

IntelliJ IDEA 中建立 SSH 应用步骤

作者：许剑航 日期：2016.2.14

测试环境 IntelliJ IDEA 15.0.3, Tomcat 8.0.30, jdk1.8.0_72, MySQL 5.7。

IDEA 工程中所用框架 Spring 4.2.4, Struts 2-2.3.20.1, Hibernate 4.2.2

还需要手动添加 struts2-spring-plugin-2.3.20.1.jar, spring-web-4.2.4.RELEASE.jar 以及 mysql-connector-java-5.1.35-bin.jar

所需文件可以到 <http://search.maven.org/> 中搜索, 也可以在 IDEA 中通过菜单从 Maven 中搜索下载

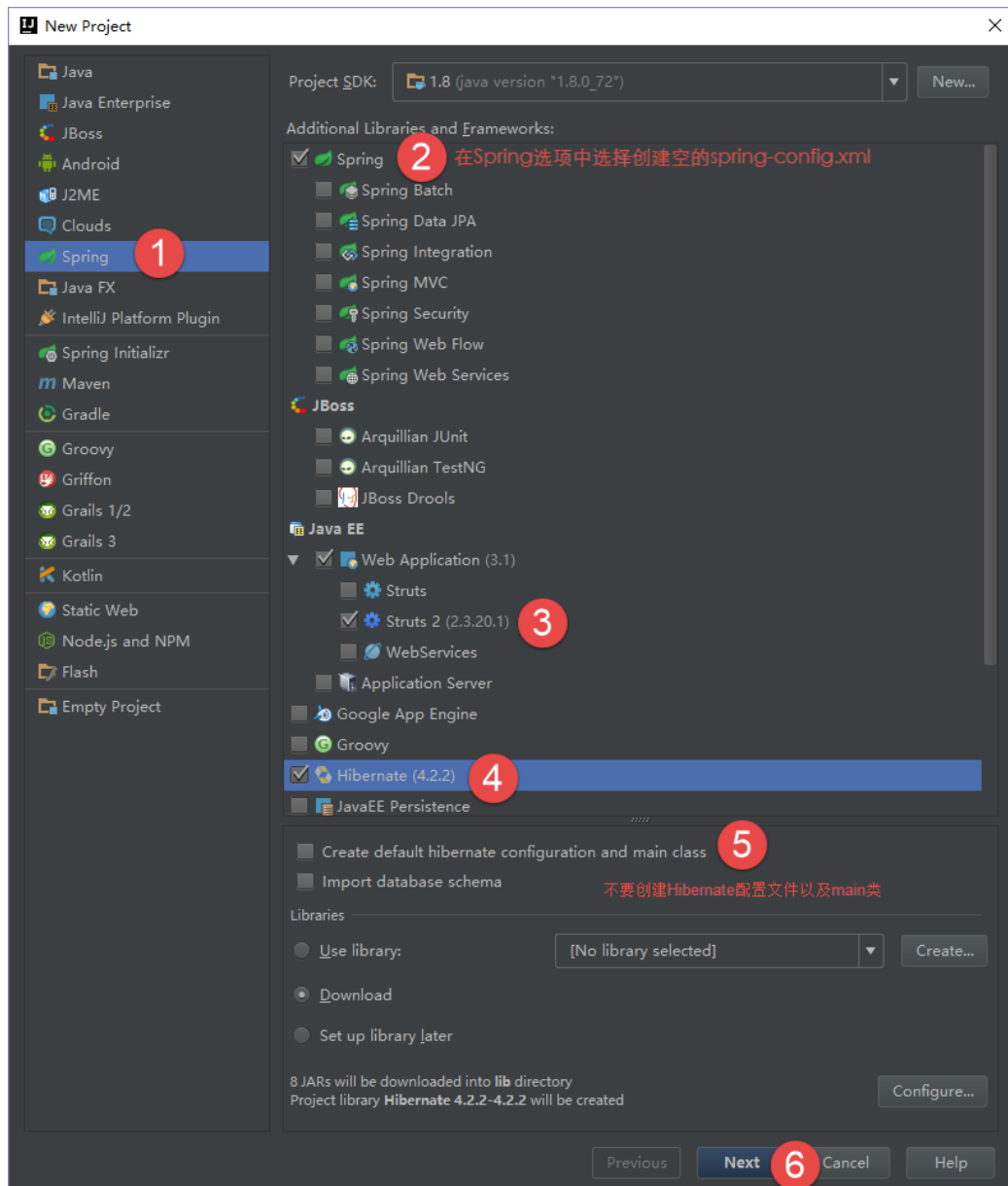
The first screenshot shows the search results for 'mysql-connector-java' on the Maven Central Repository. The search bar contains 'mysql-connector-java' with a red circle '1' next to it. The results table shows three entries, with the second entry 'mysql-connector-java' version '5.1.34_1' highlighted by a red circle '2' and a red arrow pointing to it.

Groupid	Artifactid	Latest Version	Updated	Download
mysql	mysql-connector-java	5.1.38 all (50)	02-Dec-2015	pom jar sources.jar
org.wildfly.plugins	mysql-connector-java	5.1.34_1 all (1)	13-Jan-2015	pom jar javadoc.jar sources.jar
org.ops4j.pax.tycho	org.ops4j.pax.tycho.mysql.connector.java	5.1.22.1	31-Dec-2012	pom jar

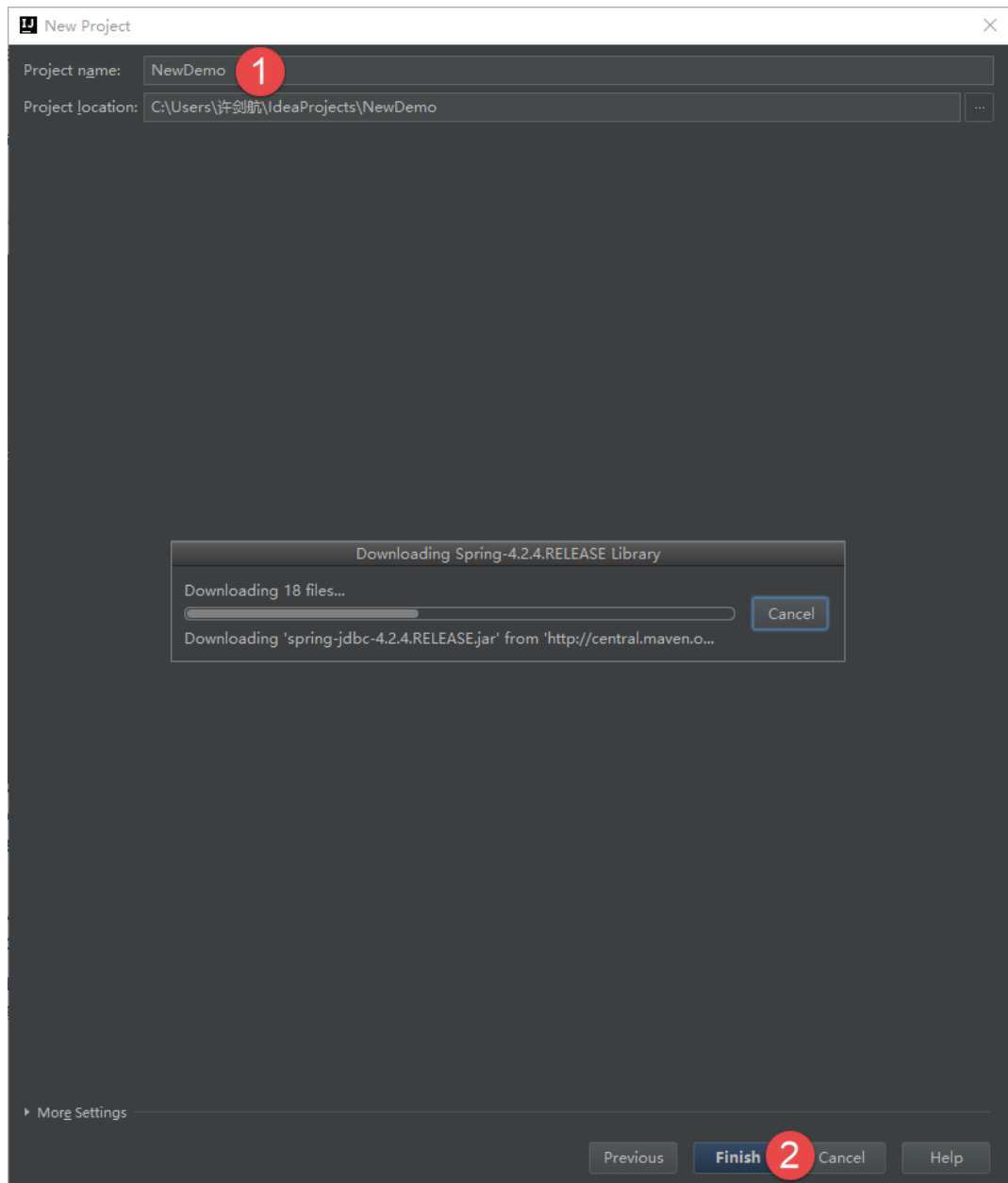
The second screenshot shows the search results for 'g:"mysql" AND a:"mysql-connector-java"' on the Maven Central Repository. The search bar contains 'g:"mysql" AND a:"mysql-connector-java"' with a red circle '1' next to it. The results table shows 20 entries, with the entry 'mysql-connector-java' version '5.1.35' highlighted by a red circle '2' and a red arrow pointing to it.

Groupid	Artifactid	Version	Updated	Download
mysql	mysql-connector-java	5.1.38	02-Dec-2015	pom jar sources.jar
mysql	mysql-connector-java	5.1.37	06-Oct-2015	pom jar sources.jar
mysql	mysql-connector-java	5.1.36	20-Jun-2015	pom jar sources.jar
mysql	mysql-connector-java	5.1.35	08-May-2015	pom jar sources.jar
mysql	mysql-connector-java	5.1.9	08-May-2015	pom jar
mysql	mysql-connector-java	5.1.8	08-May-2015	pom jar
mysql	mysql-connector-java	5.1.6	08-May-2015	pom jar
mysql	mysql-connector-java	5.1.5	08-May-2015	pom jar
mysql	mysql-connector-java	5.1.4	08-May-2015	pom jar
mysql	mysql-connector-java	5.1.34	08-May-2015	pom jar sources.jar
mysql	mysql-connector-java	5.1.33	08-May-2015	pom jar sources.jar
mysql	mysql-connector-java	5.1.32	08-May-2015	pom jar sources.jar
mysql	mysql-connector-java	5.1.31	08-May-2015	pom jar sources.jar
mysql	mysql-connector-java	5.1.30	08-May-2015	pom jar sources.jar
mysql	mysql-connector-java	5.1.29	08-May-2015	pom jar sources.jar
mysql	mysql-connector-java	5.1.28	08-May-2015	pom jar sources.jar
mysql	mysql-connector-java	5.1.3	08-May-2015	pom jar
mysql	mysql-connector-java	5.1.27	08-May-2015	pom jar sources.jar
mysql	mysql-connector-java	5.1.26	08-May-2015	pom jar sources.jar
mysql	mysql-connector-java	5.1.25	08-May-2015	pom jar sources.jar

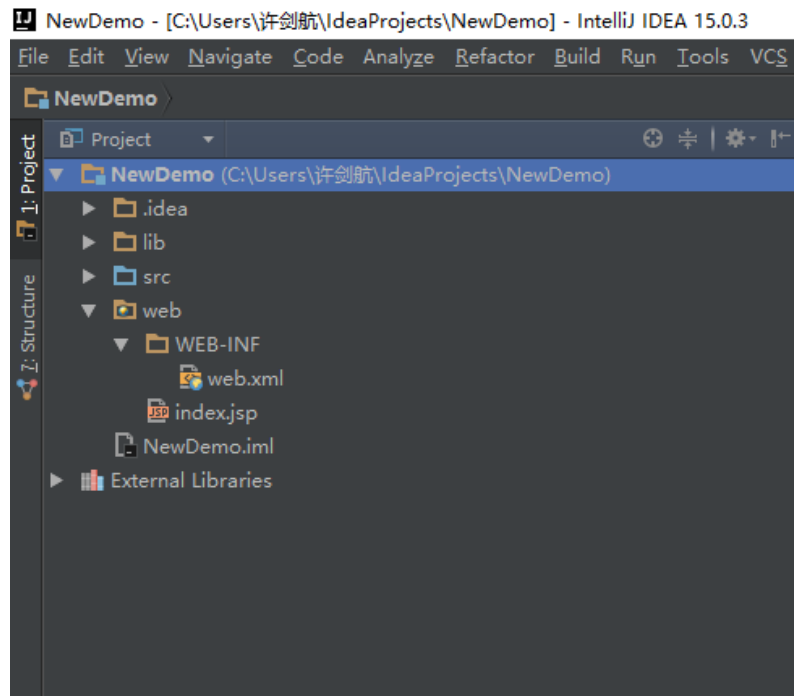
1. 打开 IDEA, 创建新的工程 (Create New Project)
2. 选择如下的配置, 勾选 Spring, Struts 2 以及 Hibernate, 在选中 Spring 选项时选择自动创建 Spring 配置文件 spring-config.xml, 选择 Hibernate 时注意由于其配置都将建立在 Spring 的配置文件中, 所以不要选择 Create default hibernate configuration and main class。



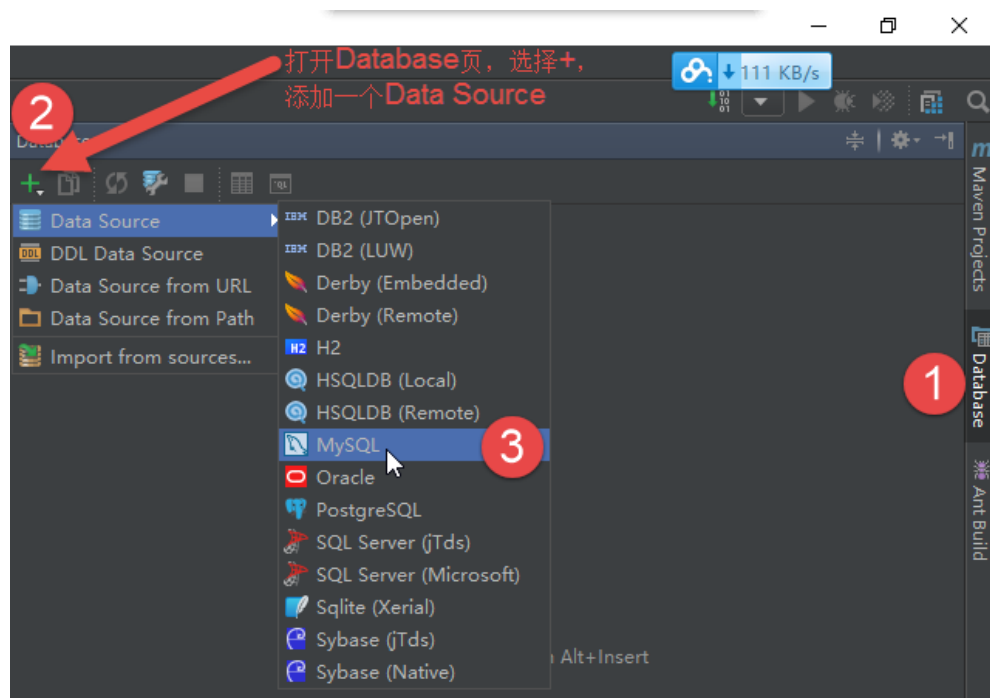
3. 选择 Next 后，给工程建立一个名称，例如 SSHDemo，再点击 Finish 按钮，软件自动下载合适的 Java 包并创建文件目录结构。



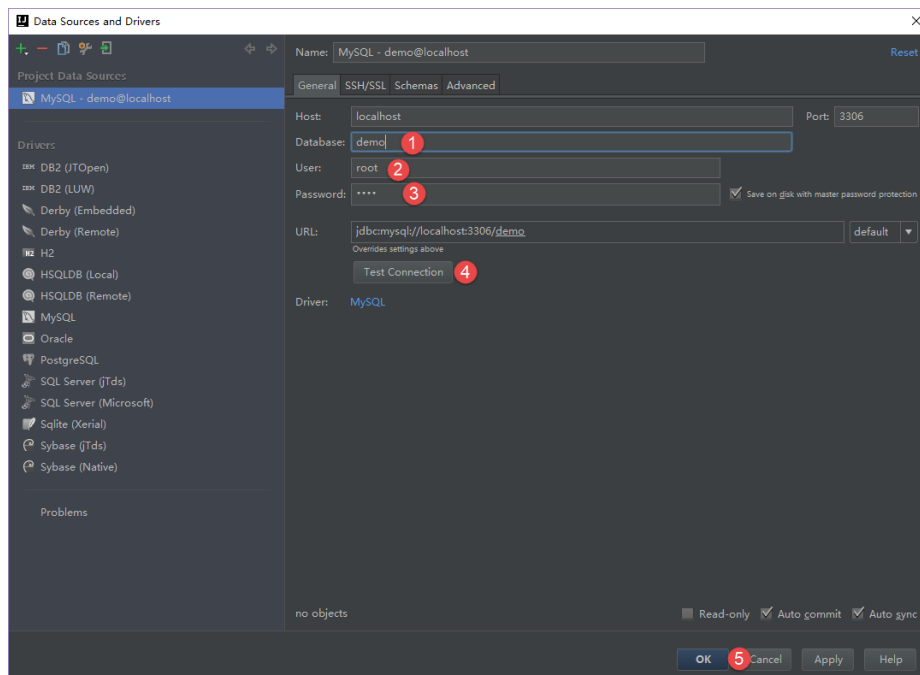
4. 创建好的目录如下，可以打开软件的 Project 页签看到



5. 打开 Database 页签为工程建立一个新的 DataSource, 选中左上角的+, 再选 Data Source, 然后选择 MySQL

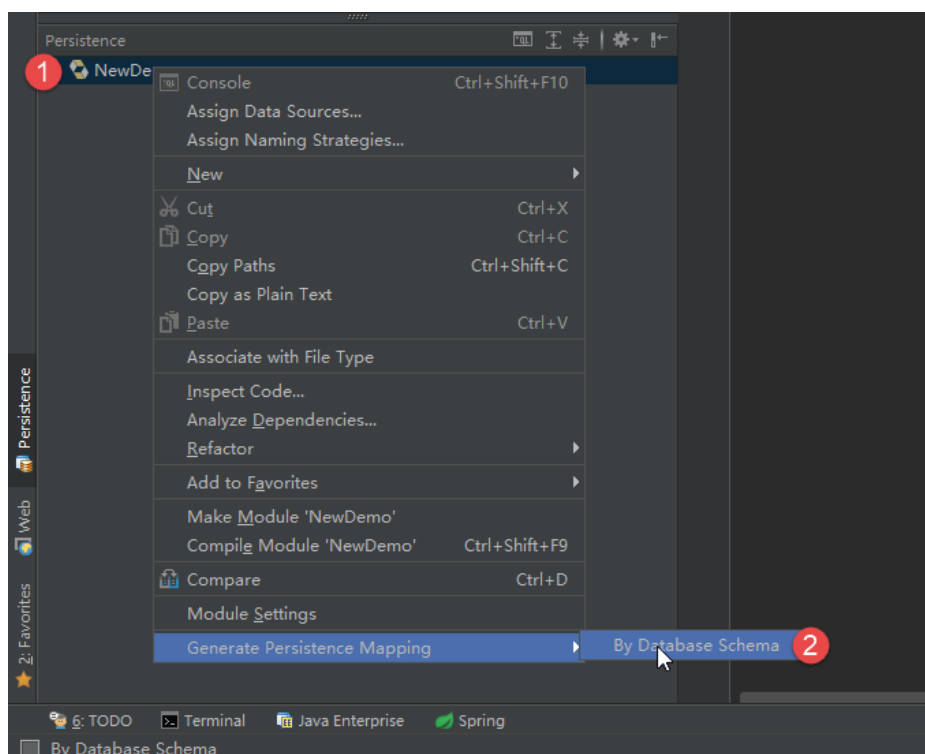


选中数据库类型后, 会出现配置界面

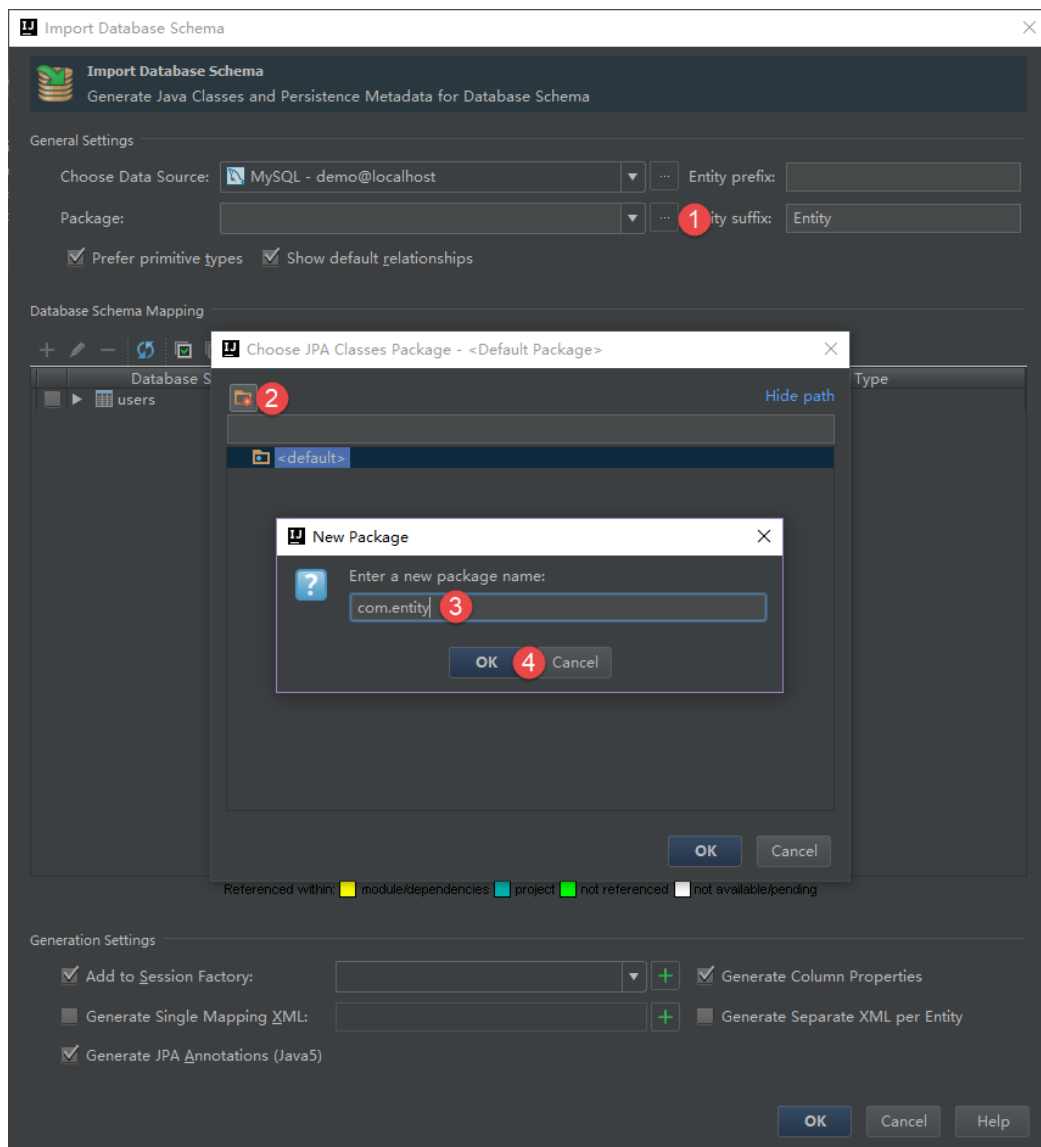


设置好后按 OK

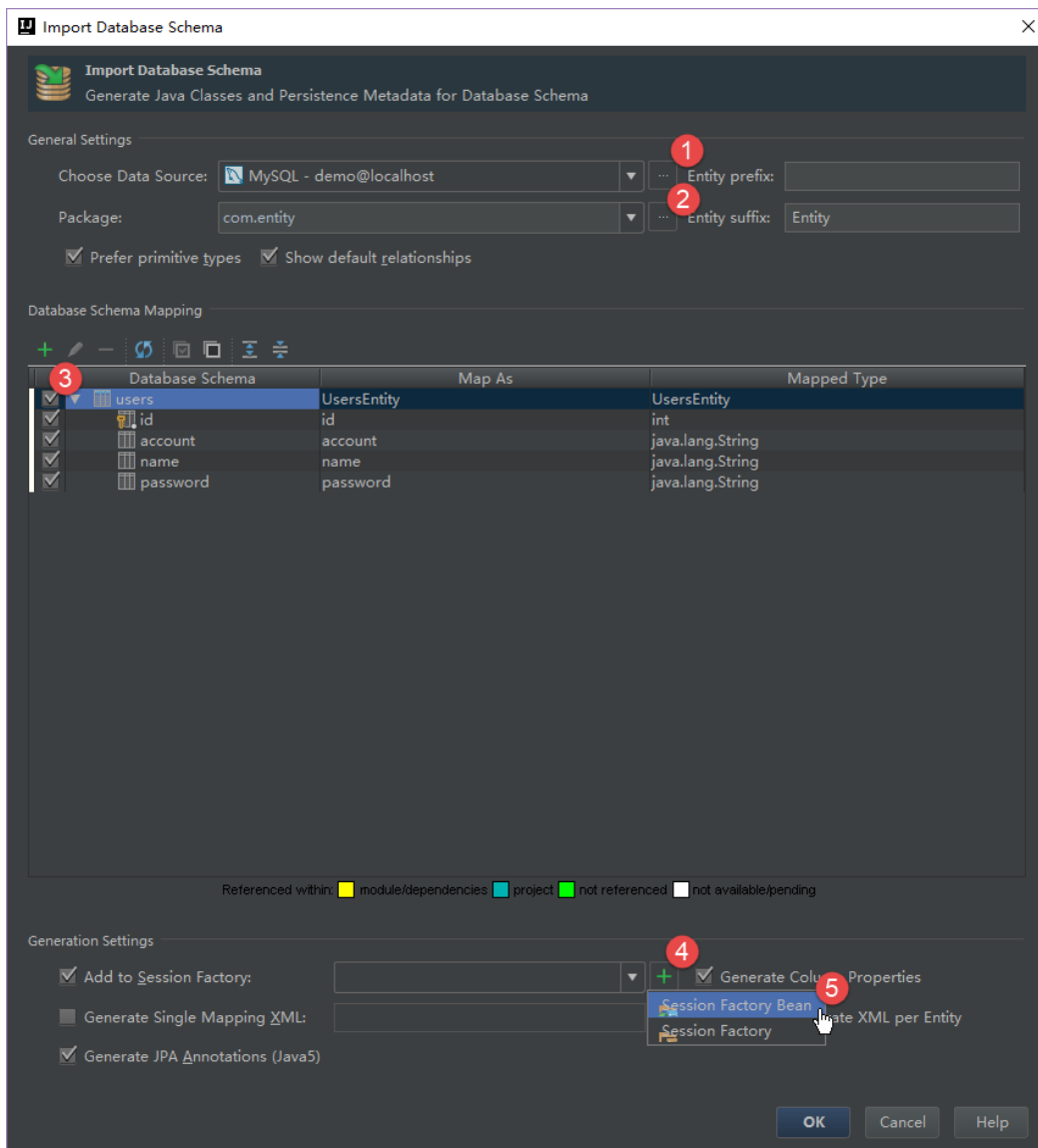
6. 点击 Persistence 页签，在上面点击右键，从菜单中选择 Generate Persistence Mapping->By Database Schema



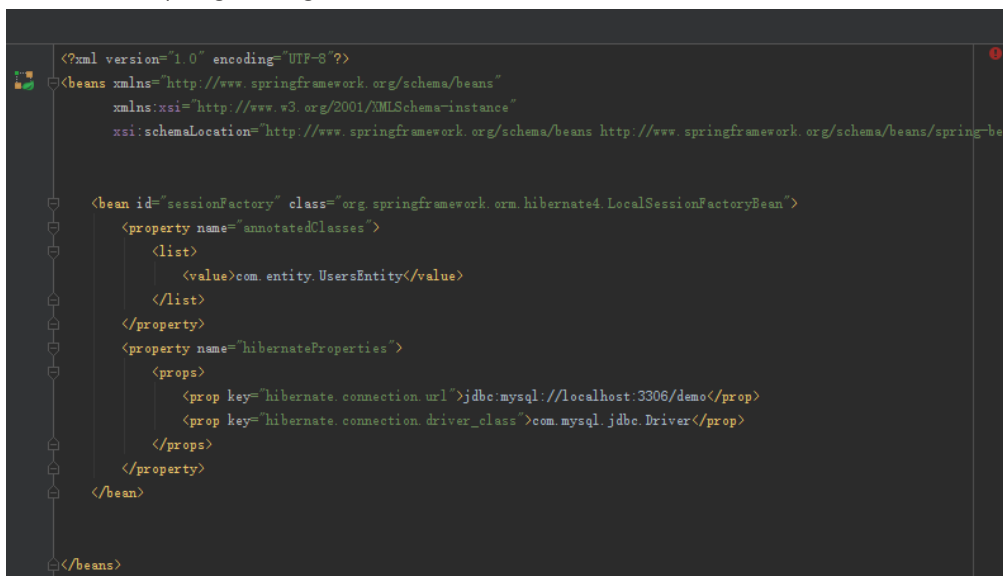
在接下来的画面中，给 Data Source 选择刚刚配置的名称，点击 Package 的...按钮新建一个存放 Persistence Mapping 的包名称，本例中输入 com.entity。



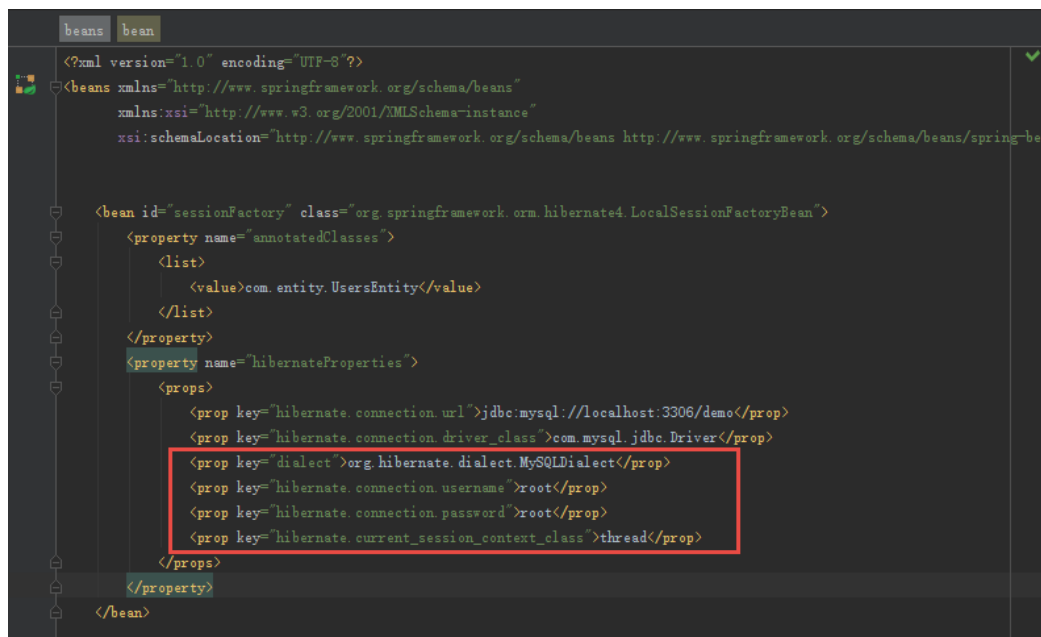
勾选要映射的 Table，本例中选中的是 Users，软件自动给出 Map 的类名为 UsersEntity，该名字可以更改，建立好的 Class 文件就存放到名为 com.entity 的包中，实际从文件结构上看就是在 src 目录下多了一个 com 目录，com 目录中又多了一个 entity 目录。然后在 Add to Session Factory 选项中选择+按钮来建立一个新的 Session Factory，因为 Spring 接管了 Hibernate，所以需要建立的是 Session Factory Bean



点击后，在 spring-config.xml 中会自动增加如下内容



由于系统不会自动添加上数据库用户名密码等信息，所以还需要增加一些配置



自动产生的 Table 的 Map 文件名为 UsersEntity.java，该文件存放到 src/com/entity 目录中的里，由于前面在创建 Persistence Mapping 时选择的是 Generate JPA Annotations(Java5)，没有勾选 Generate Separate XML per Entity，所以不会生成单独的 Entity 的 Map 配置文件，而是采用基于注解的 Annotation 技术来声明映射关系。

```
package com.entity;
import javax.persistence.*;

@Entity
@Table(name = "users", schema = "demo", catalog = "")
public class UsersEntity {
    private int id;
    private String account;
    private String name;
    private String password;

    @Id
    @Column(name = "id", nullable = false)
    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    @Basic
    @Column(name = "account", nullable = true, length = 20)
```



```

public String getAccount() {
    return account;
}

public void setAccount(String account) {
    this.account = account;
}

@Basic
@Column(name = "name", nullable = true, length = 45)
public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

@Basic
@Column(name = "password", nullable = true, length = 20)
public String getPassword() {
    return password;
}

public void setPassword(String password) {
    this.password = password;
}

@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;

    UsersEntity that = (UsersEntity) o;

    if (id != that.id) return false;
    if (account != null ? !account.equals(that.account) : that.account != null) return
false;
    if (name != null ? !name.equals(that.name) : that.name != null) return false;
    if (password != null ? !password.equals(that.password) : that.password != null)
return false;

    return true;
}

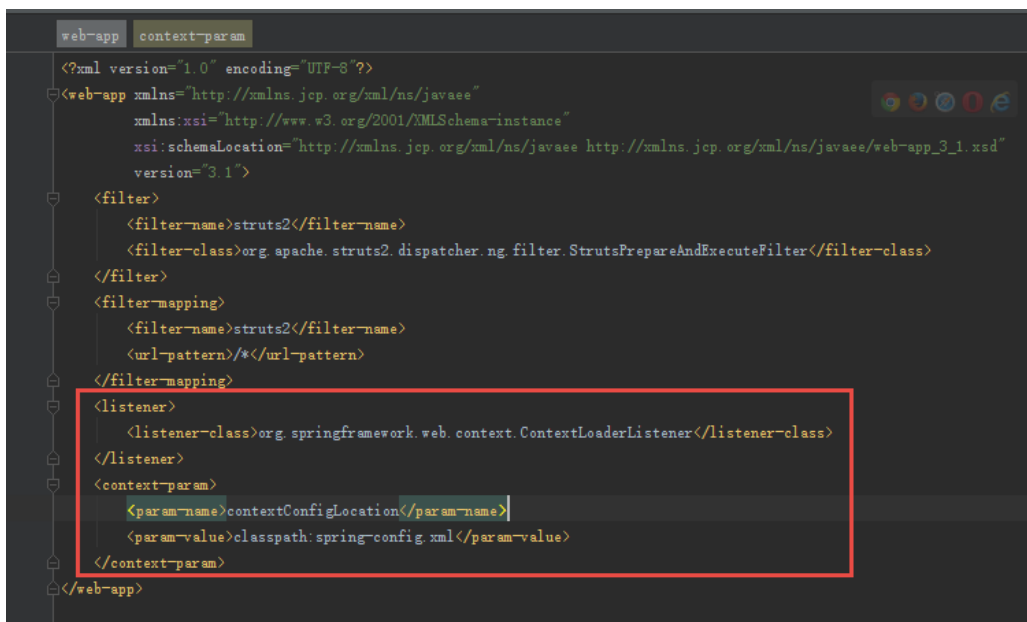
```

```

@Override
public int hashCode() {
    int result = id;
    result = 31 * result + (account != null ? account.hashCode() : 0);
    result = 31 * result + (name != null ? name.hashCode() : 0);
    result = 31 * result + (password != null ? password.hashCode() : 0);
    return result;
}
}

```

7. 接下来，我们打开 web.xml，增加如下配置，让 Web 应用程序启动时自动装载 Spring 容器，如下所示。

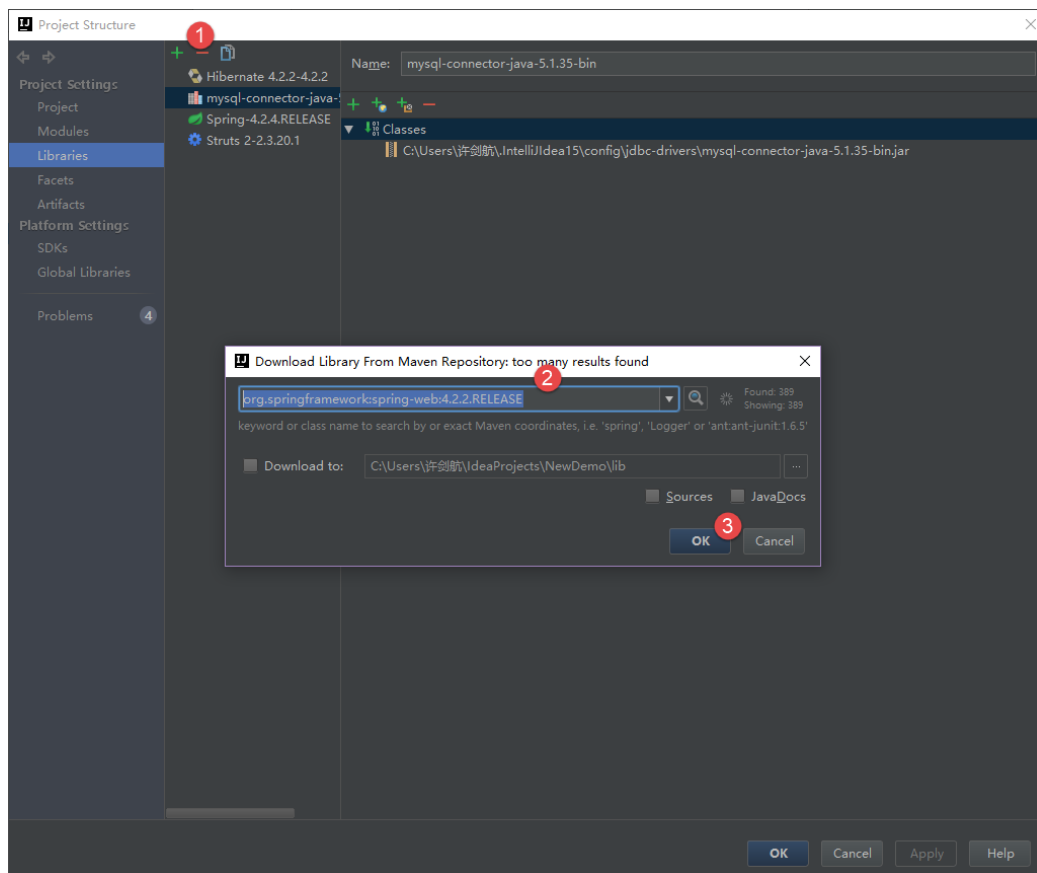


```

<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
    version="3.1">
    <filter>
        <filter-name>struts2</filter-name>
        <filter-class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</filter-class>
    </filter>
    <filter-mapping>
        <filter-name>struts2</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
    <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:spring-config.xml</param-value>
    </context-param>
</web-app>

```

如果没有在 Project 中手工添加 spring-web-x.x.x.RELEASE.jar，那么一开始会报告找不到 org.springframework.web.context.ContextLoaderListener 错误，这个接口存在于 spring-web-x.x.x.RELEASE.jar 包中，需要手工添加到系统，方法如下，打开 Project Structure，选中左侧的 Libraries，再选中+按钮，选择 From Maven，出现了 Maven 查找对话框，输入 spring-web 进行搜索，在提示的搜索结果中找到合适的版本去下载，下载的文件很多，可以删除掉其他的，只保留 spring-web-x.x.x.RELEASE.jar



后面同样还有一个类似的重要文件要添加进工程，包名称是 struts2-spring-plugin-2.3.20.1.jar，请参照上面的方法下载

8. 接下来建立数据访问层和业务逻辑层类文件

数据访问层包括两个文件

IUserDAO.java

```
package com. dao;

import com. entity. UsersEntity;
import java. util. List;

public interface IUserDAO {
    public List search(UsersEntity condition);
}
```

UserDAO.java

```
package com. dao;

import com. entity. UsersEntity;
import org. hibernate. Criteria;
import org. hibernate. Session;
import org. hibernate. SessionFactory;
import org. hibernate. Transaction;
import org. hibernate. criterion. Example;
```

```

import java.util.List;

public class UserDao implements IUserDAO {
    SessionFactory sessionFactory;

    public void setSessionFactory(SessionFactory sessionFactory) {
        this.sessionFactory = sessionFactory;
    }

    @Override
    public List search(UsersEntity condition) {
        List list = null;
        Session session = sessionFactory.getCurrentSession();

        Transaction tx = session.beginTransaction();
        try {
            Criteria c = session.createCriteria(UsersEntity.class);
            Example example = Example.create(condition);
            c.add(example);
            list = c.list();
            tx.commit();
        } catch (Exception e) {
            tx.rollback();
        }
        return list;
    }
}

```

业务逻辑层也包括两个文件

IUserBiz.java

```

package com.biz;

import com.entity.UsersEntity;
import java.util.List;

public interface IUserBiz {
    public List login(UsersEntity condition);
}

```

UserBiz.java

```

package com.biz;

import com.dao.IUserDAO;

```

```

import com.entity.UsersEntity;
import java.util.List;

public class UserBiz implements IUserBiz{
    IUserDAO userDAO;

    public void setUserDAO(IUserDAO userDAO) {
        this.userDAO = userDAO;
    }

    @Override
    public List login(UsersEntity condition) {
        return userDAO.search(condition);
    }
}

```

9. 我们下一步要继续写 Action 代码
 UserManagerAction

```

package com.action;

import com.biz.IUserBiz;
import com.entity.UsersEntity;
import com.opensymphony.xwork2.ActionSupport;
import java.util.List;

public class UserManagerAction extends ActionSupport {
    private String loginName;
    private String loginPwd;

    public void setLoginName(String loginName) {
        this.loginName = loginName;
    }

    public String getLoginName() {
        return loginName;
    }

    public void setLoginPwd(String loginPwd) {
        this.loginPwd = loginPwd;
    }

    public String getLoginPwd() {
        return loginPwd;
    }
}

```

```

IUserBiz userBiz;

public void setUserBiz(IUserBiz userBiz) {
    this.userBiz = userBiz;
}

@Override
public String execute() throws Exception {
    UsersEntity condition = new UsersEntity();
    condition.setAccount(loginName);
    condition.setPassword(loginPwd);
    List list = userBiz.login(condition);
    if(list.size()>0) {
        return "success";
    } else {
        return "error";
    }
}
}

```

10. 由于 Spring 是依赖于注入，因此各个类中的属性是依赖于 Spring 配置文件中声明由 Spring 容器自动产生后注入给类去使用，所以不需要在类中声明 new 这个属性。

例如：

```

//依赖于 Spring 注入的变量
IUserBiz userBiz;

public void setUserBiz(IUserBiz userBiz) {
    this.userBiz = userBiz;
}

```

我们要对 Spring 的配置文件再做一些修改，将业务逻辑织入配置里。

```

<?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">

    <bean id="sessionFactory" class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">
        <property name="annotatedClasses">
            <list>
                <value>com.entity.UsersEntity</value>
            </list>
        </property>
        <property name="hibernateProperties">
            <props>
                <prop key="hibernate.connection.url">jdbc:mysql://localhost:3306/demo</prop>
                <prop key="hibernate.connection.driver_class">com.mysql.jdbc.Driver</prop>
                <prop key="dialect">org.hibernate.dialect.MySQLDialect</prop>
                <prop key="hibernate.connection.username">root</prop>
                <prop key="hibernate.connection.password">root</prop>
                <prop key="hibernate.current_session_context_class">thread</prop>
            </props>
        </property>
    </bean>

    <bean id="userDAO" class="com.dao.UserDAO">
        <property name="sessionFactory" ref="sessionFactory"/>
    </bean>

    <bean id="userBiz" class="com.biz.UserBiz">
        <property name="userDAO" ref="userDAO"/>
    </bean>

    <bean id="umAction" class="com.action.UserManagerAction" scope="prototype">
        <property name="userBiz" ref="userBiz"/>
    </bean>
</beans>

```

11. 最后我们来写对应的 JSP 文件

login.jsp

```

<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
<head>
    <title>Login</title>
</head>
<body>
<s:form action="doLogin">
    <table>
        <tr>
            <s:textfield name="loginName" label="用户名"/>
        </tr>
        <tr>
            <s:textfield name="loginPwd" label="密码"/>
        </tr>
        <tr>
            <s:submit value="确认"/>
        </tr>
    </table>
</s:form>

```

```
</body>
</html>
```

Index.jsp

```
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
  <head>
    <title>${Title}</title>
  </head>
  <body>
    <a href="login.jsp">登录</a>
  </body>
</html>
```

error.jsp 和 success.jsp 很简单，省略之

12. 最后配置 struts.xml，要注意的一点是要把 Action 的 class 指向 spring-config.xml 中配置的 bean (umAction)，不是直接指向对应的 com.action.UserManagerAction 类

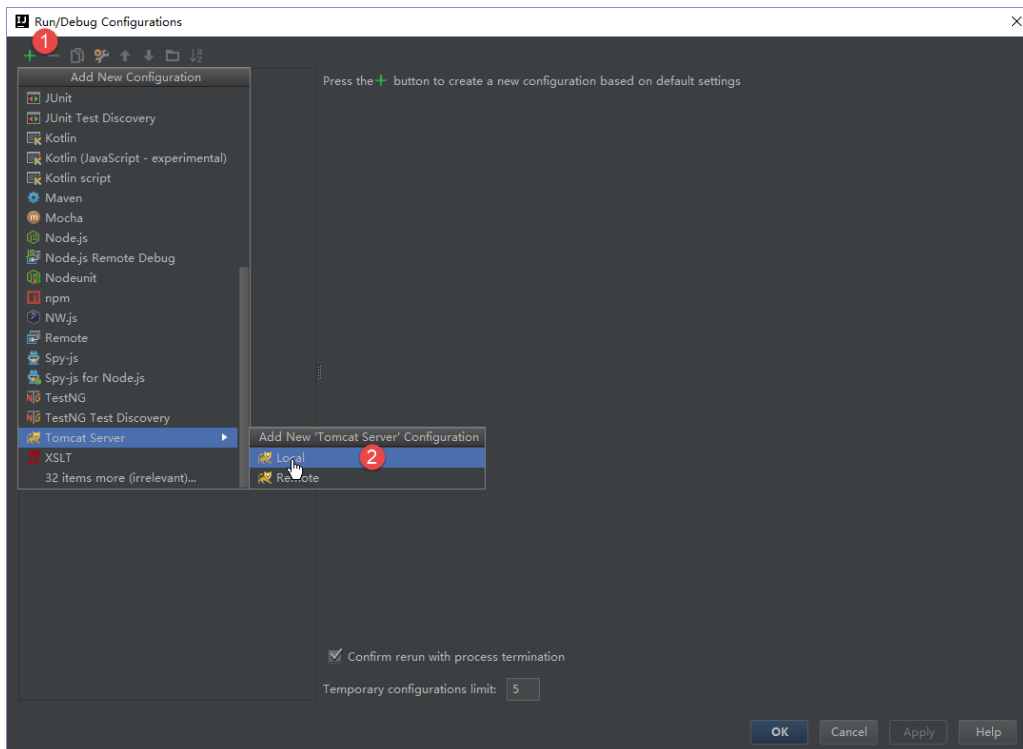
```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"
    "http://struts.apache.org/dtds/struts-2.3.dtd">

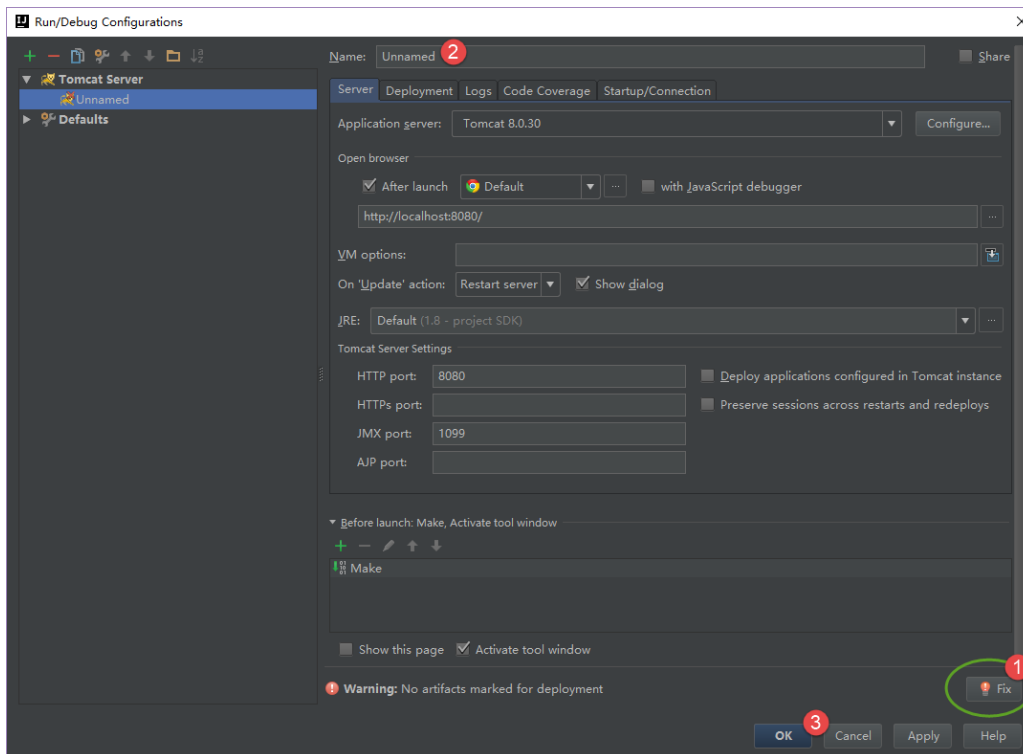
<struts>
  <package name="default" extends="struts-default">
    <action name="doLogin" class="umAction">
      <result name="error">error.jsp</result>
      <result name="success">success.jsp</result>
    </action>
  </package>
  <constant name="struts.i18n.encoding" value="GBK"/>
</struts>
```

13. 以上就是全部的代码书写过程

14. 最后我们来配置一下系统的运行环境，在菜单 Run 中，选择 Edit Configurations，然后增加一个运行环境，如下图



给运行环境起个名字，注意右下角有个错误，报告在发布时会缺少文件，点击 Fix 按钮将缺少的文件添加进去



15. 配置好后，就可以运行了