

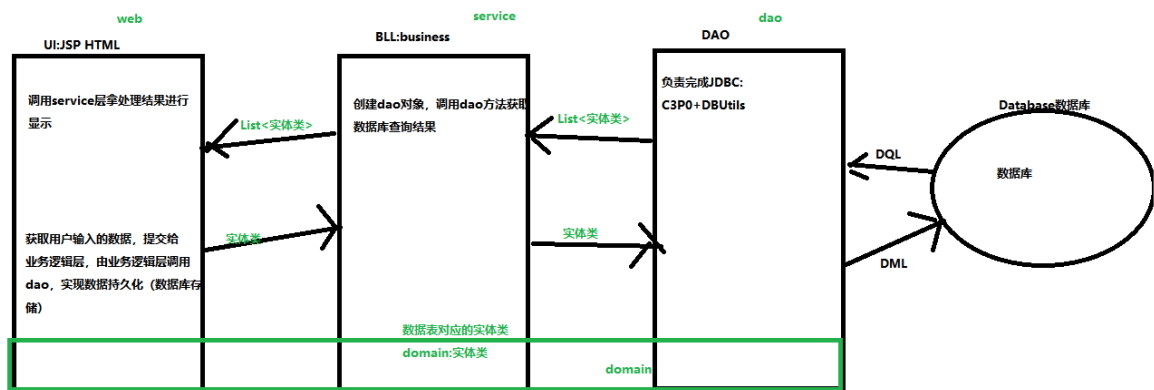
课程回顾

```
1  基于三层结构，完成一个多条件的查询语句，并处理结果
2  查询条件是：Scanner接收用户输入：员工姓名和岗位，
3
4  按照员工姓名和员工岗位的模糊查询
5  select * from emp where ename like '%a%' and job='销售员'
6
7  最后在测试类中，使用sout输出符合条件的员工信息，输入结果如下所示：
8  员工姓名    岗位    薪资    comm    部门编号
9  . . . . .
10
11 需求分析：
12  1.C3PO数据库连接池：负责数据库连接对象获取和管理
13  2.DBUtils：第三方开源组织提供一套用来处理JDBC的封装类：执行DML DQL语句
14  3.三层架构
15
16
17 实现功能：
18 三层架构开发顺序：
19  1.domain:数据表转换后对应的实体类
20 简单粗暴：将一个数据库中所有的表一次性全部转换domain中
21  2.dao层：DAO模式-为了解耦，
22      要求dao层，
23      一个表对应的一个接口    接口名：表名+Dao
24      一个接口对应一个实现类    接口实现类名：表名+Dao+Impl
25  public interface 接口{
26      //四种方法
27      List<> selectBy();
28      实体类类型 selectByPK(int 主键);
29      Object selectScalar(...); //select后+count() max()聚合函数
30
31      int insert(实体类类型 对象);
32      int update(实体类类型 对象);
33      int delete(主键... 参数);
34  }
35  3.service层：
36      一个表对应的一个接口，接口命名：表名+service
37      一个接口对应的一个实现，实现类命名：表名+service+Impl
38  public interface 接口{
39      //四种方法
40      List<> getBy();
41      实体类类型 getByPK(int 主键);
42      Object getScalar(...); //select后+count() max()聚合函数
43
44      int save(实体类类型 对象);
45      int edit/modify(实体类类型 对象);
46      int remove(主键... 参数);
47
48  }
49  public interface 实现类 implements 接口{
50      //创建该表对应的dao的对象
51      //四种方法
52      List<> getBy(){
```

```

53     return dao对象.select。。。()
54 }
55 实体类类型 getByPK(int 主键);
56 Object getScalar(...); //select后+count() max() 聚合函数
57
58 int save(实体类类型 对象);
59 int edit/modify(实体类类型 对象);
60 int remove(主键... 参数);
61
62 }
63
64 4. 测试类:
65 4-1 Scanner获取用户输入的数据
66 4-2 new Service层对象, 调用service的方法, 并接受返回值
67 4-3 使用sout()输出结果

```



课程目标

1 三层结构实现：多条件查询 ===== 掌握

2 安装tomcat服务器 ===== 掌握

3 IDEA创建并运行WEB项目 ===== 掌握

课程实施

1 三层结构实现：多条件查询

domain层

```

1 package cn.kgc.domain;
2
3 import java.math.BigDecimal;
4 import java.util.Date;
5
6 /**
7  * @Author: lc
8  * @Date: 2022/5/13
9  * @Description: 实体类:
10  * 存在的代码:
11  * 1. 与表列同名、同类型的属性

```

```
12  * 2. 无参构造方法
13  * 3. 建议: 重写toString() 一般为了方便测试!!
14  * @Version: 1.0
15  */
16  public class Emp {
17      /**
18       * 兼容NULL
19       * 数据表中遇到int float double, 转换java类时, 要求使用对应包装类类型
20       */
21      private Integer empNo;
22      private String eName;
23      private String job;
24      private Integer mgr;
25      private Date hireDate;
26      private BigDecimal sal;
27      private BigDecimal comm;
28      private Integer deptNo;
29
30      public Integer getEmpNo() {
31          return empNo;
32      }
33
34      public void setEmpNo(Integer empNo) {
35          this.empNo = empNo;
36      }
37
38      public String geteName() {
39          return eName;
40      }
41
42      public void seteName(String eName) {
43          this.eName = eName;
44      }
45
46      public String getJob() {
47          return job;
48      }
49
50      public void setJob(String job) {
51          this.job = job;
52      }
53
54      public Integer getMgr() {
55          return mgr;
56      }
57
58      public void setMgr(Integer mgr) {
59          this.mgr = mgr;
60      }
61
62      public Date getHireDate() {
63          return hireDate;
64      }
65
66      public void setHireDate(Date hireDate) {
67          this.hireDate = hireDate;
68      }
69  }
```

```

70     public BigDecimal getSal() {
71         return sal;
72     }
73
74     public void setSal(BigDecimal sal) {
75         this.sal = sal;
76     }
77
78     public BigDecimal getComm() {
79         return comm;
80     }
81
82     public void setComm(BigDecimal comm) {
83         this.comm = comm;
84     }
85
86     public Integer getDeptNo() {
87         return deptNo;
88     }
89
90     public void setDeptNo(Integer deptNo) {
91         this.deptNo = deptNo;
92     }
93
94     /**
95      * 一个类，不写构造方法，默认就是无参构造方法
96      */
97     public Emp() {
98     }
99
100    @Override
101    public String toString() {
102        final StringBuilder sb = new StringBuilder("Emp{");
103        sb.append("empNo=").append(empNo);
104        sb.append(", eName=").append(eName).append('\n');
105        sb.append(", job=").append(job).append('\n');
106        sb.append(", mgr=").append(mgr);
107        sb.append(", hireDate=").append(hireDate);
108        sb.append(", sal=").append(sal);
109        sb.append(", comm=").append(comm);
110        sb.append(", deptNo=").append(deptNo);
111        sb.append('}');
112        return sb.toString();
113    }
114 }

```

dao层的接口

```

1  package cn.kgc.dao;
2
3  import cn.kgc.domain.Emp;
4
5  import java.util.List;
6
7  /**
8   * @Author: lc
9   * @Date: 2022/5/13

```

```

10  * @Description: dao层都是数据库操作：新增、删除、修改、查询
11  * @Version: 1.0
12  */
13  public interface EmpDao {
14      //抽象方法
15      /**
16       * 查询功能：方法一定有返回值
17       * 根据主键查询：实体类对象
18       * 多条件或无条件查询：集合，集合泛型domain中表对应实体类
19       * 聚合函数查询：Object
20       * @param eName 查询条件：员工姓名，支持模糊查询
21       * @param job 查询条件：岗位 不支持模糊查询
22       */
23      List<Emp> selectBy(String eName,String job);
24  }

```

dao层的实现类

```

1  package cn.kgc.dao.impl;
2
3  import cn.kgc.dao.EmpDao;
4  import cn.kgc.domain.Emp;
5  import cn.kgc.util.JDBCUtil;
6  import org.apache.commons.dbutils.QueryRunner;
7  import org.apache.commons.dbutils.handlers.BeanListHandler;
8
9  import java.sql.SQLException;
10 import java.util.ArrayList;
11 import java.util.List;
12 import java.util.Objects;
13
14 /**
15  * @Author: lc
16  * @Date: 2022/5/13
17  * @Description: dao的实现类包，写代码有什么套路？
18  * 1.创建一个QueryRunner对象，并将你的数据库连接池交给它
19  * 2.定义sql语句，
20  *     设置参数前，需要对形参是否null进行非空校验
21  * 3.使用QueryRunner中的方法：
22  *     update():DML
23  *     query():DQL
24  * @Version: 1.0
25  */
26 public class EmpDaoImpl implements EmpDao {
27     //QueryRunner会被多个方法使用，建议定义全局变量
28     private QueryRunner qr=new QueryRunner(JDBCUtil.datasource);
29     @Override
30     public List<Emp> selectBy(String eName, String job) {
31         //where ename like '%_%'
32         //多条件模糊查询，要求：给的查询条件是NULL的情况，就不要带着条件查询
33         //1=1解决逻辑运算符拼接问题
34         StringBuilder sql=new StringBuilder("select * from emp where 1=1 ");
35         try {
36             //selectBy(null,"销售员")
37             //需要对sql中?的实参值进行非空判断 排除空字符串""
38             //引入集合，保存参数的对应关系
39             List params=new ArrayList();

```

```

40         if(!Objects.isNull(eName)&&!eName.isEmpty()){
41             //当eName条件不是nul的情况下，根据员工姓名模糊查询
42             sql.append(" and ename like ? ");
43             params.add("%"+eName+"%");
44         }
45
46         if(!Objects.isNull(job)&&!job.isEmpty()){
47             sql.append(" and job = ?");
48             params.add(job);
49         }
50         //qr.query(sql语句,resultset结果集处理方式, Object... paramsql中有几个? 就应该给几个实参)
51         /**
52          * sql: select 结果分为三种情况
53          * 1. 查询结果是多行多列 BeanListHandler<表对应的java类型>
54          * 2. 查询结果是单行多列 BeanHandler<表对应的java类型>
55          * 3. 查询结果是单行单列，一般是聚合函数 select count() from sum()
56          max() min() avg() ScalarHandler==>Object
57          */
58         return qr.query(sql.toString(),new BeanListHandler<>
59             (Emp.class),params.toArray());
60     } catch (SQLException e) {
61         throw new RuntimeException(e);
62     }

```

service层的接口

```

1 package cn.kgc.service;
2
3 import cn.kgc.domain.Emp;
4
5 import java.util.List;
6
7 /**
8  * @Author: lc
9  * @Date: 2022/5/13
10  * @Description: service写代码:
11  * DAO的方法名copy过来，修改方法名: 业务逻辑单词
12  * service依赖dao实现数据处理
13  * @Version: 1.0
14  */
15 public interface EmpService {
16     /**
17      * 根据员工姓名和岗位获取员工信息
18      * @param eName
19      * @param job
20      * @return
21      */
22     List<Emp> getBy(String eName, String job);
23 }

```

service层的实现类

```
1 package cn.kgc.service.impl;
2
3 import cn.kgc.dao.EmpDao;
4 import cn.kgc.dao.impl.EmpDaoImpl;
5 import cn.kgc.domain.Emp;
6 import cn.kgc.service.EmpService;
7
8 import java.util.List;
9
10 /**
11  * @Author: lc
12  * @Date: 2022/5/13
13  * @Description: service的实现类只写三行代码:
14  * 1. 创建dao对象, 根据接口第一个单词去找
15  * EmpDao
16  *
17  * 2. 调用dao层方法
18  *
19  * 3. 处理返回值
20  * @Version: 1.0
21  */
22 public class EmpServiceImpl implements EmpService {
23     private EmpDao empDao=new EmpDaoImpl();
24     @Override
25     public List<Emp> getBy(String eName, String job) {
26         //List<Emp> emps = empDao.selectBy(eName, job);
27         return empDao.selectBy(eName, job);
28     }
29 }
```

UI层的测试类

```
1 package cn.kgc.test;
2
3 import cn.kgc.domain.Emp;
4 import cn.kgc.service.impl.EmpServiceImpl;
5
6 import java.util.List;
7 import java.util.Scanner;
8
9 /**
10  * @Author: lc
11  * @Date: 2022/5/13
12  * @Description: cn.kgc.test
13  * @Version: 1.0
14  */
15 public class EmpTester {
16     public static void main(String[] args) {
17         Scanner input = new Scanner(System.in);
18         //1. 获取数据, 交给业务逻辑层, 由业务逻辑层提交给dao, 实现数据库交互
19         System.out.print("员工姓名: ");
20         String ename = input.next();
21         System.out.print("岗位: ");
22         String job = input.next();
```

```
23
24      //找一个对象，做一件事
25      List<Emp> emps = new EmpServiceImpl().getBy(ename, job);
26
27      //2.获取业务逻辑层处理结果，使用sout显示
28      System.out.println(emps);
29
30  }
31 }
```

2 安装tomcat服务器

2-1 注意事项
























- 1
- 2
- 3
1. tomcat解压目录不能有中文
2. tomcat的启动必须保证环境变量中有java_home环境变量
3. tomcat版本必须与jdk匹配

Servlet Spec	JSP Spec	EL Spec	WebSocket Spec	Authentication (JASPIC) Spec	Apache Tomcat Version	Latest Released Version	Supported Java Versions
6.0	3.1	5.0	2.1	3.0	10.1.x	10.1.0-M14 (alpha)	11 and later
5.0	3.0	4.0	2.0	2.0	10.0.x	10.0.20	8 and later
4.0	2.3	3.0	1.1	1.1	9.0.x	9.0.62	8 and later
3.1	2.3	3.0	1.1	1.1	8.5.x	8.5.78	7 and later
3.1	2.3	3.0	1.1	N/A	8.0.x (superseded)	8.0.53 (superseded)	7 and later
3.0	2.2	2.2	1.1	N/A	7.0.x (archived)	7.0.109 (archived)	6 and later (7 and later for WebSocket)
2.5	2.1	2.1	N/A	N/A	6.0.x (archived)	6.0.53 (archived)	5 and later
2.4	2.0	N/A	N/A	N/A	5.5.x (archived)	5.5.36 (archived)	1.4 and later
2.3	1.2	N/A	N/A	N/A	4.1.x (archived)	4.1.40 (archived)	1.3 and later
2.2	1.1	N/A	N/A	N/A	3.3.x (archived)	3.3.2 (archived)	1.1 and later

2-2 tomcat安装目录

backup	2018/8/31 15:10	文件夹	
bin	2020/11/3 9:33	文件夹	
conf	2019/3/15 17:08	文件夹	
lib	2013/7/2 8:59	文件夹	
logs	2022/5/13 16:45	文件夹	
temp	2021/12/3 14:26	文件夹	
webapps	2020/7/16 21:49	文件夹	
work	2018/8/31 15:19	文件夹	
LICENSE	2013/7/2 8:59	文件	57 KB
NOTICE	2013/7/2 8:59	文件	2 KB
RELEASE-NOTES	2013/7/2 8:59	文件	9 KB
RUNNING.txt	2013/7/2 8:59	文本文档	17 KB

bin目录：启动和关闭tomcat的可执行命令

名称	修改日期	类型	大小
 catalina.bat	2013/7/2 8:59	Windows 批处理...	13 K
 catalina.sh	2013/7/2 8:59	SH 文件	20 K
 catalina-tasks.xml	2013/7/2 8:59	XML 文件	3 K
 commons-daemon.jar	2013/7/2 8:59	Executable Jar File	24 K
 commons-daemon-native.tar.gz	2013/7/2 8:59	WinRAR 压缩文件	201 K
 configtest.bat	2013/7/2 8:59	Windows 批处理...	3 K
 configtest.sh	2013/7/2 8:59	SH 文件	2 K
 cpappend.bat	2013/7/2 8:59	Windows 批处理...	2 K
 daemon.sh	2013/7/2 8:59	SH 文件	8 K
 digest.bat	2013/7/2 8:59	Windows 批处理...	3 K
 digest.sh	2013/7/2 8:59	SH 文件	2 K
 setclasspath.bat	2013/7/2 8:59	Windows 批处理...	4 K
 setclasspath.sh	2013/7/2 8:59	SH 文件	4 K
 shutdown.bat 关闭服务器	2013/7/2 8:59	Windows 批处理...	3 K
 shutdown.sh	2013/7/2 8:59	SH 文件	2 K
 startup.bat 启动服务器	2013/7/2 8:59	Windows 批处理...	3 K
 startup.sh	2013/7/2 8:59	SH 文件	2 K
 tomcat-juli.jar	2013/7/2 8:59	Executable Jar File	38 K
 tomcat-native.tar.gz	2013/7/2 8:59	WinRAR 压缩文件	282 K
 tool-wrapper.bat	2013/7/2 8:59	Windows 批处理...	5 K
 tool-wrapper.sh	2013/7/2 8:59	SH 文件	5 K
 version.bat	2013/7/2 8:59	Windows 批处理...	3 K
 version.sh	2013/7/2 8:59	SH 文件	2 K

conf: 服务器配置文件所在目录

修改服务器的端口号:server.xml

```
1 <Connector connectionTimeout="20000" port="8080" protocol="HTTP/1.1"
  redirectPort="8443"/>
```

获取服务器登录账号: tomcat-users.xml

```
1 <user username="tomcat" password="tomcat" roles="tomcat,manager-gui"/>
```

webapp: 服务器上发布项目的目录

名称	修改日期	类型
docs	2013/7/2 8:59	文件夹
examples	2013/7/2 8:59	文件夹
hello 自己手动创建的静态项目	2022/5/13 17:35	文件夹
hello2 自己手动创建的动态项目	2022/5/13 17:47	文件夹
host-manager	2013/7/2 8:59	文件夹
manager	2013/7/2 8:59	文件夹
review	2018/9/6 11:28	文件夹
ROOT	2013/7/2 8:59	文件夹

Tomcat创建项目成功以后，运行方式

1. 启动浏览器
2. 输入访问的网址：
- 网址的写法遵循：
- 协议名://域名:端口/路径，例如：`http://www.baidu.com:80/index.html`
- 例如：
- `http://localhost:8080/发布项目的名称/要查看的项目资源名称`

补充：Tomcat服务器上静态项目和动态项目的区别

静态项目：网页不能跟数据库交互，比如常见的 `html+css+js+jquery`

- 1 创建方式：
- 1 在webapps目录下创建一个目录（命名必须不包含中文和空格），这个目录称之为项目目录；
- 2 在项目目录下创建一个html文件；

动态项目：能与数据库交互的项目，都可以理解为动态项目，比如（java的jsp C#的aspx）

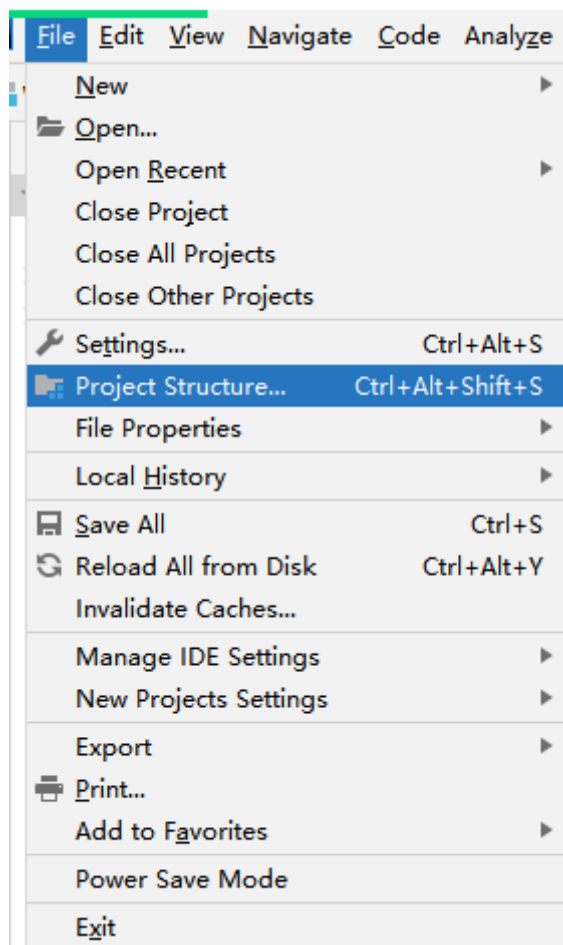
- 1 创建方式：
- 1 在webpass目录下创建一个项目目录；
- 2 在项目目录下创建如下内容：
- WEB-INF目录；
- 在WEB-INF目录下创建web.xml文件
- 3 创建静态或动态页面

3 IDEA创建并运行WEB项目

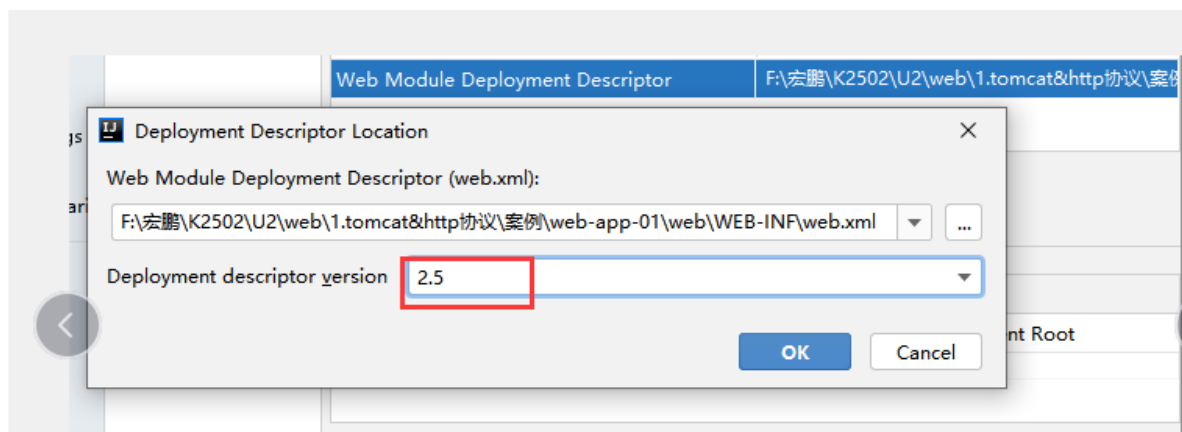
3-1 创建JavaWeb项目的步骤

创建一个普通java项目

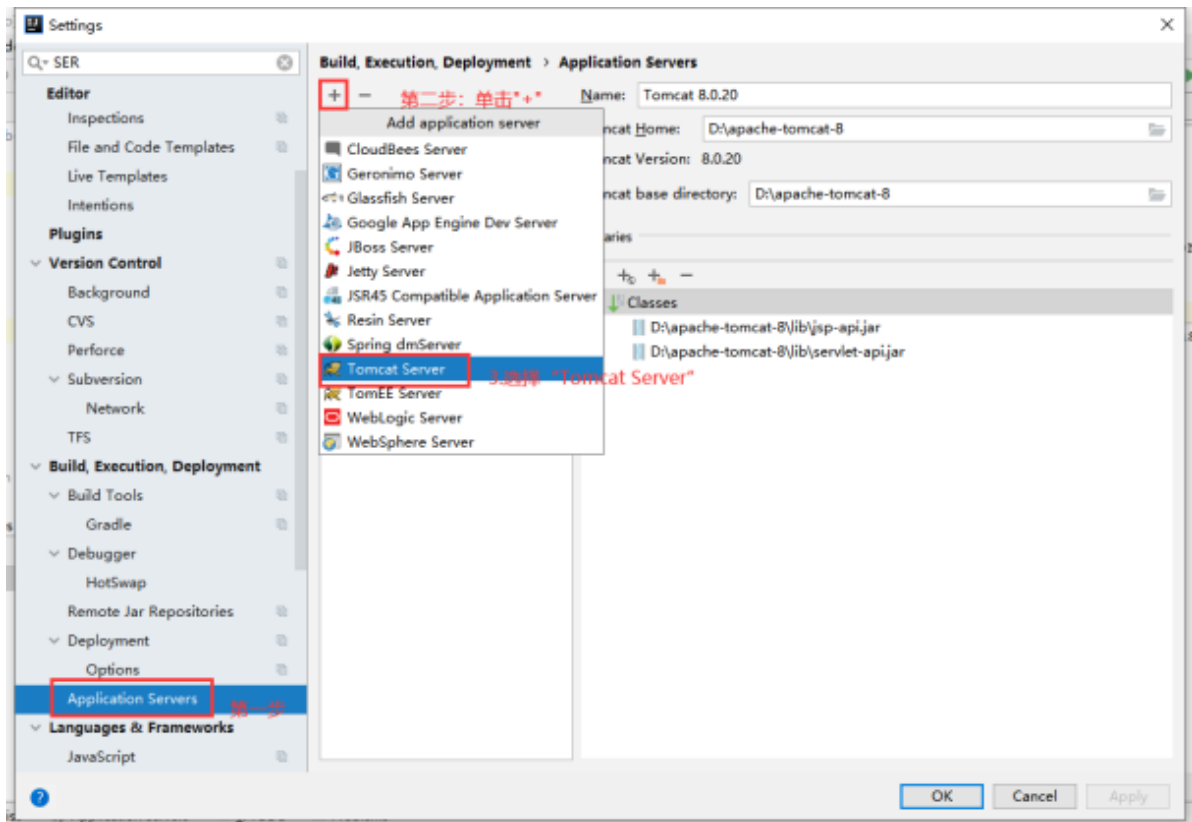
添加web模块



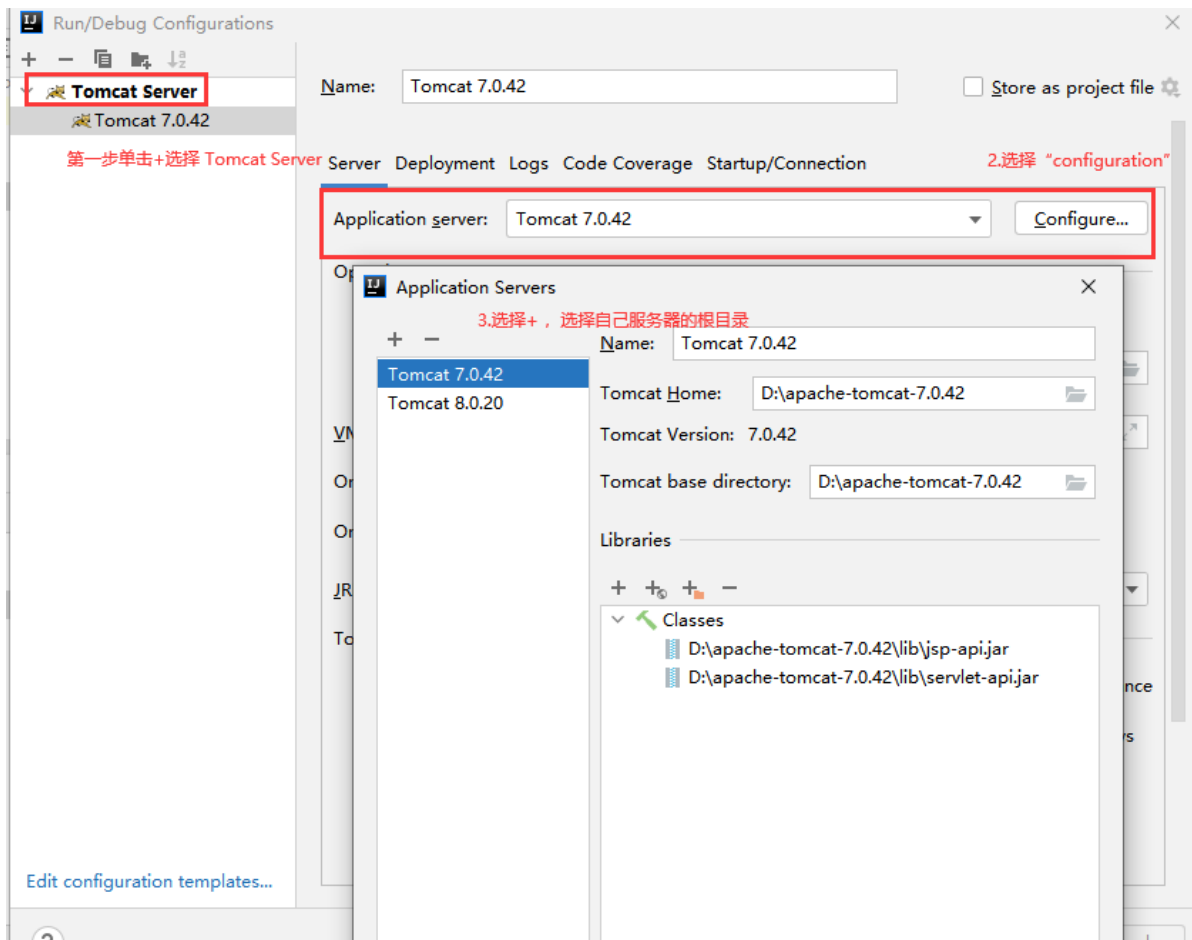
设置web版本

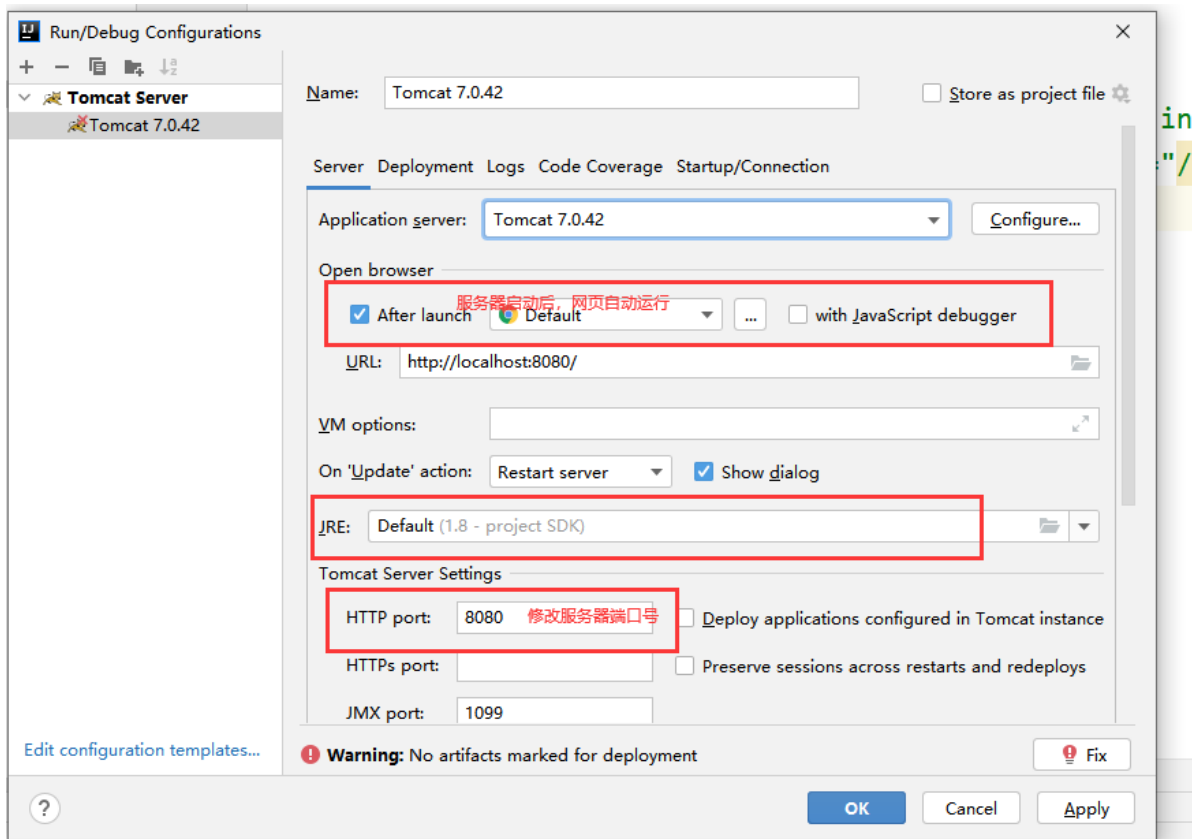


3-2 Idea配置服务器的步骤

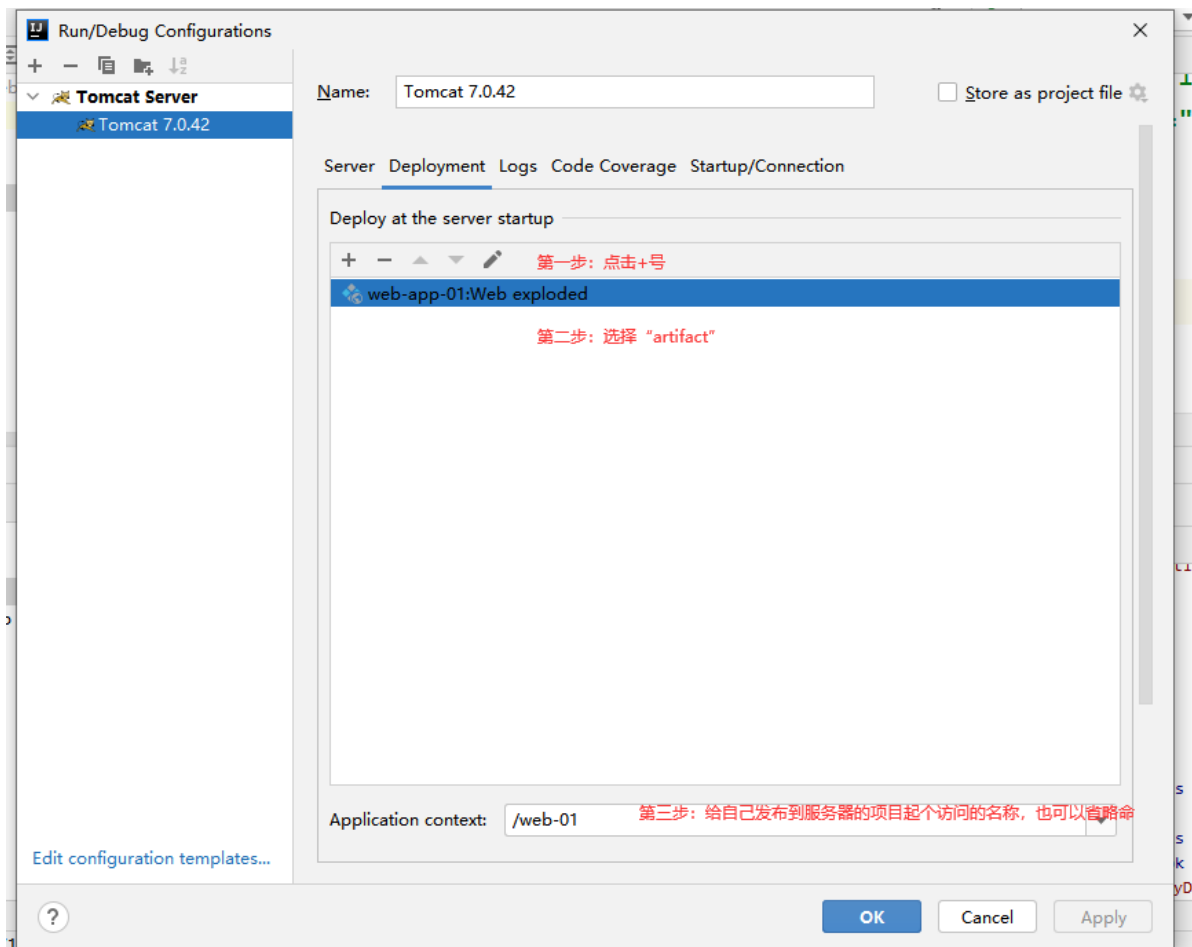


或

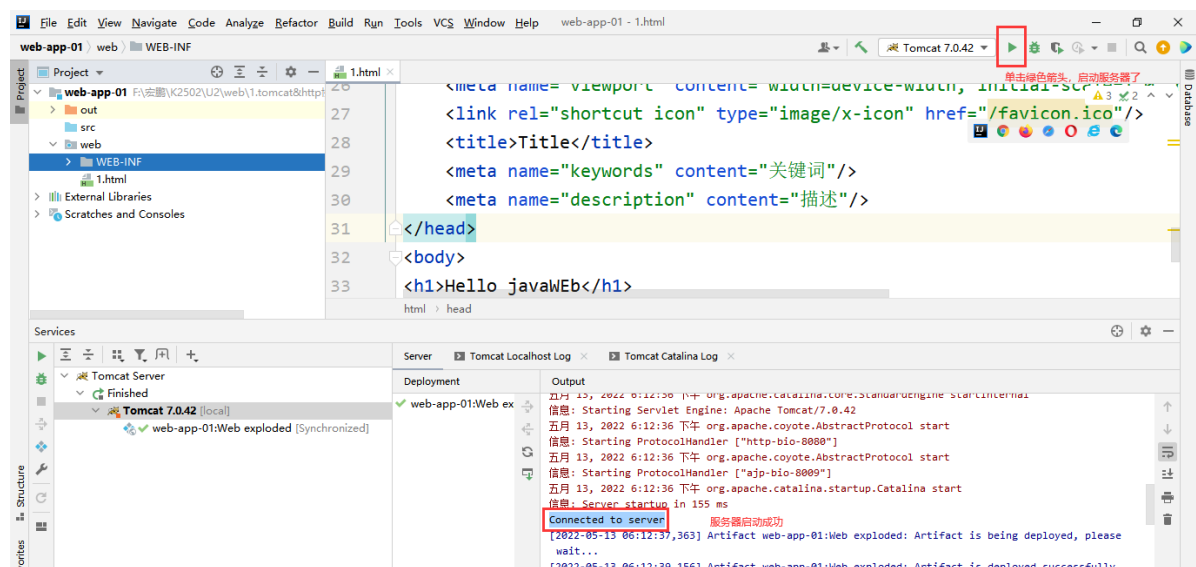




3-3 Idea发布项目的步骤



3-4 Idea启动服务器的步骤



3-5 Idea项目的访问方式



课程总结

1 完成三层架构

2 idea集成tomcat

3 JavaWeb项目的创建及部署

预习

Servlet使用

HTTP协议