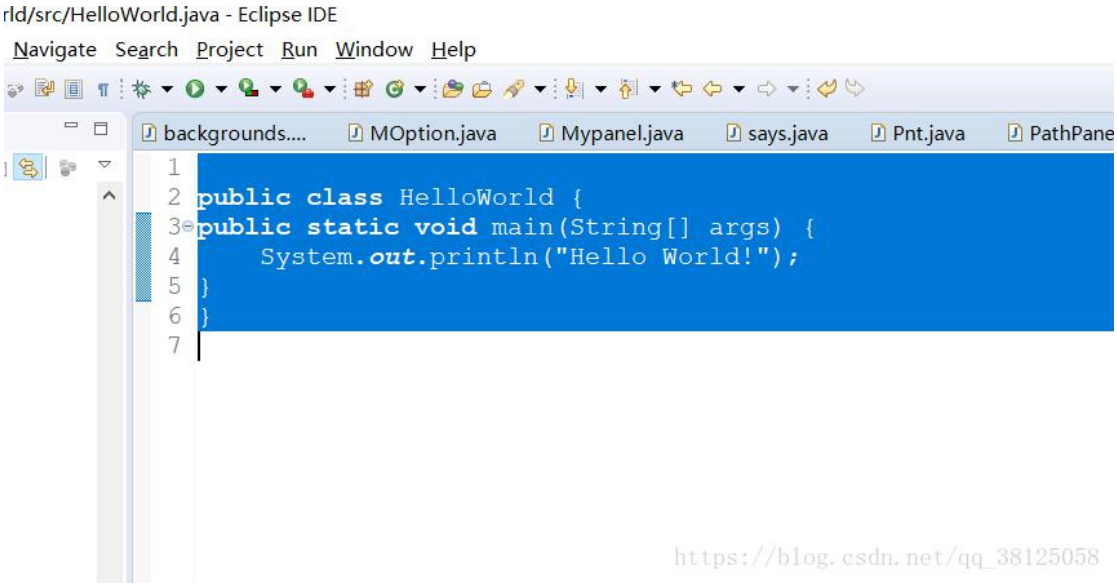
例：





2、全部选中：Ctrl+A

代码对齐：Ctrl+I



3、Ctrl+Z：撤回

Ctrl+Y：反撤回

4、Ctrl+S：保存

5、Ctrl+/：注释

五、 项目方法

使用投影进行讲解演示，并抽样进行检查。

六、 考核办法

此部分项目内容采用抽样考察的方法，考核以操作的熟练程度和正确性为评分标准，以 （优秀）、 （良好）、 （及格）、 （不及格）为成绩标准。

项目二

一、 项目目的和要求

通过对一个 ，使学生了 的步骤，让同学们 的粗略和宏观认识得到细化，使其懂得如何独立面对 的设计。

二、 项目内容

讲解和演示 spring 设计方法。

三、 项目准备

。

四、 项目步骤

Spring 概念

1 spring 是开源的轻量级框架

2 spring 核心主要两部分：

（1）aop：面向切面编程，扩展功能不是修改源代码实现

（2）ioc：控制反转，

- 比如有一个类，在类里面有方法（不是静态的方法），调用类里面的方法，创建类的对象，使用对象调用方法，创建类对象的过程，需要 new 出来对象

- 把对象的创建不是通过 new 方式实现，而是交给 spring 配置创建类对象

3 spring 是一站式框架

（1）spring 在 javaee 三层结构中，每一层都提供不同的解决技术

- web 层：springMVC

- service 层：spring 的 ioc

- dao 层：spring 的 jdbcTemplate

4 spring 版本

（1）hibernate5.x

（2）spring4.x

Spring 的 ioc 操作

1 把对象的创建交给 spring 进行管理

2 ioc 操作两部分：

（1）ioc 的配置文件方式

（2）ioc 的注解方式

IOC 底层原理

1 ioc 底层原理使用技术

（1）xml 配置文件

（2）dom4j 解决 xml

（3）工厂设计模式

（4）反射

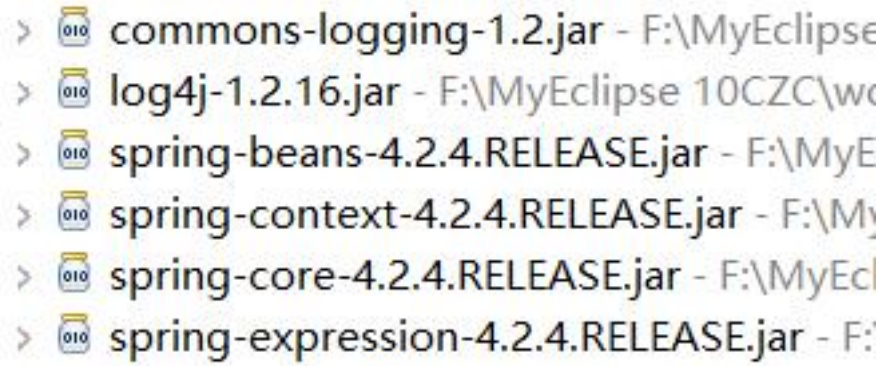
2 画图分析 ioc 实现原理

IOC 入门案例

第一步 导入 jar 包

（1）四个核心 jar 包：

本人导入 jar 如下：



（2）做 spring 最基本功能时候，导入四个核心的 jar 包就可以了

（3）导入支持日志输出的 jar 包

第二步 创建类，在类里面创建方法

```
public class User {

    public void add() {
        System.out.println("add:");
    }

    public static void main(String[] args) {
        // 原始做法
```

```
        User user = new User();
        user.add();
    }
}
```

第三步 创建 spring 配置文件，配置创建类

（1）spring 核心配置文件名称和位置不是固定的

- 建议放到 src 下面，官方建议 applicationContext.xml

（2）引入 schema 约束

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
```

（3）配置对象创建

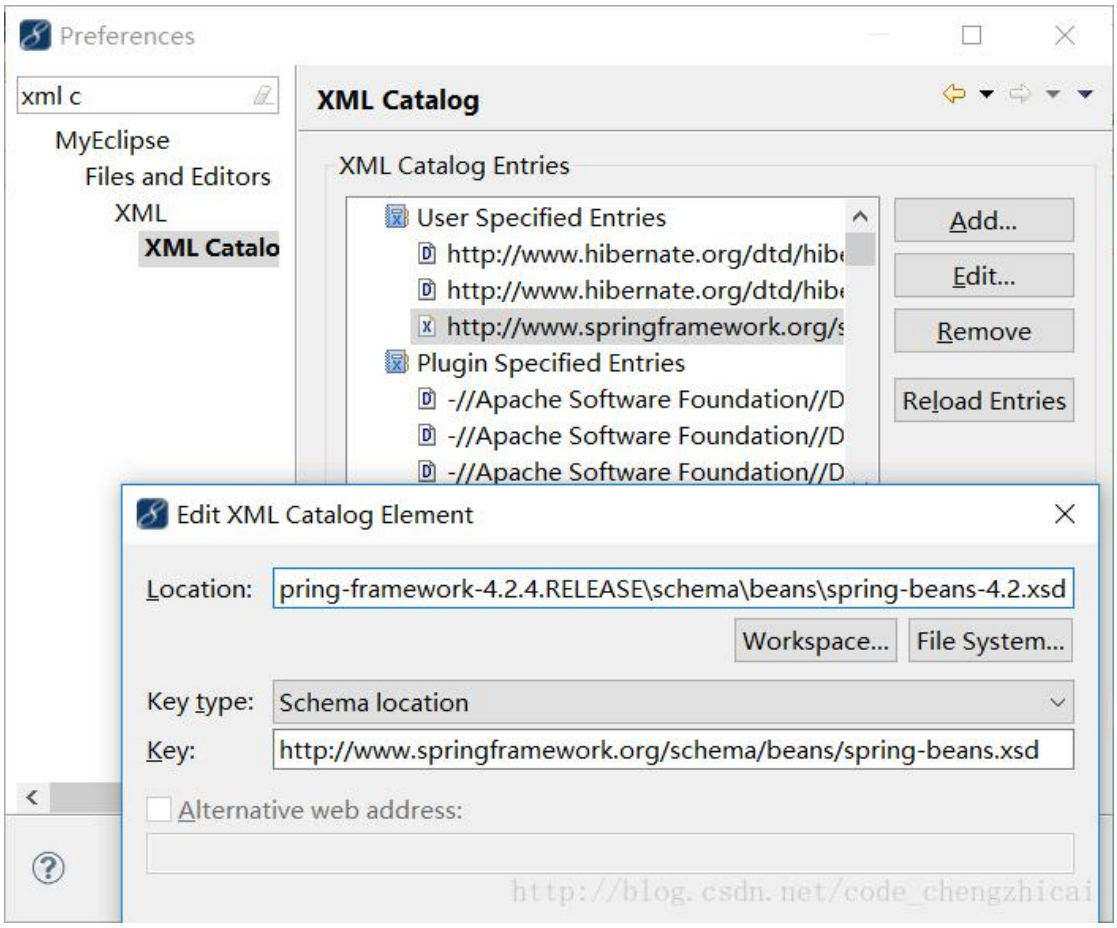
```
<!-- ioc入门 -->
<bean id="user" class="com.lightning.ioc.User"></bean>
```

第四步 写代码测试对象创建

（1）这段代码在测试中使用

```
// 1 加载 spring 配置文件

ApplicationContext context = new

ClassPathXmlApplicationContext("bean1.xml");

// 2 得到配置创建的对象

User user = (User) context.getBean("user");

System.out.println(user);

user.add();
```

配置文件没有提示问题



Spring 的 bean 管理（xml 方式）

Bean 实例化的方式

1 在 spring 里面通过配置文件创建对象

2 bean 实例化三种方式实现

第一种 使用类的无参数构造创建（重点）

```
<!-- ioc 入门 -->

<bean id="user" class="com.lightning.ioc.User"></bean>
```

类里面没有无参数的构造函数，出现异常

```
org.springframework.beans.factory.BeanCreationExceptic
antiate [cn.itcast.ioc.User]: No default constructor found;
```

第二种 使用静态工厂创建

（1）创建静态的方法，返回类对象

```
public class Bean2Factory {


    // 静态的方法，返回 Bean2 对象

    public static Bean2 getBean2() {

        return new Bean2();

    }
}
<!-- 使用静态工厂创建对象 -->

<bean id="bean2" class="com.lightning.bean.Bean2Factory"

factory-method="getBean2">
```

第三种 使用实例工厂创建

（1）创建不是静态的方法，返回类对象

```
public class Bean3Factory {


    public Bean3 getBean3() {

        return new Bean3();

    }
}
<!-- 使用实例工厂创建对象 -->

<!-- 创建工厂对象 -->

<bean id="Bean3Factory" class="com.lightning.bean.Bean3Factory"></bean>

<bean id="bean3" factory-bean="Bean3Factory" factory-method="getBean3">
```

Bean 标签常用属性

（1）id 属性：起名称，id 属性值名称任意命名

- id 属性值，不能包含特殊符号

- 根据 id 值得到配置对象

（2）class 属性：创建对象所在类的全路径

（3）name 属性：功能和 id 属性一样的，id 属性值不能包含特殊符号，但是在 name 属性值里面可以包含特殊符号

（4）scope 属性

- singleton：默认值，单例

```
scope="singleton">
cn.itcast.ioc.User@1be3a66
cn.itcast.ioc.User@1be3a66
```

- prototype：多例

```
cn.itcast.ioc.User@e90eef
cn.itcast.ioc.User@d381e4
```

- request：创建对象把对象放到 request 域里面

- session：创建对象把对象放到 session 域里面

- globalSession：创建对象把对象放到 globalSession 里面

属性注入介绍

1 创建对象时候，向类里面属性里面设置值

2 属性注入的方式介绍（三种方式）

（1）使用 set 方法注入

（2）使用有参数构造注入

（3）使用接口注入

第一种 使用set方法注入

```
public class User {
  private String name;
  public void setName(String name) {
    this.name = name;
  }
}
User user = new User();
user.setName("abcd");
```

第二种 有参数构造注入

```
public class User {
  private String name;
  public User(String name) {
    this.name = name;
  }
}
User user = new User("lucy");
```

第三种 使用接口注入

```
public interface Dao {
  public void delete(String name);
}
public class DaoImpl implements Dao {
  private String name;
  public void delete(String name) {
    this.name = name;
  }
}
```


3 在 spring 框架里面，支持前两种方式

（1）set 方法注入（重点）

（2）有参数构造注入

使用有参数构造注入属性

```
<!-- 使用有参数构造注入属性 -->
<bean id="demo" class="com.lightning.property.PropertyDemo1">
    <!-- 使用有参构造注入 -->
    <constructor-arg name="username" value="小王小马"></constructor-arg>
</bean>
public class PropertyDemo1 {

    private String username;
    public PropertyDemo1(String username) {
        this.username = username;
    }

    public void test1() {
        System.out.println("demo1:" + username);
    }
}
```

使用 set 方法注入属性（重点）

```
public class Book {

    private String bookname;
```

```
    // set 方法
    public void setBookname(String bookname) {
        this.bookname = bookname;
    }

    public void demobook() {
        System.out.println("book:" + bookname);
    }
}
<!-- 使用 set 方法注入属性 -->
<bean id="book" class="com.lightning.property.Book">
    <!-- 注入属性
        name 属性值：类里面定义的属性名称
        value 属性：设置具体的值
    -->
    <property name="bookname" value="六脉神剑"></property>
</bean>
```

注入对象类型属性（重点）

1 创建 service 类和 dao 类

（1）在 service 得到 dao 对象

2 具体实现过程

（1）在 service 里面把 dao 作为类型属性

（2）生成 dao 类型属性的 set 方法

```
public class UserService {


    // 1 定义 dao 属性

    private UserDao userDao;


    // 2 生成 set 方法

    public void setUserDao(UserDao userDao) {

        this.userDao = userDao;

    }
}
```

（3）配置文件中注入关系

```
<!-- 1配置service和dao对象 -->
<bean id="userDao" class="cn.itcast.ioc.UserDao"></bean>

<bean id="userService" class="cn.itcast.ioc.UserService">
    <!-- 注入dao对象
        name属性值：service类里面属性名称
                    现在不要写value属性，因为刚才是字符串，现在是对象
        写ref属性：dao配置bean标签中id值
    -->
    <property name="userDao" ref="userDao"></property>
</bean>
```


P 名称空间注入

```
xmlns:p="http://www.springframework.org/schema/p"
```

```
<!-- p 名称空间注入 -->

<bean id="person" class="cn.itcast.property.Person" p:pname="lucy"></bean>
```

注入复杂类型属性

1 数组

2 list 集合

3 map 集合

4 properties 类型

Person.java:

```
package com.lightning.property;



import java.util.List;

import java.util.Map;

import java.util.Properties;



public class Person {

    private String pname;

    private String[] arrs;

    private List<String> list;

    private Map<String, String> map;

    private Properties properties;

    public void setPname(String pname) {

        this.pname = pname;
```

```
    }

    public void setArrs(String[] arrs) {
        this.arrs = arrs;
    }

    public void setList(List<String> list) {
        this.list = list;
    }

    public void setMap(Map<String, String> map) {
        this.map = map;
    }

    public void setProperties(Properties properties) {
        this.properties = properties;
    }

    public void test1() {
        System.out.println("arrs:"+arrs);
        System.out.println("list:"+list);
        System.out.println("map:"+map);
        System.out.println("properties"+properties);
    }
}
```

配置文件 bean1.xml：

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

<!-- 注入复杂类型属性值 -->
<bean id="person" class="com.lightning.property.Person">
    <!-- 数组 -->
    <property name="arrs">
        <list>
            <value>赵德汉</value>
            <value>丁义珍</value>
            <value>赵瑞龙</value>
        </list>
    </property>

    <!-- list -->
    <property name="list">
        <list>
            <value>侯亮平</value>
            <value>季昌明</value>
            <value>沙瑞金</value>
```

```
        </list>
    </property>

    <!-- map -->
    <property name="map">
        <map>
            <entry key="aa" value="lucy"></entry>
            <entry key="bb" value="jack"></entry>
            <entry key="cc" value="tom"></entry>
        </map>
    </property>

    <!-- properties -->
    <property name="properties">
        <props>
            <prop key="driverclass">com.mysql.jdbc.Driver</prop>
            <prop key="username">root</prop>
        </props>
    </property>
</bean>
</beans>
```

测试类 TestIOC.java：

```
public class TestIOC {
```

```
    @Test
    public void testUser() {
        //1 加载 spring 配置文件，根据里面的配置创建对象
        ApplicationContext context = new
ClassPathXmlApplicationContext("bean1.xml");
        //2 得到配置创建的对象
        Person person = (Person)context.getBean("person");
        person.test1();
    }
}
```

IOC 和 DI 区别

（1）IOC：控制反转，把对象创建交给 spring 进行配置

（2）DI：依赖注入，向类里面的属性中设置值

（3）关系：依赖注入不能单独存在，需要在 ioc 基础之上完成操作

Spring 整合 web 项目原理

1 加载 spring 核心配置文件

```
ApplicationContext context = new
ClassPathXmlApplicationContext("bean1.xml");
```

（1）new 对象，功能可以实现，效率很低

2 实现思想：把加载配置文件和创建对象过程，在服务器启动时候完成

3 实现原理

（1）ServletContext 对象

（2）监听器

（3）具体使用：

- 在服务器启动时候，为每个项目创建一个 ServletContext 对象

- 在 ServletContext 对象创建时候，使用监听器可以具体到 ServletContext 对象在什么时候创建

- 使用监听器监听到 ServletContext 对象创建时候，

– 加载 spring 配置文件，把配置文件配置对象创建

– 把创建出来的对象放到 ServletContext 域对象里面（setAttribute 方法）

- 获取对象时候，到 ServletContext 域得到 （getAttribute 方法）

五、 项目方法

使用投影讲解演示，并抽样进行检查。

六、 考核办法

此部分项目内容采用抽样考察的方法，考核以操作的熟练程度和正确性为评分标准，以 （优秀）、 （良好）、 （及格）、 （不及格）为成绩标准。

项目四 mybatis 操作

一、 项目目的和要求

使学生在 的规划，让学生对自己即将 进行规划并完成站点规划书的撰写。

二、 项目内容

让学生在因特网上申请免费的个人主页空间，向学生讲解站点规划的重要性以及站点规划书的撰写方法，并由学生自行思考并完成项目指导书的撰写。

三、 项目准备

。

四、 项目步骤

模糊查询

项目代码根据入门案例修改，这里只填写修改的内容

**

**

修改映射文件的查询：

```
<select id="findUserByName" parameterType="String"
resultType="com.szl.model.User">
       SELECT * from user where username like '%${value}%'
   </select>
```

测试类中修改的语句：

```
  /*
   * 查询多条记录
   * */
  @Test
   public void test1() throws IOException {

      *  d)      调用 SqlSession 的操作数据库方法。
```

```
      User user=session.selectOne("findUserById",10);
     System.out.println(user);
       List<User> users=session.selectList("findUserByName","张");
     System.out.println(users);


  }
```

插入

映射文件:

```
  <!--插入数据-->
    <!--这里的占位写的是模型的属性-->
    <insert id="insertUser" parameterType="com.szl.model.User">
        insert  into user (username,sex,birthday,address)
        values (#{username},#{sex},#{birthday},#{address});
    </insert>
```

测试类:

```
public class Demo2 {
    SqlSession session;
   @Before
    public void before() throws IOException {
       System.out.println("before......获取 session");
//      *  a)    读取配置文件;
       InputStream is= Resources.getResourceAsStream("SqlMapConfig.xml");
//        *  b)  通过 SqlSessionFactoryBuilder 创建 SqlSessionFactory 会话工
厂。
       SqlSessionFactory factory= new
SqlSessionFactoryBuilder().build(is);
//        *  c)  通过 SqlSessionFactory 创建 SqlSession。
     session=factory.openSession();
    }
    @After
    public void after(){
        System.out.println("after......关闭 session");
//        *  e)  关闭 SqlSession。
        session.close();
    }
```

```
    /*
    * 插入数据
    * */
    @Test
    public void test2() throws IOException {
        User user=new User("abc","1",new Date(),"广州");
      session.insert("insertUser",user);
      session.commit();//提交事务，如果执行操作数据库没有变化就执行这个方法
    }
}
```

删除

映射文件：

```
<!--删除数据-->
    <delete id="deleteUser" parameterType="int">
        delete from user where id=#{ids};
    </delete>
```

测试类：

```
  /*
    * 删除用户
    * */
    @Test
    public void Test3() throws IOException{
        int i=session.delete("deleteUser",16);
        session.commit();
        System.out.println("受影响的行数:"+i);
    }
```

修改用户：

映射文件：

```
<!--修改数据-->
    <update id="updateUser" parameterType="com.szl.model.User">
        update  user set address=#{address},sex=#{sex}
        where  id=#{id};
    </update>
```

测试：

```
/*
   * 修改用户
   * */
   @Test
  public void test4() throws IOException {
       User user=new User();
       user.setId(10);
       user.setSex("女");
       user.setAddress("武汉");

        int i= session.update("updateUser",user);
     session.commit();
       System.out.println("收影响的行数:"+i);
   }
```

插入后返回主键

```
<insert id="insertUser" parameterType="com.gyf.domain.User">
                <!--
                        [selectKey 标签]：通过 select 查询来生成主键
                        [keyProperty]：指定存放生成主键的属性
                        [resultType]：生成主键所对应的 Java 类型
                        [order]：指定该查询主键 SQL 语句的执行顺序，相对于
insert 语句
                        [last_insert_id]：MySQL 的函数，要配合 insert 语句一
起使用 -->
                <selectKey keyProperty="id" resultType="int"
order="AFTER">
                        SELECT LAST_INSERT_ID()
                </selectKey>
                <!-- 如果主键的值是通过 MySQL 自增机制生成的，那么我们此处不需
要再显示的给 ID 赋值 -->
                INSERT INTO USER (username,sex,birthday,address)
                VALUES(#{username},#{sex},#{birthday},#{address})
</insert>
```

映射文件

```
<!--插入返回 id-->
   <insert id="insertUser2" parameterType="com.szl.model.User">
```

```
        <selectKey keyProperty="id" resultType="int" order="AFTER">
            select last_insert_id()
        </selectKey>
        insert  into user (username,sex,birthday,address)
        values (#{username},#{sex},#{birthday},#{address});
    </insert>
```

测试类

```
//* 插入数据后，往模型中设置 id
    //*
    @Test
    public void test2() throws IOException {
        User user=new User("AAAAA","1",new Date(),"广州");
         int i= session.insert("insertUser2",user);
       session.commit();
        System.out.println("收影响的行数:"+i);
        System.out.println("用户 id"+user.getId());
    }
```

主键返回自增 UUID

注:

parameterType 和 resultType

parameterType 指定输入参数的 java 类型，可以填写别名或 Java 类的全限定名。

resultType 指定输出结果的 java 类型，可以填写别名或 Java 类的全限定名。

#{}和${}

#{}：相当于预处理中的占位符？。

#{}里面的参数表示接收 java 输入参数的名称。

#{}可以接受 HashMap、简单类型、POJO 类型的参数。

当接受简单类型的参数时，#{}里面可以是 value，也可以是其他。

#{}可以防止 SQL 注入。

${}：相当于拼接 SQL 串，对传入的值不做任何解释的原样输出。

${}会引起 SQL 注入，所以要谨慎使用。

${}可以接受 HashMap、简单类型、POJO 类型的参数。

当接受简单类型的参数时，${}里面只能是 value。

selectOne 和 selectList

selectOne：只能查询 0 或 1 条记录，大于 1 条记录的话，会报错：

selectList：可以查询 0 或 N 条记录

五、 项目方法

首先由项目指导教师在机房讲解站点规划书的书写方法和基本格式，其次由学生针对所选的主题进行站点规划。

六、 考核办法

此部分项目内容采用全体考察的方法，考核以站点规划书的完整性、实用性和创造性为评分标准，以 （优秀）、 （良好）、 （及格）、 （不及格）为成绩标准。

项目五 springmvc 基本操作

一、 项目目的和要求

在 的基础上，项目 的基本操作。

设计。

二、 项目内容

站点的创建。

绘制草稿图。

利用草稿图对网站各页面进行布局设计。

三、 **项目准备**

。

四、 项目步骤

SpringMVC 框架

回顾

项目开发步骤：

1. 准备 maven 项目
2. 加入 Spring MVC 依赖
3. 在 web.xml 添加大 C 的配置
4. 创建配置类，需要@Configuration, @ComponentScan, @EnableWebMvc
5. 开发控制器

如何做静态资源的处理[让静态资源的请求不经过大 C]

- 详见第一天笔记

控制器的开发

1. 写一个 JAVA 类，打上 @Controller 注解
2. 在控制器，开发处理用户请求的方法，这个方法上要打上@RequestMapping，并且写在请求的 URI。

注:

@RequestMapping 即可以写在类上面，也可以写在 方法上面。

```
@Controller

@RequestMapping("/hello")

public class HelloController {
```

```
@RequestMapping("/abc")
public String hello() {

    return "hello.jsp";
}
}
```

如果在类上面和方法上面都有写 @RequestMapping 的话，则我们的请求 URI 是两者的结合。

请求参数的处理

1. 在 SpringMVC 的控制器方法中，可以使用 Servlet 时代的任意对象做为参数。比如：我们可以在方法中添加 HttpServletRequest,HttpServletResponse,HttpSession, ...
2. 前台表单数据提交后，Spring MVC 可以自动帮助我们封装成 JavaBean，我们可以直接以这个 JAVABEAN 做为方法的参数来接。要求是：表单域的名字要与 JAVABEAN 的属性名保持一样。
3. 有关 Model 参数，它本质就是一个 LinkedHashMap, key 就是字符串，不允许为 NULL，Value 就是 Object，我们可以通过 Model 来添加任意的对象到前端页面。

视图处理器

SpringMVC 框架默认的视图是采用 JSP。

我们可以配置视图解析器。

编码问题

1. 请求的方式如果是 GET 的话，浏览器会把请求数据封装到请求头【Request Header】中传输到后端.
2. 请求的方式如果是 POST 的话，浏览器会把请求数据封装到请求体【Request Body】中传输到后端。

编码过滤器中设置请求和响应的编码，它只能影响请求体部，不能改变头部的编码。换句话说，它只对 POST 请求有效。

如果需要修改请求头的编码，只能修改容器的配置。Tomcat 的配置如下：

```
<Connector connectionTimeout="20000" port="8088" protocol="HTTP/1.1"
redirectPort="8443" URIEncoding="UTF-8"/>
```

如果你使用的是 maven 的 tomcat7 插件，则只需要在 pom.xml 中配置即可：如下：

```
<plugin>

<groupId>org.apache.tomcat.maven</groupId>

<artifactId>tomcat7-maven-plugin</artifactId>

<version>2.2</version>

<configuration>

   <port>8888</port>

    <path>/</path>

    <!-- 指定请求头部的编码 -->

    <uriEncoding>utf8</uriEncoding>

</configuration>

</plugin>
```

SpringMVC 框架

请求的转发和重定向处理

默认情况下，SPRINGMVC 的请求都是以"转发"到下一个资源的，如果在控制方法中，没有指定@ResponseBody 注解，则默认采用 jsp 为视图。

如果我们要重定向一个请求，则需要使用 "redirect:" 开头，后面再接 视图的名字。如："redirect:/user/list"

默认情况下，视图解析器采用转发来跳转请求，那么使用了 forward 关键字和默认情况的区别是什么 ？

1. 使用了 forward: 或 redirect: 为前缀的话，则这个请求跳转会忽略视图解析器。如：

- **return "user/register"; //真正的 URI: /WEB-INF/user/register.jsp**

- **return "forward:/WEB-INF/user/register.jsp"; //自己写完整的 URI**

2. 使用 redirect，相当于是重定向请求。

异常的处理

其实，在 容器层面也有全局的异常处理机制，它是通过在 web.xml 中配置 的标签来实现的，如下：

```
<error-page>

  <exception-type>java.lang.Exception</exception-type>

  <location>/error.jsp</location>

</error-page>

<error-page>

  <error-code>404</error-code>

  <location>/404.jsp</location>

</error-page>
```

当容器收到控制器抛出的异常时或者容器出现了服务端错误【500】时，则根据配置把页面转发到 /error.jsp 中，在 这个页面是，通过在 page 指令中加入 isErrorpage="true" 属性后，就可以拿到容器中出现的异常对象.

在 SpringMVC 中，我们也可以针对异常进行处理，这个处理是由大 C 来负责。有如下方式：

1. 配置全局的异常处理器

   在 WebMvcConfig.java 中,添加一个 HandlerExceptionResolver 的 @Bean, 如下：

```
@Bean

public HandlerExceptionResolver exceptionHandler() {

    //创建 SimpleMappingExceptionResolver

    SimpleMappingExceptionResolver exceptionResolver =

                                new
SimpleMappingExceptionResolver();
```

```
        //设置属性
        exceptionResolver.setDefaultErrorView("forward:/error.jsp");
        //排除的异常

    exceptionResolver.setExcludedExceptions(LoginException.class);
        //
        exceptionResolver.setDefaultStatusCode(500);
        //

    exceptionResolver.setWarnLogCategory("org.springframework.web.servlet.handler.SimpleMappingExceptionResolver");
        //返回
        return exceptionResolver;
    }
```

注： exceptionResolver.setWarnLogCategory 是表示打开全局的异常日志功能

2. 配置控制器级别的异常处理器

利用注解 @ExceptionHandler 来修饰一个方法，这个方法必需要在 @Controller 修饰的类中，这样一来，此 Controller 中的方法有任何一个抛出异常后，框架都会回调 @ExceptionHandler 修饰的方法，这样一来，我们就可以在此方法中做异常信息的日志、记录或信息的展示，并把某些信息绑定一前台页面中供用户查看【这个由程序员决定】

它的代码如下：

```
@ExceptionHandler(value = {RuntimeException.class,
    IOException.class})
 public String initEx(Model model,Exception e) {
        System.out.println("---> ExceptionController 中的方法出异常了...");
     /* System.out.println(e.getMessage());
```

```
        e.printStackTrace();*/
        model.addAttribute("server.ex",e);
        //
        return "forward:/ex.jsp";
    }
```

以上只是一个代码片断。

拦截器的使用[Interceptor]

注：

SpringMVC 的拦截器是针对控制器级别的拦截，它由 大 C 来处理，而不是由容器来处理。它主要是针对控制器的拦截，下面的图示分别说明了 Filter, Interceptor, AOP 三种拦截的范围

开发拦截器的步骤：

1. 写一个类实现 HandlerInterceptor 接口，并重写它的三个方法
    - preHandle 方法 ，在调用目标小 C 之前
    - postHandle 方法， 在调用目标小 C 之后，但是视图渲染之前
    - afterCompletion 方法，视图渲染之后
2. 配置这个拦截器，采用 xml 的配置法

拦截器的使用:

把小 C 中需要执行的一些共性操作拿出来做为 拦截器的功能。比如：

日志，认证，授权，性能监测....

此处我们编写一个拦截器，用来记录每一次请求的处理时间，为了精切地记录这个时间值，我们建议采用拦截器来完成，从 preHandle 方法中开始计算，在 afterCompletion 方法结束计时，并计算出两个时间差，把这个时间差写入到数据库的性能监控表中。

注：

这里我们不去直接实现 HandlerInterceptor 类，而是继承一个抽象父类 HandlerInterceptorAdapter

注：

当我们配置多个拦截器时，它的 preHandle 方法是按顺序执行的，而 postHandle 和 afterCompletion 方法则是按逆序进行执行的。

json 数据格式的使用

在 SpringMVC 框架中，@RequestMapping 方法一般有如下几种返回值：

1. 返回 String, 表示视图的逻辑名【前提是配置视图解析器】
2. 返回 void,表示没有返回值，此时，你需要在方法体中，通过 request 或是 response 来编程进行跳转或重定向，当然，也可以通过 response.getWriter.println(".....") 输出内容到浏览器端。
3. 返回 ModelAndView, 表示返回一个模型和视图，我们需要自己创建这个对象，并且设置 Model 数据和视图的名字。
4. 如果不是以上三种，那就是返回单个 JAVABEAN 或是它的集合体，这种情况一般都需要把这个数据转换成 JSON 格式，这个过程是自动的，只需要在 RequestMapping 中，添加 produce 属性，指定生成的内容就是 json,同时，还要加上 @ResponseBody 注解。

案例：

文件上传和下载

构建 SpringMVC 框架的结构

1. 构建 maven 项目
2. 添加 spring mvc 框架相关的依赖
3. **在 web.xml 中配置大 C 以及编码过滤器**

Spring MVC 框架的核心 API 和核心注解

1. DispatcherServlet,俗称大 C， 也就是前置控制器
2. 核心注解：

    - @EnableWebMvc
    - @Import 导入另一个配置类
    - @ImportResource 读入 spring 框架的 xml 配置
    - @PropertySource 读入 .properties 属性文件

SpringMVC 框架的注解配置类

一般的策略是： 把 **WEB** 层、业务层和持久层 分开配置，其中，**AppConfig.java** 配置类负责像数据源、持久层的支持类、事务、等。而在 **WebMvcConfig.java** 配置类中负责像 静态资源处理器、视图解析器、文件上传处理器、**BeanValidation** 处理器、全局异常处理等

SpringMVC 的注解配置类需要继承 WebMvcConfigurerAdapter 类，它的意义在于给我们一个机会，让我们参与到配置的过程当中。

有关这个类的详细的配置，请参考代码。

有关大 C 的匹配规则

1. 扩展名匹配， 如: *.do, *.action, *.xxx
2. 通配匹配， 如： /

如何处理静态资源？

只是针对大 C 采用 / 来做为请求匹配规则时才需要的操作。

控制器的开发

记住如下几个核心注解：

1. @Controller 它是 @Component 的一种
2. @RequestMapping
3. @ResponseBody
4. @RequestParam
5. @PathVariable
6. @ModelAttribute
7. @SessionAttribute
8. @ExceptionHandler
9. @RestController 支持 RESTFul 风格 URI 的控制器，它相当于：
   @Controller+@ResponseBody
10. ....

与业务层、持久层的整合

直接导入 AppConfig.java 即可。

SpringMVC 中的重要功能点

1. 静态资源处理
2. SpringMVC 对请求参数的自动化封装
3. 请求的转发和重定向
4. 视图解析器的配置
5. 异常的处理
6. 拦截器的开发
7. 返回 json 数据格式
8. 文件上传和下载
9. Bean Validation
10. RESTFUL 风格的 URI

文件上传和下载

文件的上传必需要使用 post 方式提交，所以，我们需要定义一个表单

另外一个，表单的 enctype 一定要指定为: multipart/form-data

在服务端，需要能过一个特殊对象： MultipartFile 来接收用户的文件上传请求。

Bean Validation

前端页面的验证是极其不安全的，可以轻易地跳过前端的验证。

为了数据的有效性和一致性，我们在后端也需要对前端传递过来的数据进行验证。

JSR303 规范就是定义了 Bean Validation 的规范，目前，官方说明中，Hibernate Validator 组件是 JSR303 的参考实现。所以，Spring MVC 框架结合 Hibernate Validator。

操作步骤

1. 导入 hibernate-validtor 的依赖

2. 在 WebMvcConfig.java 中重写父类的 getValidator 方法，在实现中返回 LocalValidatorFactoryBean
3. 在需要验证的对象中，使用 JSR303 的注解进行注解
4. 在控制器的方法参数中，使用 @Validated 注解需要验证的对象，并添加 BindingResult 对象为参数。

五、 项目方法

机房操作。

六、 考核办法

此部分项目内容采用全体考察的方法，以百分制为满分，具体评分标准如下：

站点创建的规范性和熟练程度（ 分）

草稿图的绘制（ 分）

草稿图的载入（ 分）

布局视图的相关操作（ 分）

七、 思考和练习

布局视图和表格视图在版面布局中各有优势和劣势？对怎样的版面布局应该用布局视图，又对怎样的版面布局应该用表格视图？

项目六 SSM 合成

一、 项目目的和要求

按照项目四所完成的　　设计和制作中资源的搜集和创建工作。

二、 项目内容

在因特网上搜索资源或在本机上创建所需资源。

三、 项目准备

。

四、 项目步骤

SSM 合成

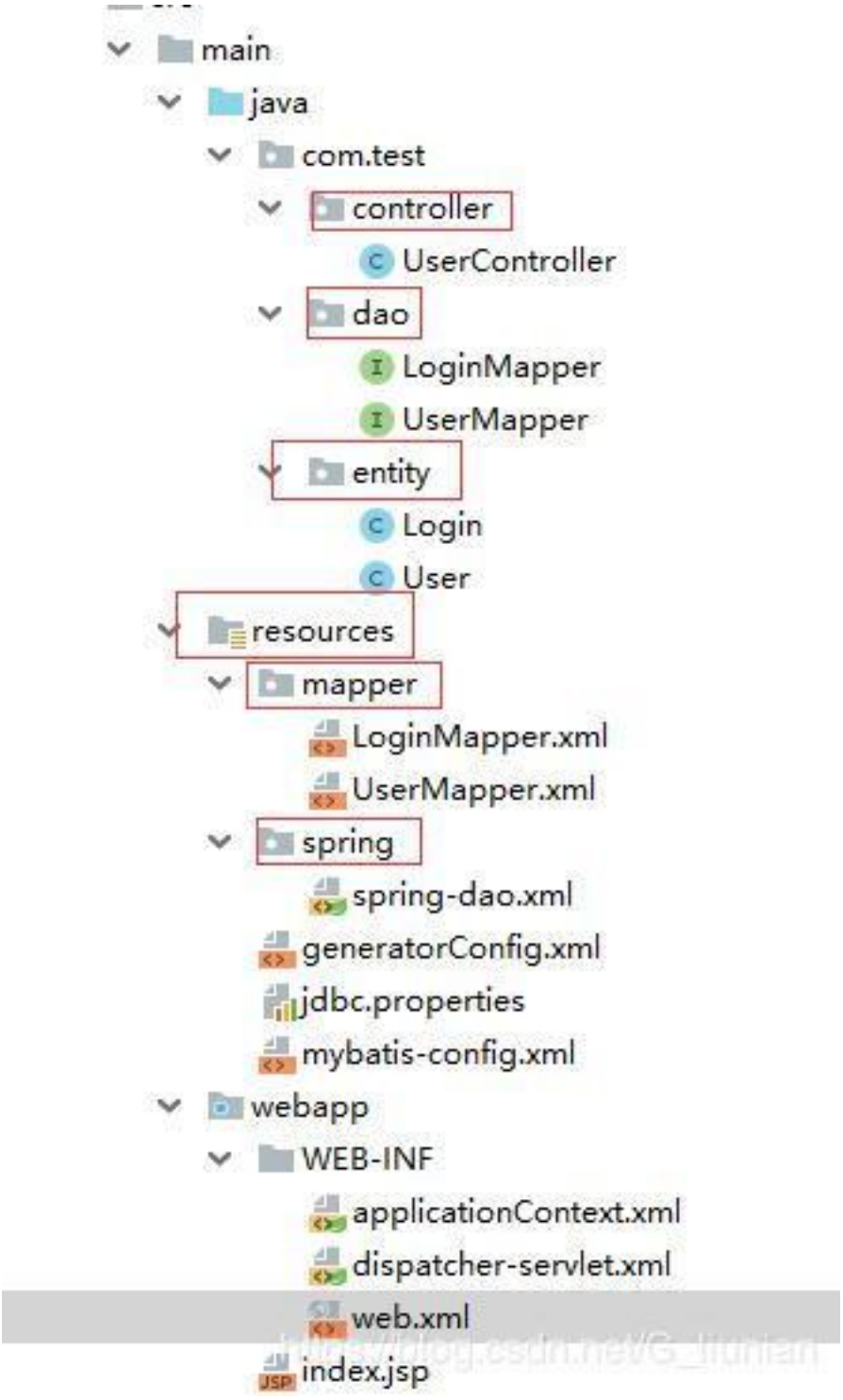
一、创建基于 maven 的 SpringMVC 框架

点这里

注意：dispatcher.xml 不需要配置

二、配置边角（我里面有可以自动生成 bean，mapper 等的配置文件）

1、创建包：



resources 建发与 java 减法类似，看第一部的链接

三、直接上代码

pom.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/xsd/maven-4.0.0.xsd">

  <modelVersion>4.0.0</modelVersion>



  <groupId>com.nian</groupId>

  <artifactId>TestSSM</artifactId>

  <version>1.0-SNAPSHOT</version>

  <packaging>war</packaging>



  <name>TestSSM Maven Webapp</name>

  <!-- FIXME change it to the project's website -->

  <url>http://www.example.com</url>



  <properties>

    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
```

```
<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

    <maven.compiler.source>1.7</maven.compiler.source>

    <maven.compiler.target>1.7</maven.compiler.target>



    <!--lib setting-->

    <spring.version>4.3.3.RELEASE</spring.version>

    <jackson.version>2.6.3</jackson.version>

  </properties>



  <dependencies>

    <!-- config junit jar -->

    <dependency>

      <groupId>junit</groupId>

      <artifactId>junit</artifactId>

      <version>4.12</version>

      <scope>test</scope>

    </dependency>
```

```
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>5.1.39</version>
  <scope>runtime</scope>
</dependency>
<dependency>
  <groupId>c3p0</groupId>
  <artifactId>c3p0</artifactId>
  <version>0.9.1.1</version>
</dependency>


<!-- DAO 框架：mybatis-->
<dependency>
  <groupId>org.mybatis</groupId>
  <artifactId>mybatis</artifactId>
  <version>3.4.1</version>
</dependency>
<dependency>
  <groupId>org.mybatis</groupId>
```

```
    <artifactId>mybatis-spring</artifactId>
    <version>1.3.0</version>
  </dependency>


  <!-- servlet 相关依赖-->
  <dependency>
    <groupId>taglibs</groupId>
    <artifactId>standard</artifactId>
    <version>1.1.2</version>
  </dependency>
  <dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
    <version>1.2</version>
  </dependency>
  <dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.6.7</version>
  </dependency>
```

```
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>javax.servlet-api</artifactId>
  <version>3.1.0</version>
</dependency>

<!--spring 依赖-->
<!--1)核心依赖-->
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-core</artifactId>
  <version>${spring.version}</version>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-beans</artifactId>
  <version>${spring.version}</version>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
```

```
    <artifactId>spring-context</artifactId>
    <version>${spring.version}</version>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context-support</artifactId>
    <version>${spring.version}</version>
  </dependency>
  <!--2)spring DAO-->
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
    <version>${spring.version}</version>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-tx</artifactId>
    <version>${spring.version}</version>
  </dependency>
  <!--3)spring web-->
```

```
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-web</artifactId>
      <version>${spring.version}</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-webmvc</artifactId>
      <version>${spring.version}</version>
    </dependency>
    <!--4)spring test-->
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-test</artifactId>
      <version>${spring.version}</version>
    </dependency>


    <!--
https://mvnrepository.com/artifact/org.mybatis.generator/mybatis-genera
tor-core -->
    <dependency>
```

```
    <groupId>org.mybatis.generator</groupId>
    <artifactId>mybatis-generator-core</artifactId>
    <version>1.3.7</version>
  </dependency>
  <!-- mybatis 工具包-->
  <dependency>
    <groupId>tk.mybatis</groupId>
    <artifactId>mapper</artifactId>
    <version>3.4.2</version>
  </dependency>

  <!--代码生成器依赖-->
  <dependency>
    <groupId>org.freemarker</groupId>
    <artifactId>freemarker</artifactId>
    <version>2.3.23</version>
    <scope>test</scope>
  </dependency>

</dependencies>
```

```
 <build>
   <finalName>TestSSM</finalName>


   <plugins>
     <plugin>
       <groupId>org.mybatis.generator</groupId>
       <artifactId>mybatis-generator-maven-plugin</artifactId>
       <version>1.3.5</version>
       <configuration>
         <!--配置文件的位置-->
         <configurationFile>src/main/resources/generatorConfig.xml
         </configurationFile>
         <overwrite>true</overwrite>
         <verbose>true</verbose>
       </configuration>
     </plugin>
   </plugins>
   <!--<pluginManagement>&lt;!&ndash; lock down plugins versions to avoid
using Maven defaults (may be moved to parent pom) &ndash;&gt;
     <plugins>
```

```
      <plugin>
        <artifactId>maven-clean-plugin</artifactId>
        <version>3.1.0</version>
      </plugin>
      &lt;!&ndash; see
http://maven.apache.org/ref/current/maven-core/default-bindings.html#Pl
ugin_bindings_for_war_packaging &ndash;&gt;
      <plugin>
        <artifactId>maven-resources-plugin</artifactId>
        <version>3.0.2</version>
      </plugin>
      <plugin>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.8.0</version>
      </plugin>
      <plugin>
        <artifactId>maven-surefire-plugin</artifactId>
        <version>2.22.1</version>
      </plugin>
      <plugin>
        <artifactId>maven-war-plugin</artifactId>
```

```
        <version>3.2.2</version>
      </plugin>
      <plugin>
        <artifactId>maven-install-plugin</artifactId>
        <version>2.5.2</version>
      </plugin>
      <plugin>
        <artifactId>maven-deploy-plugin</artifactId>
        <version>2.8.2</version>
      </plugin>
    </plugins>
  </pluginManagement>-->
 </build>
</project>



<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:p="http://www.springframework.org/schema/p"
      xmlns:context="http://www.springframework.org/schema/context"
```

```
      xmlns:mvc="http://www.springframework.org/schema/mvc"
xmlns:classpath="http://www.springframework.org/schema/c"

      xsi:schemaLocation="http://www.springframework.org/schema/beans


http://www.springframework.org/schema/beans/spring-beans-3.0.xsd

          http://www.springframework.org/schema/context


http://www.springframework.org/schema/context/spring-context-3.0.xsd
http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc.xsd">



    <!-- 启动 SpringMVC 的注解功能，它会自动注册 HandlerMapping、
HandlerAdapter、ExceptionResolver 的相-->

    <mvc:annotation-driven />

    <mvc:default-servlet-handler />



    <!--扫描,Controller 类-->

    <context:component-scan base-package="com.test"/>



    <context:property-placeholder location="classpath:jdbc.properties"/>




    <bean id="dataSource"
class="com.mchange.v2.c3p0.ComboPooledDataSource" destroy-method="close"
```

```
        p:driverClass="${driver}"

        p:jdbcUrl="${url}"

        p:user="${username}"

        p:password="${password}"

        p:idleConnectionTestPeriod="${idleConnectionTestPeriod}"

        p:maxIdleTime="${maxIdleTime}"

        p:acquireIncrement="${acquireIncrement}"

        p:initialPoolSize="${initialPoolSize}"

        p:maxPoolSize="${maxPoolSize}"

        p:minPoolSize="${minPoolSize}"

        p:autoCommitOnClose="${autoCommitOnClose}"

        p:checkoutTimeout="${checkoutTimeout}"

        p:acquireRetryAttempts="${acquireRetryAttempts}"

        p:preferredTestQuery="SELECT 1"

        p:maxConnectionAge="3000"/>



    <!-- mybatis 文件配置，扫描所有 mapper 文件 -->

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

        <property name="dataSource" ref="dataSource"/>
```

```
        <property name="configLocation"
value="classpath:mybatis-config.xml"/>



        <property name="typeAliasesPackage" value="com.test.entity"/>



        <property name="mapperLocations" value="classpath:mapper/*.xml"/>

    </bean>



    <!-- spring 与 mybatis 整合配置，扫描所有 dao,指定的映射器类是接口,接口方法
可以用注解来指定 SQL 语句,但是 MyBatis 的映射器 XML 文件也可以用。 -->

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

          p:basePackage="com.test.dao"

          p:sqlSessionFactoryBeanName="sqlSessionFactory"/>



    <bean id="defaultViewResolver"
class="org.springframework.web.servlet.view.InternalResourceViewResolve
r">

        <property name="viewClass"
value="org.springframework.web.servlet.view.JstlView"/>

        <property name="prefix" value="/"/><!--设置 JSP 文件的目录位置-->

        <property name="suffix" value=".jsp"/>

        <property name="exposeContextBeansAsAttributes" value="true"/>

    </bean>
```

```
</beans>
```

mybatis-config.xml

```
<?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>



    <!-- mybatis 全局设置 -->

    <settings>

        <!--使用数据库自增 id-->

        <setting name="useGeneratedKeys" value="true" />



        <setting name="useColumnLabel" value="true" />



        <!-- 开启驼峰命名规范-->

        <setting name="mapUnderscoreToCamelCase" value="true" />

    </settings>



</configuration>
```

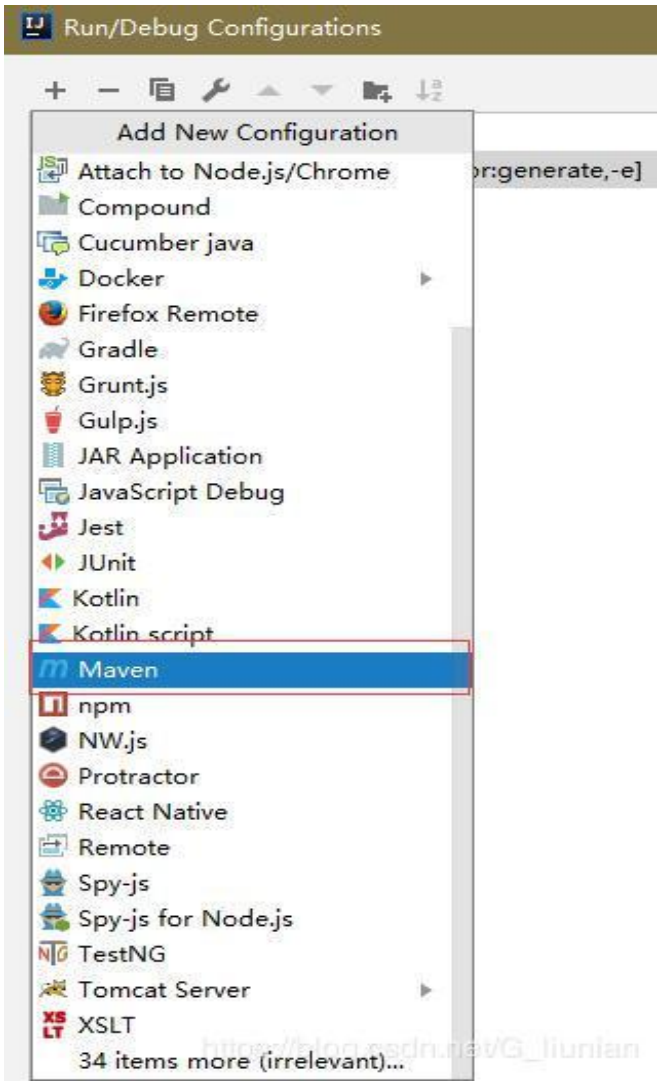
jdbc.properties

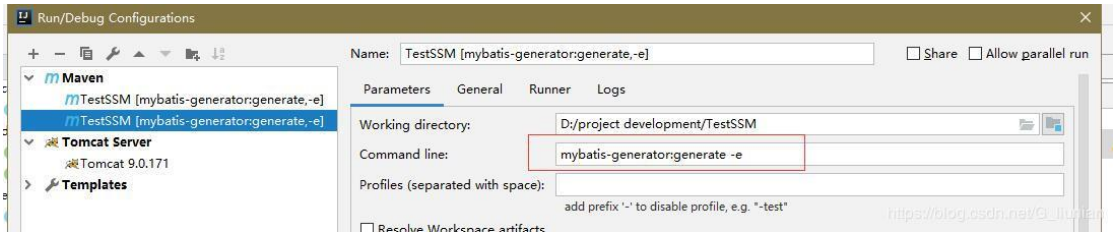
```
driver=com.mysql.jdbc.Driver

url=jdbc:mysql://127.0.0.1:3306/lian

username=root

password=root



idleConnectionTestPeriod=60

maxIdleTime=240

acquireIncrement=5

initialPoolSize=10

maxPoolSize=30

minPoolSize=10

autoCommitOnClose=false

checkoutTimeout=1000

acquireRetryAttempts=2

<?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>
```

```
    <!-- 数据库驱动-->

    <classPathEntry
location="D:\maven\mysql\mysql-connector-java\5.1.39\mysql-connector-ja
va-5.1.39.jar"/>

    <context id="DB2Tables"  targetRuntime="MyBatis3">

        <commentGenerator>

            <property name="suppressDate" value="true"/>

            <!-- 是否去除自动生成的注释 true：是 ：  false:否 -->

            <property name="suppressAllComments" value="true"/>

        </commentGenerator>

        <!--数据库链接 URL，用户名、密码 -->

        <jdbcConnection driverClass="com.mysql.jdbc.Driver"
connectionURL="jdbc:mysql://127.0.0.1/lian" userId="root"
password="root">

        </jdbcConnection>

        <javaTypeResolver>

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

        </javaTypeResolver>

        <!-- 生成模型的包名和位置-->

        <javaModelGenerator targetPackage="com.test.entity"
targetProject="src/main/java">

            <property name="enableSubPackages" value="true"/>

            <property name="trimStrings" value="true"/>
```

```
        </javaModelGenerator>

        <!-- 生成映射文件的包名和位置-->

        <sqlMapGenerator targetPackage="mapper"
targetProject="src/main/resources">

            <property name="enableSubPackages" value="true"/>

        </sqlMapGenerator>

        <!-- 生成 DAO 的包名和位置-->

        <javaClientGenerator type="XMLMAPPER"
targetPackage="com.test.dao" targetProject="src/main/java">

            <property name="enableSubPackages" value="true"/>

        </javaClientGenerator>

        <!-- 要生成的表 tableName 是数据库中的表名或视图名 domainObjectName
是实体类名-->

        <table tableName="user" domainObjectName="User"
enableCountByExample="false" enableUpdateByExample="false"
enableDeleteByExample="false" enableSelectByExample="false"
selectByExampleQueryId="false"></table>

        <table tableName="login" domainObjectName="Login"
enableCountByExample="false" enableUpdateByExample="false"
enableDeleteByExample="false" enableSelectByExample="false"
selectByExampleQueryId="false"></table>

    </context>

</generatorConfiguration>
```

四、配置自动生成





mybatis-generator:generate -e

五、 项目方法

机房上网并利用本机软件完成。

六、 考核办法

此部分是学生在因特网和本机上进行资源的搜集和创建，故无需进行考核，

但需要指导学生按照站点规划书的要求完成站点资源的搜集和创建工作。

项目十二　总结

一、 项目目的和要求

将学生制作的作品进行综合的考核，并进行总结。

二、 项目内容

对学生作品进行考核。

选择典型的（优秀的和劣质的）作品分别进行总结。

三、 项目准备

中文版、　　以上中文版本。

连接因特网的局域网。

四、 项目步骤

对学生的作品依次进行综合考核。

抽取典型（优秀和劣质）的作品进行全面的解析。

五、 项目方法

机房利用本机软件完成。

六、 考核办法

此部分项目以考核为主，对学生组品进行整体的考核。采用全体考察的方法，以百分制为满分，具体评分标准如下：

系统文档　20 分

编写代码　30 分

程序调试　10 分

项目出勤　20 分

技术含量　10 分

美工设计　10 分

七、 思考与练习

无。