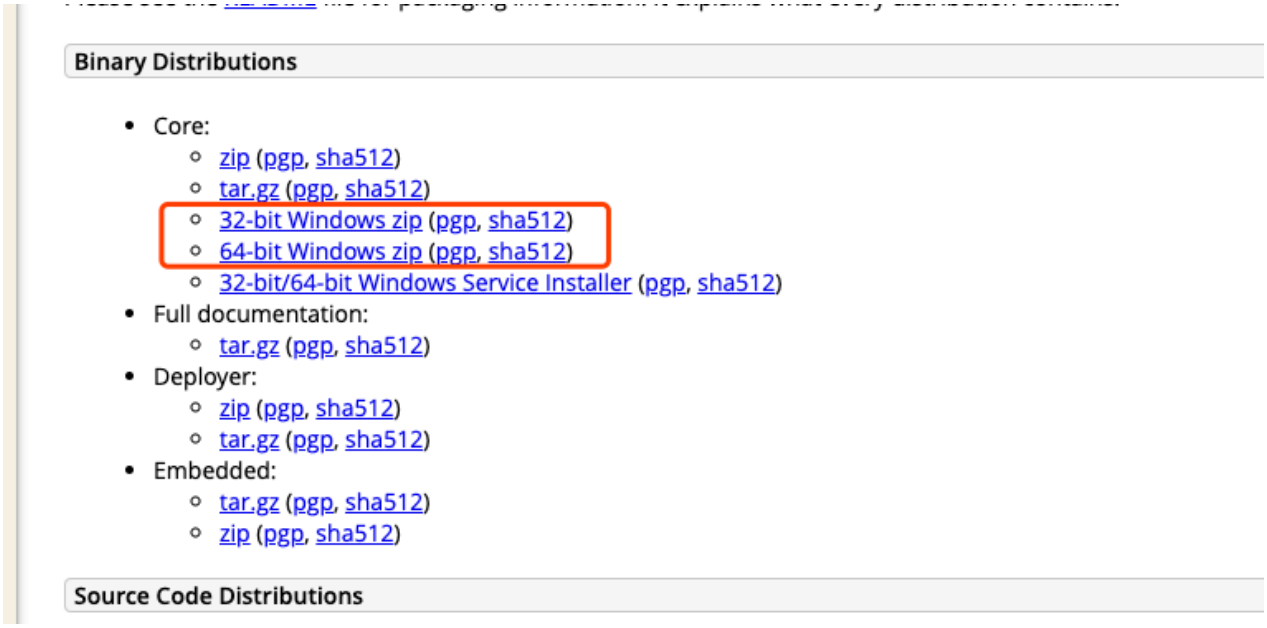
Tomcat

Web 应用服务器：Tomcat、Jboos、Weblogic、Jetty

- 安装 Tomcat

1、官网下载压缩文件。

https://tomcat.apache.org/download-90.cgi



2、解压缩。

bin：存放各个平台下启动和停止 Tomcat 服务的脚本文件。

conf：存放各种 Tomcat 服务器的配置文件。

lib：存放 Tomcat 服务器所需要的 jar。

logs：存放 Tomcar 服务运行的日志。

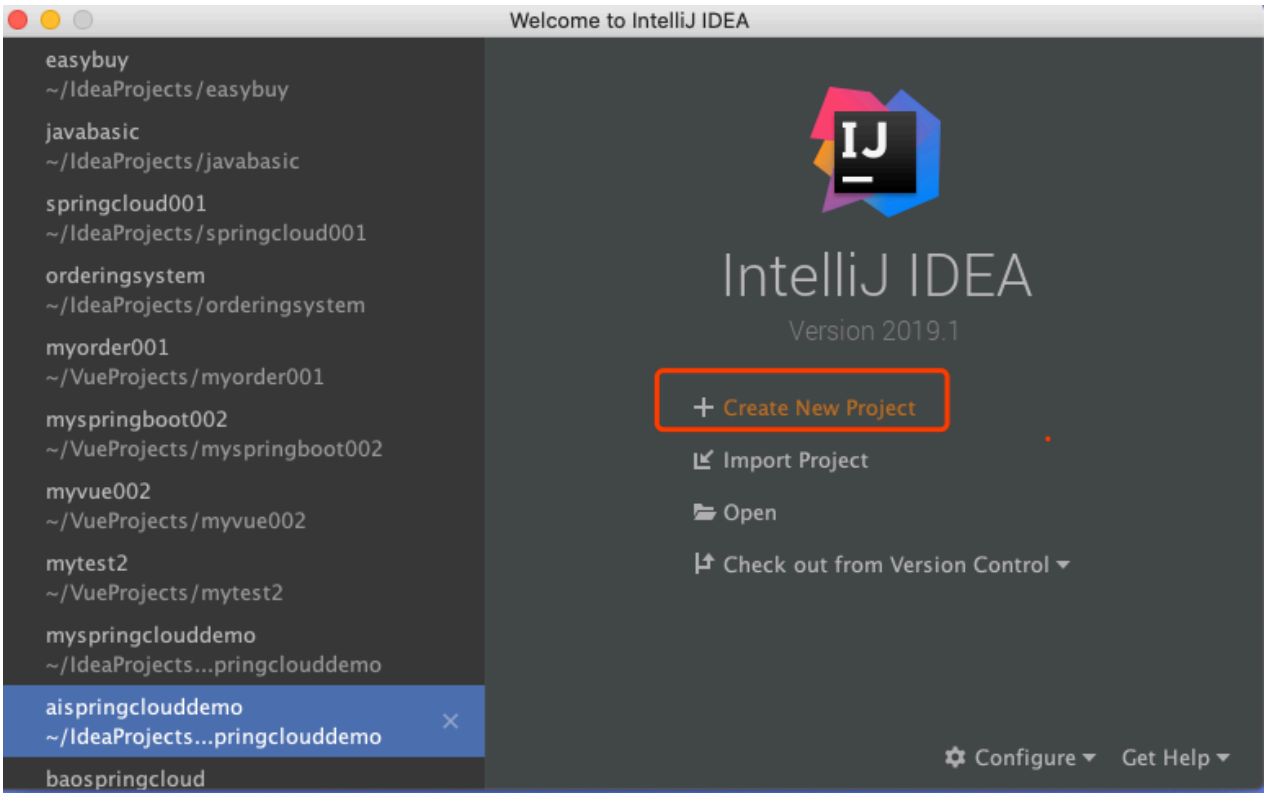
temp：Tomcat 运行时的临时文件。

webapps：存放允许客户端访问的资源（Java 程序）。

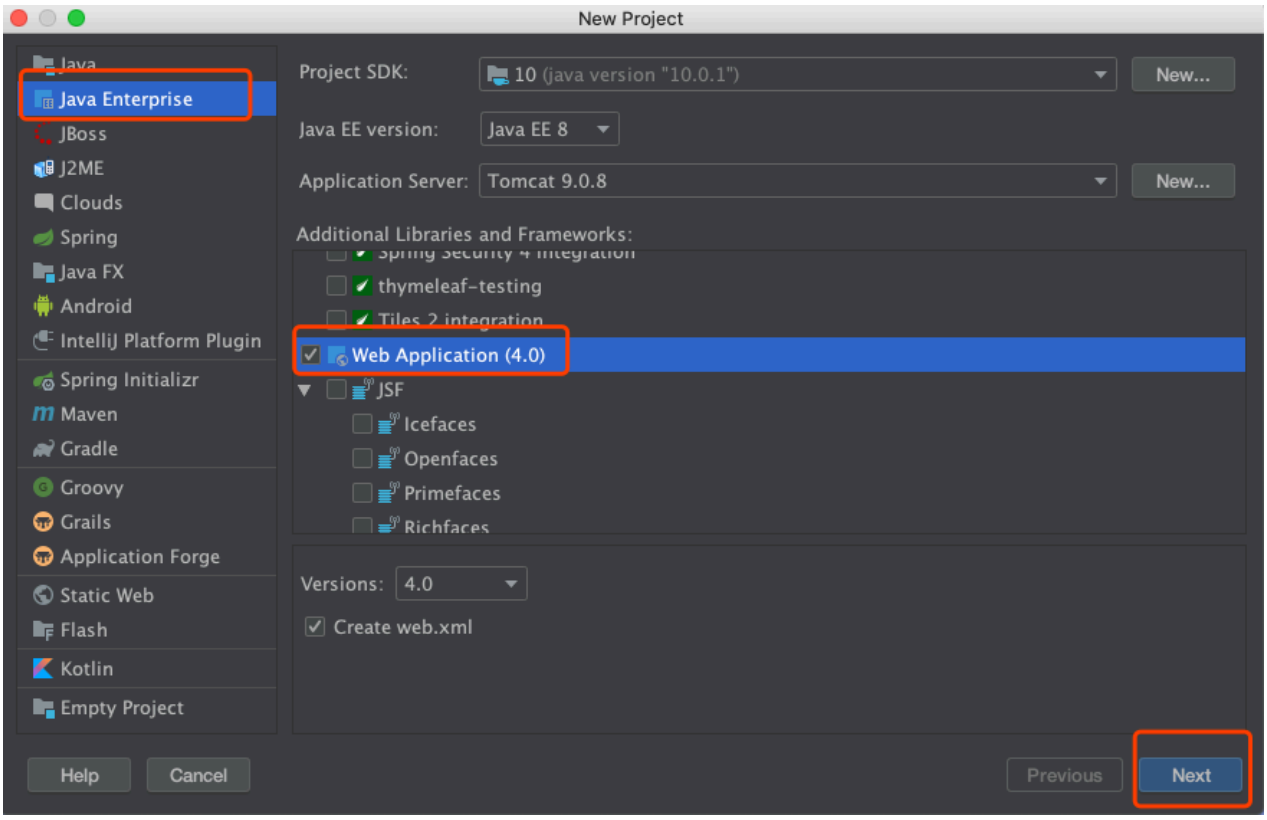
work：存放 Tomcat 将 JSP 转换之后的 Servlet 文件。

IDEA 集成 Tomcat

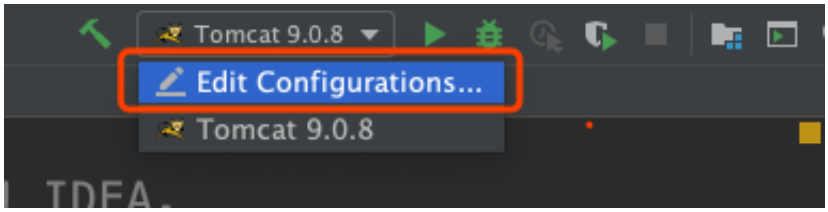
1、创建 Java Web 工程。

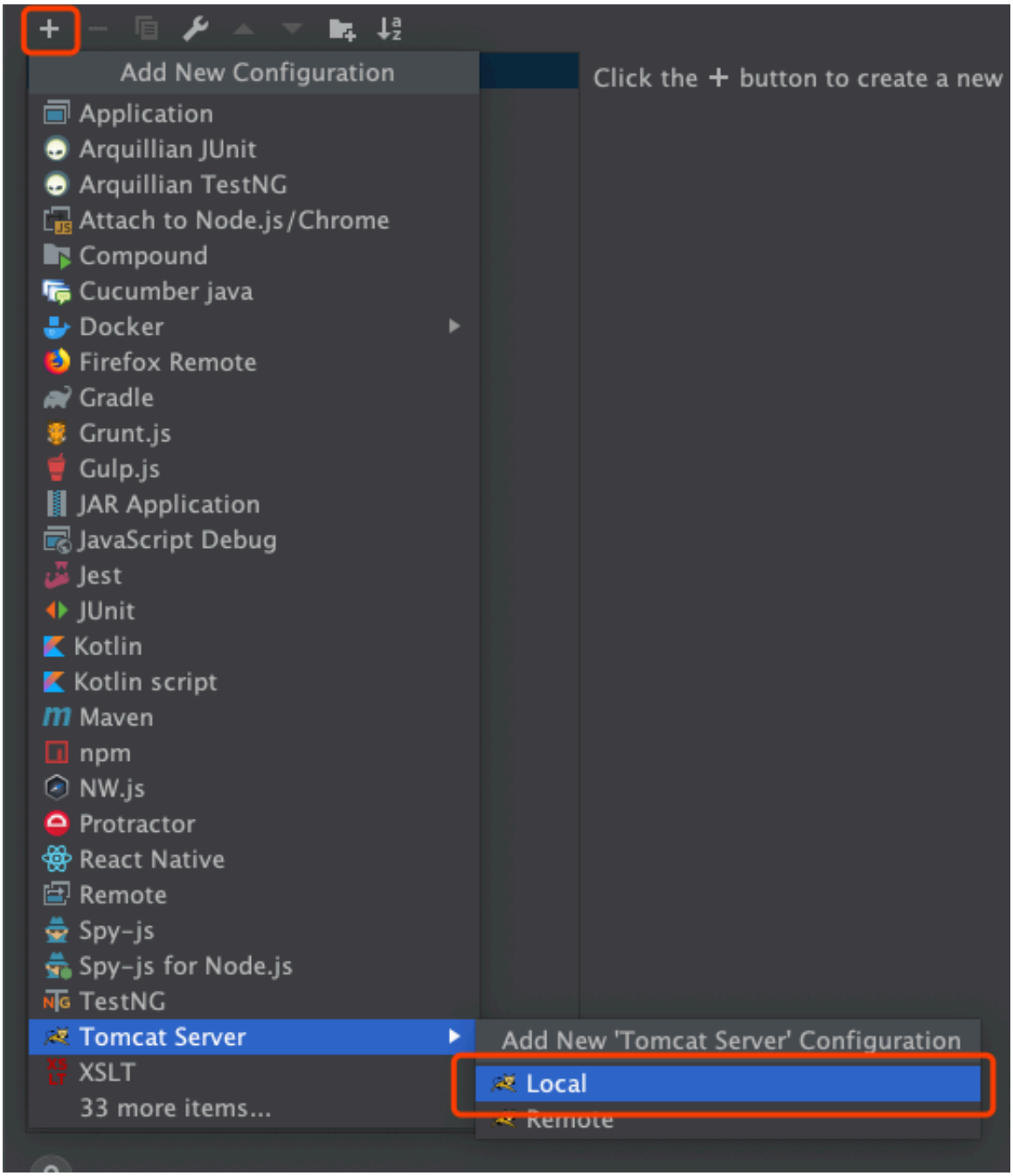


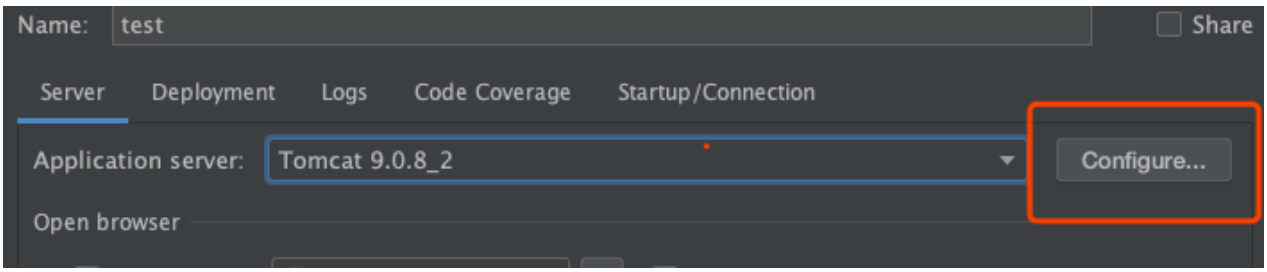
2、

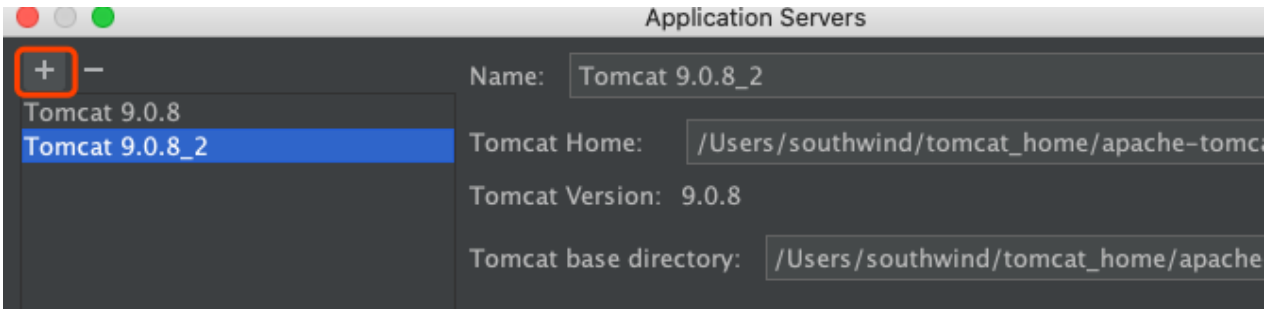


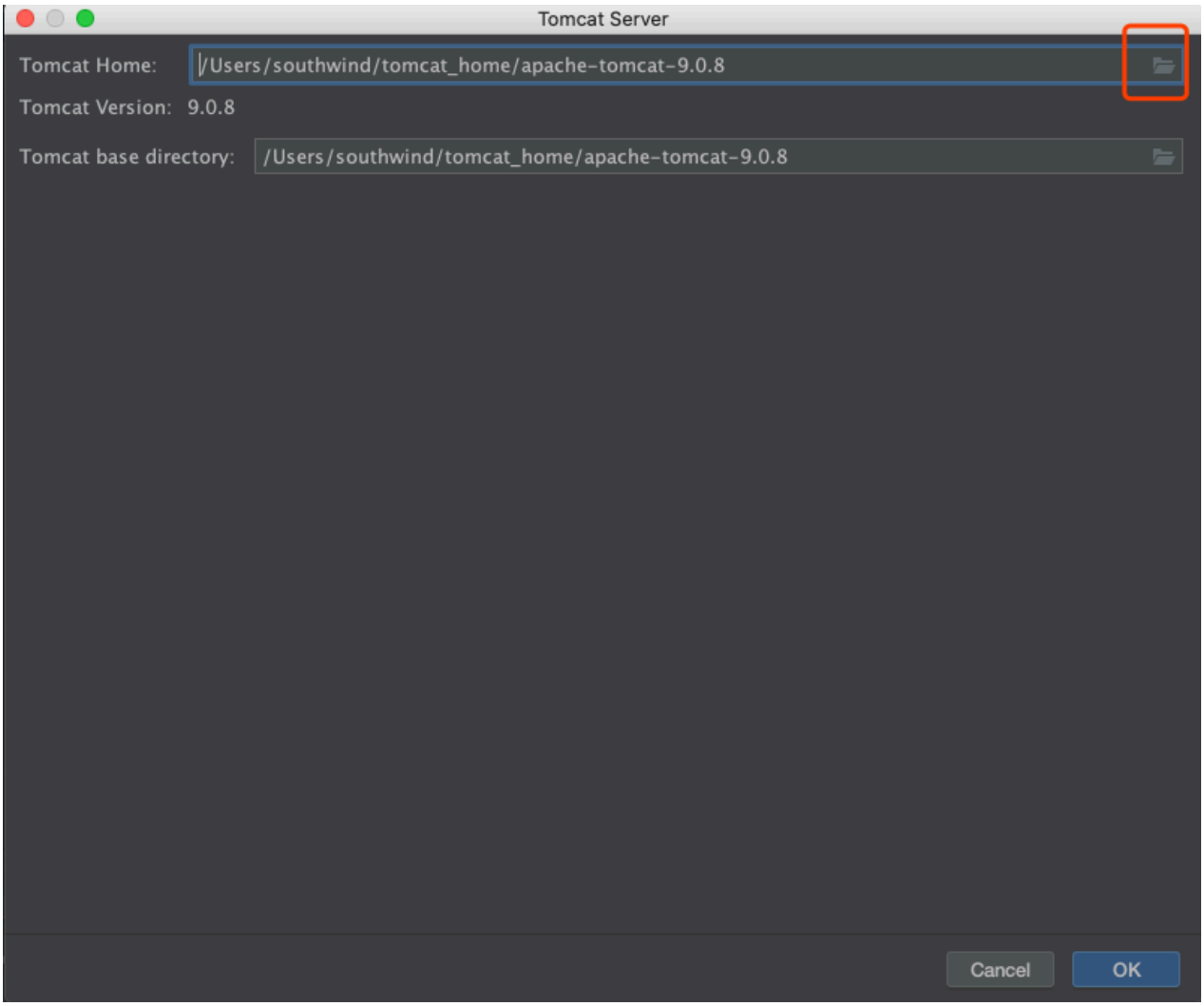
3、IDEA 中配置 Tomcat。

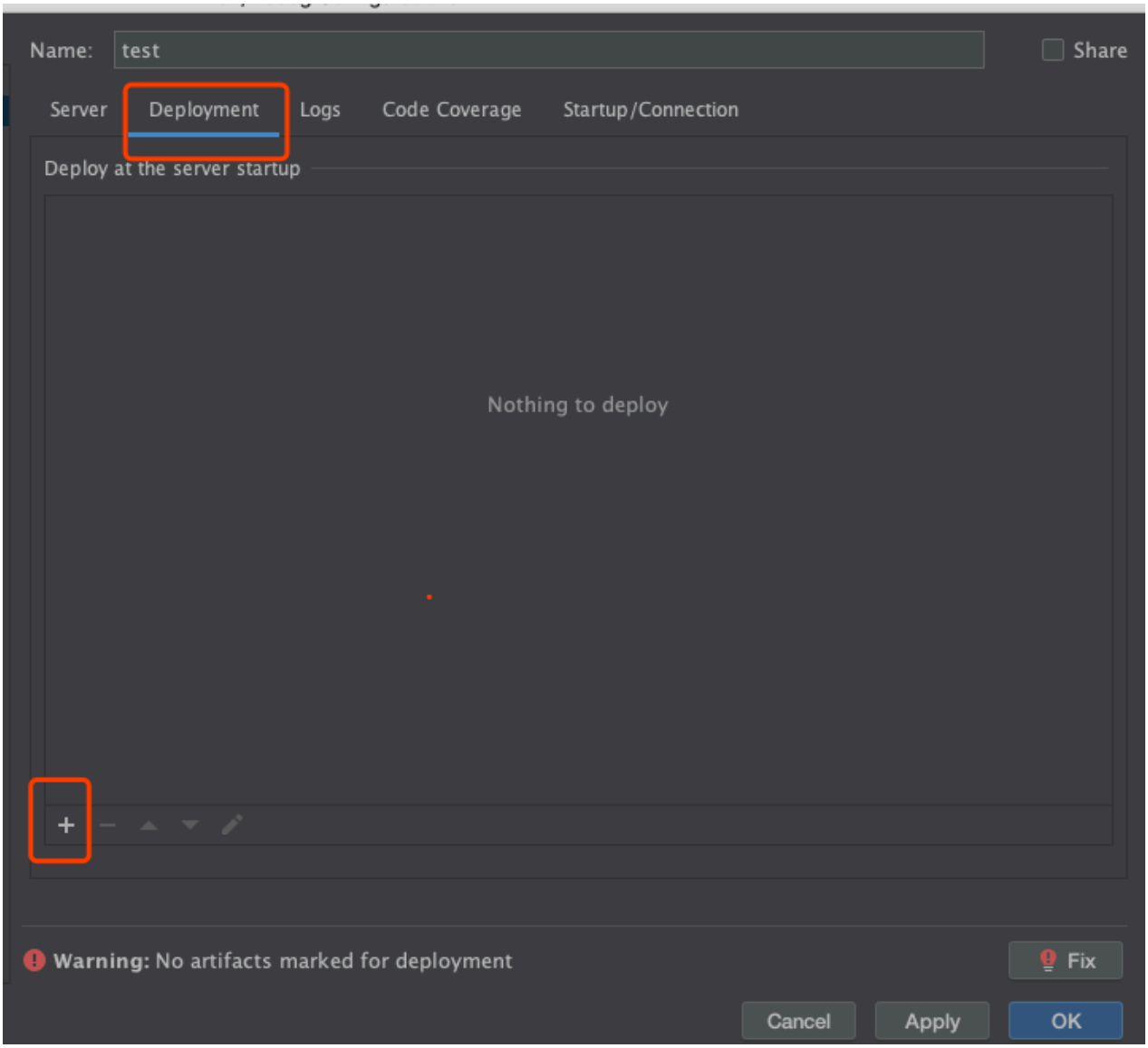



Add New Configuration
Application
Arquillian JUnit
Arquillian TestNG
Attach to Node.js/Chrome
Compound
Cucumber java
Docker
Firefox Remote
Gradle
Grunt.js
Gulp.js
JAR Application
JavaScript Debug
Jest
JUnit
Kotlin
Kotlin script
Maven
npm
NW.js
Protractor
React Native
Remote
Spy-js
Spy-js for Node.js
TestNG
Tomcat Server
XSLT
33 more items...
Add New 'Tomcat Server' Configuration
Local
Remote
Click the + button to create a new


Name:
test
Share
Server
Deployment
Logs
Code Coverage
Startup/Connection
Application server:
Tomcat 9.0.8_2
Configure...
Open browser


Application Servers
Tomcat 9.0.8
Tomcat 9.0.8_2
Name:
Tomcat 9.0.8_2
Tomcat Home:
/Users/southwind/tomcat_home/apache-tomc
Tomcat Version:
9.0.8
Tomcat base directory:
/Users/southwind/tomcat_home/apache


Tomcat Server
Tomcat Home:
/Users/southwind/tomcat_home/apache-tomcat-9.0.8
Tomcat Version: 9.0.8
Tomcat base directory:
/Users/southwind/tomcat_home/apache-tomcat-9.0.8
Cancel
OK



Servlet

- 什么是 Servlet?

Servlet 是 Java Web 开发的基石，与平台无关的服务器组件，它是运行在 Servlet 容器/Web 应用服务器/Tomcat，负责与客户端进行通信。

Servlet 的功能：

1、创建并返回基于客户请求的动态 HTML 页面。

2、与数据库进行通信。

- 如何使用 Servlet?

Servlet 本身是一组接口，自定义一个类，并且实现 Servlet 接口，这个类就具备了接受客户端请求以及做出响应的功能。

```
package com.southwind.servlet;
```

```
import javax.servlet.*;
import java.io.IOException;

public class MyServlet implements Servlet {
    @Override
    public void init(ServletConfig servletConfig) throws ServletException {

    }

    @Override
    public ServletConfig getServletConfig() {
        return null;
    }

    @Override
    public void service(ServletRequest servletRequest, ServletResponse
servletResponse) throws ServletException, IOException {
        String id =  servletRequest.getParameter("id");
        System.out.println("我是Servlet，我已经接收到了客户端发来的请求，参数是"+id);
        servletResponse.setContentType("text/html;charset=UTF-8");
        servletResponse.getWriter().write("客户端你好，我已接收到你的请求");
    }

    @Override
    public String getServletInfo() {
        return null;
    }

    @Override
    public void destroy() {

    }
}
```

浏览器不能直接访问 Servlet 文件，只能通过映射的方式来间接访问 Servlet，映射需要开发者手动配置，有两种配置方式。

- 基于 XML 文件的配置方式。

```
<servlet>
  <servlet-name>hello</servlet-name>
  <servlet-class>com.southwind.servlet.HelloServlet</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>hello</servlet-name>
  <url-pattern>/demo2</url-pattern>
</servlet-mapping>
```

- 基于注解的方式。

```
@WebServlet("/demo2")
public class HelloServlet implements Servlet {

}
```

上述两种配置方式结果完全一致，将 demo2 与 HelloServlet 进行映射，即在浏览器地址栏中直接访问 demo 就可以映射到 HelloServlet。

Servlet 的生命周期

1、当浏览器访问 Servlet 的时候，Tomcat 会查询当前 Servlet 的实例化对象是否存在，如果不存在，则通过反射机制动态创建对象，如果存在，直接执行第 3 步。

2、调用 init 方法完成初始化操作。

3、调用 service 方法完成业务逻辑操作。

4、关闭 Tomcat 时，会调用 destory 方法，释放当前对象所占用的资源。

Servlet 的生命周期方法：无参构造函数、init、service、destory

1、无参构造函数只调用一次，创建对象。

2、init 只调用一次，初始化对象。

3、service 调用 N 次，执行业务方法。

4、destory 只调用一次，卸载对象。

ServletConfig

该接口是用来描述 Servlet 的基本信息的。

getServletName() 返回 Servlet 的名称，全类名(带着包名的类名)

getInitParameter(String key) 获取 init 参数的值（web.xml）

getInitParameterNames() 返回所有的 initParamter 的 name 值，一般用作遍历初始化参数

getServletContext() 返回 ServletContext 对象，它是 Servlet 的上下文，整个 Servlet 的管理者。

ServletConfig 和 ServletContext 的区别：

ServletConfig 作用于某个 Servlet 实例，每个 Servlet 都有对应的 ServletConfig，ServletContext 作用于整个 Web 应用，一个 Web 应用对应一个 ServletContext，多个 Servlet 实例对应一个 ServletContext。

一个是局部对象，一个是全局对象。

Servlet 的层次结构

Servlet ---》GenericServlet ---〉HttpServlet

HTTP 请求有很多种类型，常用的有四种：

GET 读取

POST 保存

PUT 修改

DELETE 删除

GenericServlet 实现 Servlet 接口，同时为它的子类屏蔽了不常用的方法，子类只需要重写 service 方法即可。

HttpServlet 继承 GenericServlet，根据请求类型进行分发处理，GET 进入 doGET 方法，POST 进入 doPOST 方法。

开发者自定义的 Servlet 类只需要继承 HttpServlet 即可，重新 doGET 和 doPOST。

```
package com.southwind.servlet;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

@WebServlet("/test")
public class TestServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        resp.getWriter().write("GET");
    }

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        resp.getWriter().write("POST");
    }
}
```

JSP

JSP 本质上就是一个 Servlet，JSP 主要负责与用户交互，将最终的界面呈现给用户，HTML+JS+CSS+Java 的混合文件。

当服务器接收到一个后缀是 jsp 的请求时，将该请求交给 JSP 引擎去处理，每一个 JSP 页面第一次被访问的时候，JSP 引擎会将它翻译成一个 Servlet 文件，再由 Web 容器调用 Servlet 完成响应。

单纯从开发的角度看，JSP 就是在 HTML 中嵌入 Java 程序。

具体的嵌入方式有 3 种：

1、JSP 脚本，执行 Java 逻辑代码

```
<% Java代码 %>
```

2、JSP 声明：定义 Java 方法

```
<%!
  声明 Java 方法
%>
```

3、JSP 表达式：把 Java 对象直接输出到 HTML 页面中

```
<%=Java变量 %>
```

```
<%!
  public String test(){
  return "HelloWorld";
}
%>

<%
String str = test();
%>

<%=str%>
```

JSP内置对象 9 个

1、request：表示一次请求，HttpServletRequest。

2、response：表示一次响应，HttpServletResponse。

3、pageContext：页面上下文，获取页面信息，PageContext。

4、session：表示一次会话，保存用户信息，HttpSession。

5、application：表示当前 Web 应用，全局对象，保存所有用户共享信息，ServletContext。

6、config：当前 JSP 对应的 Servlet 的 ServletConfig 对象，获取当前 Servlet 的信息。

7、out：向浏览器输出数据，JspWriter。

8、page：当前 JSP 对应的 Servlet 对象，Servlet。

9、exception：表示 JSP 页面发生的异常，Exception。

常用的是 request、response、session、application、pageContext

request 常用方法：

1、String getParameter(String key) 获取客户端传来的参数。

2、void setAttribute(String key,Object value) 通过键值对的形式保存数据。

3、Object getAttribute(String key) 通过 key 取出 value。

4、RequestDispatcher getRequestDispatcher(String path) 返回一个 RequestDispatcher 对象，该对象的 forward 方法用于请求转发。

5、String[] getParameterValues() 获取客户端传来的多个同名参数。

6、void setCharacterEncoding(String charset) 指定每个请求的编码。

HTTP 请求状态码

200：正常

404：资源找不到

400：请求类型不匹配

500：Java 程序抛出异常

response 常用方法：

1、sendRedirect(String path) 重定向，页面之间的跳转。

转发 getRequestDispatcher 和重定向 sendRedirect 的区别：

转发是将同一个请求传给下一个页面，重定向是创建一个新的请求传给下一个页面，之前的请求结束生命周期。

转发：同一个请求在服务器之间传递，地址栏不变，也叫服务器跳转。

重定向：由客户端发送一次新的请求来访问跳转后的目标资源，地址栏改变，也叫客户端跳转。

如果两个页面之间需要通过 request 来传值，则必须使用转发，不能使用重定向。

用户登录，如果用户名和密码正确，则跳转到首页（转发），并且展示用户名，否则重新回到登陆页面（重定向）。

Session

用户会话

服务器无法识别每一次 HTTP 请求的出处（不知道来自于哪个终端），它只会接受到一个请求信号，所以就存在一个问题：将用户的响应发送给其他人，必须有一种技术来让服务器知道请求来自哪，这就是会话技术。

会话：就是客户端和服务器之间发生的一系列连续的请求和响应的过程，打开浏览器进行操作到关闭浏览器的过程。

会话状态：指服务器和浏览器在会话过程中产生的状态信息，借助于会话状态，服务器能够把属于同一次会话的一系列请求和响应关联起来。

实现会话有两种方式：

- session
- cookie

属于同一次会话的请求都有一个相同的标识符，sessionID

session 常用的方法：

String getId() 获取 sessionID

void setMaxInactiveInterval(int interval) 设置 session 的失效时间，单位为秒

int getMaxInactiveInterval() 获取当前 session 的失效时间

void invalidate() 设置 session 立即失效

void setAttribute(String key,Object value) 通过键值对的形式来存储数据

Object getAttribute(String key) 通过键获取对应的数据

void removeAttribute(String key) 通过键删除对应的数据

Cookie

Cookie 是服务端在 HTTP 响应中附带传给浏览器的一个小文本文件，一旦浏览器保存了某个 Cookie，在之后的请求和响应过程中，会将此 Cookie 来回传递，这样就可以通过 Cookie 这个载体完成客户端和服务端的数据交互。

Cookie

- 创建 Cookie

```
Cookie cookie = new Cookie("name","tom");
response.addCookie(cookie);
```

- 读取 Cookie

```
Cookie[] cookies = request.getCookies();
for (Cookie cookie:cookies){
    out.write(cookie.getName()+":"+cookie.getValue()+"<br/>");
}
```

Cookie 常用的方法

void setMaxAge(int age) 设置 Cookie 的有效时间，单位为秒

int getMaxAge() 获取 Cookie 的有效时间

String getName() 获取 Cookie 的 name

String getValue() 获取 Cookie 的 value

Session 和 Cookie 的区别

session：保存在服务器

保存的数据是 Object

会随着会话的结束而销毁

保存重要信息

cookie：保存在浏览器

保存的数据是 String

可以长期保存在浏览器中，无会话无关

保存不重要信息

存储用户信息：

session：setAttribute("name","admin") 存

getAttribute("name") 取

生命周期：服务端：只要 WEB 应用重启就销毁，客户端：只要浏览器关闭就销毁。

退出登录：session.invalidate()

cookie：response.addCookie(new Cookie(name,"admin")) 存

```
Cookie[] cookies = request.getCookies();
for (Cookie cookie:cookies){
    if(cookie.getName().equals("name")){
        out.write("欢迎回来"+cookie.getValue());
    }
}
```

取

生命周期：不随服务端的重启而销毁，客户端：默认是只要关闭浏览器就销毁，我们通过 setMaxAge() 方法设置有效期，一旦设置了有效期，则不随浏览器的关闭而销毁，而是由设置的时间来决定。

退出登录：setMaxAge(0)

JSP 内置对象作用域

4个

page、request、session、application

setAttribute、getAttribute

page 作用域：对应的内置对象是 pageContext。

request 作用域：对应的内置对象是 request。

session 作用域：对应的内置对象是 session。

application 作用域：对应的内置对象是 application。

page < request < session < application

page 只在当前页面有效。

request 在一次请求内有效。

session 在一次会话内有效。

application 对应整个 WEB 应用的。

- 网站访问量统计

```
<%
  Integer count = (Integer) application.getAttribute("count");
  if(count == null){
    count = 1;
    application.setAttribute("count",count);
  }else{
    count++;
    application.setAttribute("count",count);
  }
%>
您是当前的第<%=count%>位访客
```

EL 表达式

Expression Language 表达式语言，替代 JSP 页面中数据访问时的复杂编码，可以非常便捷地取出域对象（pageContext、request、session、application）中保存的数据，前提是一定要先 setAttribute，EL 就相当于在简化 getAttribute

${变量名} 变量名就是 setAttribute 对应的 key 值。

1、EL 对于 4 种域对象的默认查找顺序：

pageContext -》request-〉session-》application

按照上述的顺序进行查找，找到立即返回，在 application 中也无法找到，则返回 null

2、指定作用域进行查找

pageContext：${pageScope.name}

request：${requestScope.name}

session：${sessionScope.name}

application：${applicationScope.name}

数据级联：

```
<%
//        pageContext.setAttribute("name","page");
//        request.setAttribute("name","request");
//        session.setAttribute("name","session");
//        application.setAttribute("name","application");
        User user = new User(1,"张三",86.5,new Address(1,"小寨"));
        System.out.println(user.toString());
        pageContext.setAttribute("user",user);
    %>
    <table>
        <tr>
            <th>编号</th>
            <th>姓名</th>
            <th>成绩</th>
            <th>地址</th>
        </tr>
        <tr>
            <td>${user.id}</td>
            <td>${user.name}</td>
            <td>${user.score}</td>
            <td>${user.address}</td>
        </tr>
    </table>
```

${user["id"]}

EL 执行表达式

```
${num1&&num2}
&& || ! < > <= <= ==

&& and
|| or
! not
== eq
!= ne
< lt
> gt
<= le
>= ge
empty 变量为 null，长度为0的String，size为0的集合
```

JSTL

JSP Standard Tag Library JSP 标准标签库，JSP 为开发者提供的一系列的标签，使用这些标签可以完成一些逻辑处理，比如循环遍历集合，让代码更加简洁，不再出现 JSP 脚本穿插的情况。

实际开发中 EL 和 JSTL 结合起来使用，JSTL 侧重于逻辑处理，EL 负责展示数据。

JSTL 的使用

1、需要导入 jar 包（两个 jstl.jar standard.jar）存放的位置 web/WEB-INF

2、在 JSP 页面开始的地方导入 JSTL 标签库

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
```

3、在需要的地方使用

```
<c:forEach items="${list}" var="user">
  <tr>
    <td>${user.id}</td>
    <td>${user.name}</td>
    <td>${user.score}</td>
    <td>${user.address.value}</td>
  </tr>
</c:forEach>
```

JSTL 优点：

1、提供了统一的标签

2、可以用于编写各种动态功能

核心标签库常用标签：

- set、out、remove、catch

set：向域对象中添加数据

```
<%
  requset.setAttribute(key,value)
%>

<c:set var="name" value="tom" scope="request"></c:set>
${requestScope.name}

<%
User user = new User(1,"张三",66.6,new Address(1,"科技路"));
request.setAttribute("user",user);
%>
${user.name}
<hr/>
<c:set target="${user}" property="name" value="李四"></c:set>
${user.name}
```

out：输出域对象中的数据

```
<c:set var="name" value="tom"></c:set>
<c:out value="${name}" default="未定义"></c:out>
```

remove：删除域对象中的数据

```
<c:remove var="name" scope="page"></c:remove>
<c:out value="${name}" default="未定义"></c:out>
```

catch：捕获异常

```
<c:catch var="error">
  <%
  int a = 10/0;
  %>
</c:catch>
${error}
```

- 条件标签：if choose

```
<c:set var="num1" value="1"></c:set>
<c:set var="num2" value="2"></c:set>
<c:if test="${num1>num2}">ok</c:if>
<c:if test="${num1<num2}">fail</c:if>
<hr/>
<c:choose>
  <c:when test="${num1>num2}">ok</c:when>
  <c:otherwise>fail</c:otherwise>
</c:choose>
```

- 迭代标签：forEach

```
<c:forEach items="${list}" var="str" begin="2" end="3" step="2"
varStatus="sta">
  ${sta.count}、${str}<br/>
</c:forEach>
```

格式化标签库常用的标签：

```
<%
request.setAttribute("date",new Date());
%>
<fmt:formatDate value="${date}" pattern="yyyy-MM-dd HH:mm:ss">
</fmt:formatDate><br/>
<fmt:formatNumber value="32145.23434" maxIntegerDigits="2"
maxFractionDigits="3"></fmt:formatNumber>
```

函数标签库常用的标签：

```
<%
request.setAttribute("info","Java,C");
%>
${fn:contains(info,"Python")}<br/>
${fn:startsWith(info, "Java")}<br/>
${fn:endsWith(info, "C")}<br/>
${fn:indexOf(info, "va")}<br/>
${fn:replace(info, "C","Python")}<br/>
${fn:substring(info, 2, 3)}<br/>
${fn:split(info, ",")[0]}-${fn:split(info, ",")[1]}
```

过滤器

Filter

功能：

1、用来拦截传入的请求和传出的响应。

2、修改或以某种方式处理正在客户端和服务端之间交换的数据流。

如何使用?

与使用 Servlet 类似，Filter 是 Java WEB 提供的一个接口，开发者只需要自定义一个类并且实现该接口即可。

```
package com.southwind.filter;

import javax.servlet.*;
import java.io.IOException;

public class CharacterFilter implements Filter {

    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse
servletResponse, FilterChain filterChain) throws IOException, ServletException
{
        servletRequest.setCharacterEncoding("UTF-8");
        filterChain.doFilter(servletRequest,servletResponse);
    }

}
```

web.xml 中配置 Filter

```
<filter>
  <filter-name>charcater</filter-name>
  <filter-class>com.southwind.filter.CharacterFilter</filter-class>
</filter>

<filter-mapping>
  <filter-name>charcater</filter-name>
  <url-pattern>/login</url-pattern>
  <url-pattern>/test</url-pattern>
</filter-mapping>
```

注意：doFilter 方法中处理完业务逻辑之后，必须添加 filterChain.doFilter(servletRequest,servletResponse);

否则请求/响应无法向后传递，一直停留在过滤器中。

Filter 的生命周期

当 Tomcat 启动时，通过反射机制调用 Filter 的无参构造函数创建实例化对象，同时调用 init 方法实现初始化，doFilter 方法调用多次，当 Tomcat 服务关闭的时候，调用 destory 来销毁 Filter 对象。

无参构造函数：只调用一次，当 Tomcat 启动时调用（Filter 一定要进行配置）

init 方法：只调用一次，当 Filter 的实例化对象创建完成之后调用

doFilter：调用多次，访问 Filter 的业务逻辑都写在 Filter 中

destory：只调用一次，Tomcat 关闭时调用。

同时配置多个 Filter，Filter 的调用顺序是由 web.xml 中的配置顺序来决定的，写在上面的配置先调用，因为 web.xml 是从上到下顺序读取的。

```
<filter>
  <filter-name>my</filter-name>
  <filter-class>com.southwind.filter.MyFilter</filter-class>
</filter>

<filter-mapping>
  <filter-name>my</filter-name>
  <url-pattern>/login</url-pattern>
</filter-mapping>

<filter>
  <filter-name>charcater</filter-name>
  <filter-class>com.southwind.filter.CharacterFilter</filter-class>
</filter>

<filter-mapping>
  <filter-name>charcater</filter-name>
  <url-pattern>/login</url-pattern>
  <url-pattern>/test</url-pattern>
</filter-mapping>
```

1、MyFilter

2、CharacterFilter

也可以通过注解的方式来简化 web.xml 中的配置

```
<filter>
  <filter-name>my</filter-name>
  <filter-class>com.southwind.filter.MyFilter</filter-class>
</filter>

<filter-mapping>
  <filter-name>my</filter-name>
  <url-pattern>/login</url-pattern>
</filter-mapping>
```

等于

```
@WebFilter("/login")
public class MyFilter implements Filter {

}
```

实际开发中 Filter 的使用场景：

1、统一处理中文乱码。

2、屏蔽敏感词。

```
package com.southwind.filter;

import javax.servlet.*;
import javax.servlet.annotation.WebFilter;
import java.io.IOException;

@WebFilter("/test")
public class WordFilter implements Filter {
    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse
servletResponse, FilterChain filterChain) throws IOException, ServletException
{
        servletRequest.setCharacterEncoding("UTF-8");
        //将"敏感词"替换成"***"
        String name = servletRequest.getParameter("name");
        name = name.replaceAll("敏感词","***");
        servletRequest.setAttribute("name",name);
        filterChain.doFilter(servletRequest,servletResponse);
    }
}
```

```
package com.southwind.servlet;
```

```
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

@WebServlet("/test")
public class TestServlet extends HttpServlet {
    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        String name = (String) req.getAttribute("name");
        System.out.println("servlet:"+name);
    }
}
```

3、控制资源的访问权限。

```
package com.southwind.filter;

import javax.servlet.*;
import javax.servlet.annotation.WebFilter;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import java.io.IOException;

@WebFilter("/download.jsp")
public class DownloadFilter implements Filter {

    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse
servletResponse, FilterChain filterChain) throws IOException, ServletException
{
        HttpServletRequest request = (HttpServletRequest) servletRequest;
        HttpServletResponse response = (HttpServletResponse) servletResponse;
        HttpSession session = request.getSession();
        String name = (String) session.getAttribute("name");
        if(name == null){
            //不是登录状态
            response.sendRedirect("/login.jsp");
        }else{
            filterChain.doFilter(servletRequest,servletResponse);
        }
    }
}
```

文件上传下载

- JSP

1、input 的 type 设置为 file

2、form 表单的 method 设置 post，get 请求会将文件名传给服务端，而不是文件本身

3、form 表单的 enctype 设置 multipart/form-data，以二进制的形式传输数据

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
    <form enctype="multipart/form-data" action="/upload" method="post">
        <input name="desc" type="text"/><br/>
        <input name="text" type="file"/><br/>
        <input type="submit" value="上传"/>
    </form>
</body>
</html>
```

- Servlet

fileupload 组件可以将所有的请求信息都解析成 FileIteam 对象，可以通过对 FileItem 对象的操作完成上传，面向对象的思想。

```
package com.southwind.servlet;

import org.apache.commons.fileupload.FileItem;
import org.apache.commons.fileupload.FileUploadException;
import org.apache.commons.fileupload.disk.DiskFileItemFactory;
import org.apache.commons.fileupload.servlet.ServletFileUpload;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.*;
import java.util.List;

@WebServlet("/upload")
public class UploadServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
```

```
    }

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
//        //通过输入流获取客户端传来的数据流
//        InputStream inputStream = req.getInputStream();
//        Reader reader = new InputStreamReader(inputStream);
//        BufferedReader bufferedReader = new BufferedReader(reader);
//        //通过输出流将数据流输出到本地硬盘
//        //获取文件夹的绝对路径
//        String path = req.getServletContext().getRealPath("file/copy.txt");
//        OutputStream outputStream = new FileOutputStream(path);
//        Writer writer = new OutputStreamWriter(outputStream);
//        BufferedWriter bufferedWriter = new BufferedWriter(writer);
//        String str = "";
//        while((str = bufferedReader.readLine())!=null){
//            System.out.println(str);
//            bufferedWriter.write(str);
//        }
//        bufferedWriter.close();
//        writer.close();
//        outputStream.close();
//        bufferedReader.close();
//        reader.close();
//        inputStream.close();
        try {
            DiskFileItemFactory fileItemFactory = new DiskFileItemFactory();
            ServletFileUpload servletFileUpload = new
ServletFileUpload(fileItemFactory);
            List<FileItem> list = servletFileUpload.parseRequest(req);
            for(FileItem fileItem : list){
                if(fileItem.isFormField()){
                    String name = fileItem.getFieldName();
                    String value = fileItem.getString("UTF-8");
                    System.out.println(name+":"+value);
                }else{
                    String fileName = fileItem.getName();
                    long size = fileItem.getSize();
                    System.out.println(fileName+":"+size+"Byte");
                    InputStream inputStream = fileItem.getInputStream();
//                    Reader reader = new InputStreamReader(inputStream);
//                    BufferedReader bufferedReader = new
BufferedReader(reader);
                    String path =
req.getServletContext().getRealPath("file/"+fileName);
                    OutputStream outputStream = new FileOutputStream(path);
//                    Writer writer = new OutputStreamWriter(outputStream);
```

```
//                    BufferedWriter bufferedWriter = new
BufferedWriter(writer);
                    int temp = 0;
                    while((temp = inputStream.read())!=-1){
                        outputStream.write(temp);
                    }
//                    bufferedWriter.close();
//                    writer.close();
                    outputStream.close();
//                    bufferedReader.close();
//                    reader.close();
                    inputStream.close();
                    System.out.println("上传成功");
                }
            }
        } catch (FileUploadException e) {
            e.printStackTrace();
        }
    }
}
```

文件下载

```
package com.southwind.servlet;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

@WebServlet("/download")
public class DownloadServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        String type = req.getParameter("type");
        String fileName = "";
        switch (type){
            case "png":
                fileName = "1.png";
                break;
            case "txt":
                fileName = "test.txt";
                break;
        }
        //设置响应方式
        resp.setContentType("application/x-msdownload");
        //设置下载之后的文件名
        resp.setHeader("Content-Disposition","attachment;filename="+fileName);
        //获取输出流
        OutputStream outputStream = resp.getOutputStream();
        String path = req.getServletContext().getRealPath("file/"+fileName);
        InputStream inputStream = new FileInputStream(path);
        int temp = 0;
        while((temp=inputStream.read())!=-1){
            outputStream.write(temp);
        }
        inputStream.close();
        outputStream.close();
    }
}
```

Ajax

Asynchronous JavaScript And XML：异步的 JavaScript 和 XML

AJAX 不是新的编程，指的是一种交互方式，异步加载，客户端和服务器的数据交互更新在局部页面的技术，不需要刷新整个页面（局部刷新）

优点：

1、局部刷新，效率更高

2、用户体验更好

基于 jQuery 的 AJAX

```
<%--
  Created by IntelliJ IDEA.
  User: southwind
  Date: 2019-12-10
  Time: 10:30
  To change this template use File | Settings | File Templates.
--%>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
    <script type="text/javascript" src="js/jquery-3.3.1.min.js"></script>
    <script type="text/javascript">
        $(function(){
            var btn = $("#btn");
            btn.click(function(){
                $.ajax({
                   url:'/test',
                    type:'post',
                    data:'id=1',
                    dataType:'text',
                    success:function(data){
                       var text = $("#text");
                       text.before("<span>"+data+"</span><br/>");
                    }
                });
            });
        })
    </script>
</head>
<body>
    <input id="text" type="text"/><br/>
    <input id="btn" type="button" value="提交"/>
</body>
```

```
</html>
```

不能用表单提交请求，改用 jQuery 方式动态绑定事件来提交。

Servlet 不能跳转到 JSP，只能将数据返回。

```
package com.southwind.servlet;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

@WebServlet("/test")
public class TestServlet extends HttpServlet {

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        String id = req.getParameter("id");
        try {
            Thread.sleep(3000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        String str = "Hello World";
        resp.getWriter().write(str);
    }
}
```

传统的 WEB 数据交互 VS AJAX 数据交互

- 客户端请求的方式不同：

传统，浏览器发送同步请求 （form、a）

AJAX，异步引擎对象发送异步请求

- 服务器响应的方式不同：

传统，响应一个完整 JSP 页面（视图）

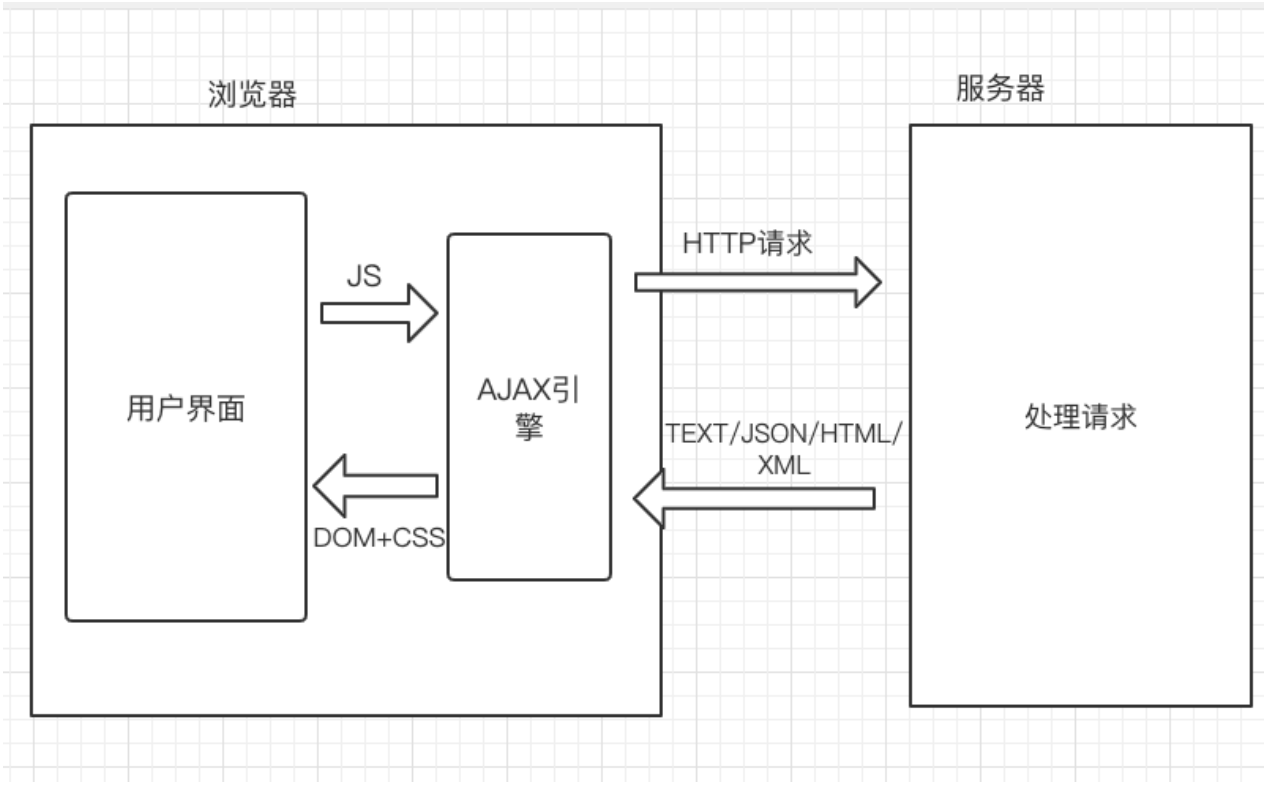
AJAX，响应需要的数据

- 客户端处理方式不同：

传统：需要等待服务器完成响应并且重新加载整个页面之后，用户才能进行后续的操作

AJAX：动态更新页面中的局部内容，不影响用户的其他操作

AJAX 原理



基于 jQuery 的 AJAX 语法

$.ajax({属性})

常用的属性参数：

url：请求的后端服务地址

type：请求方式，默认 get

data：请求参数

dataType：服务器返回的数据类型，text/json

success：请求成功的回调函数

error：请求失败的回调函数

complete：请求完成的回调函数（无论成功或者失败，都会调用）

JSON

JavaScript Object Notation，一种轻量级数据交互格式，完成 js 与 Java 等后端开发语言对象数据之间的转换

客户端和服务器之间传递对象数据，需要用 JSON 格式。

```
package com.southwind.entity;

public class User {
    private Integer id;
    private String name;
    private Double score;

    public Integer getId() {
        return id;
    }

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

    public String getName() {
        return name;
    }

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

    public Double getScore() {
        return score;
    }

    public void setScore(Double score) {
        this.score = score;
    }

    public User(Integer id, String name, Double score) {
        this.id = id;
        this.name = name;
        this.score = score;
    }
}

User user = new User(1,"张三",96.5);
```

```
var user = {
    id:1,
    name:"张三",
    score:96.5
}
```

```java
package com.southwind.servlet;

import com.southwind.entity.User;
import net.sf.json.JSONObject;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

@WebServlet("/test")
public class TestServlet extends HttpServlet {

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        User user = new User(1,"张三",96.5);
        //将 Java 对象转为 JSON 格式
        resp.setCharacterEncoding("UTF-8");
        JSONObject jsonObject = JSONObject.fromObject(user);
        resp.getWriter().write(jsonObject.toString());
    }
}
```

```jsp
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
    <script type="text/javascript" src="js/jquery-3.3.1.min.js"></script>
    <script type="text/javascript">
        $(function(){
            var btn = $("#btn");
            btn.click(function(){
                $.ajax({
                   url:'/test',
                    type:'post',
                    dataType:'json',
                    success:function(data){
                       $("#id").val(data.id);
                        $("#name").val(data.name);
                        $("#score").val(data.score);
                    }
                });
            });
```

```
        })
    </script>
</head>
<body>
    编号：<input id="id" type="text"/><br/>
    姓名：<input id="name" type="text"/><br/>
    成绩：<input id="score" type="text"/><br/>
    <input id="btn" type="button" value="提交"/>
</body>
</html>
```

AJAX

```
<%--
  Created by IntelliJ IDEA.
  User: southwind
  Date: 2019-12-10
  Time: 15:46
  To change this template use File | Settings | File Templates.
--%>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
    <script type="text/javascript" src="js/jquery-3.3.1.min.js"></script>
    <script type="text/javascript">
        $(function(){
            //修改省份
            $("#province").change(function () {
                var id = $(this).val();
                $.ajax({
                    url:"/location",
                    type:"POST",
                    data:"id="+id+"&type=province",
                    dataType:"JSON",
                    success:function(data){
                        var content = "";
                        var cities = data.cities;
                        for(var i=0;i<cities.length;i++){
                            content += "<option>"+cities[i]+"</option>";
                        }
                        $("#city").html(content);

                        content = "";
                        var areas = data.areas;
                        for(var i=0;i<areas.length;i++){
                            content += "<option>"+areas[i]+"</option>";
```

```
                        }
                        $("#area").html(content);
                    }
                });
            });
            //修改城市
            $("#city").change(function(){
                var id = $(this).val();
                $.ajax({
                    url:"/location",
                    type:"POST",
                    data:"id="+id+"&type=city",
                    dataType:"JSON",
                    success:function(data){
                        var content = "";
                        for(var i=0;i<data.length;i++){
                           content += "<option>"+data[i]+"</option>";
                        }
                        $("#area").html(content);
                    }
                });
            });
        });
    </script>
</head>
<body>
    省: <select id="province">
        <option value="陕西省">陕西省</option>
        <option value="河南省">河南省</option>
        <option value="江苏省">江苏省</option>
    </select>
    市: <select id="city">
        <option value="西安市">西安市</option>
        <option value="宝鸡市">宝鸡市</option>
        <option value="渭南市">渭南市</option>
    </select>
    区: <select id="area">
        <option>雁塔区</option>
        <option>莲湖区</option>
        <option>新城区</option>
    </select>
</body>
</html>
```

```
package com.southwind.servlet;

import com.southwind.entity.Location;
import net.sf.json.JSONArray;
import net.sf.json.JSONObject;
```

```java
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

@WebServlet("/location")
public class LocationServlet extends HttpServlet {

    private static Map<String, List<String>> cityMap;
    private static Map<String,List<String>> provinceMap;

    static {
        cityMap = new HashMap<>();
        List<String> areas = new ArrayList<>();
        //西安
        areas.add("雁塔区");
        areas.add("莲湖区");
        areas.add("新城区");
        cityMap.put("西安市",areas);
        //宝鸡
        areas = new ArrayList<>();
        areas.add("陈仓区");
        areas.add("渭宾区");
        areas.add("新城区");
        cityMap.put("宝鸡市",areas);
        //渭南
        areas = new ArrayList<>();
        areas.add("临渭区");
        areas.add("高新区");
        cityMap.put("渭南市",areas);
        //郑州
        areas = new ArrayList<>();
        areas.add("郑州A区");
        areas.add("郑州B区");
        cityMap.put("郑州市",areas);
        //洛阳
        areas = new ArrayList<>();
        areas.add("洛阳A区");
        areas.add("洛阳B区");
        cityMap.put("洛阳市",areas);

        provinceMap = new HashMap<>();
```

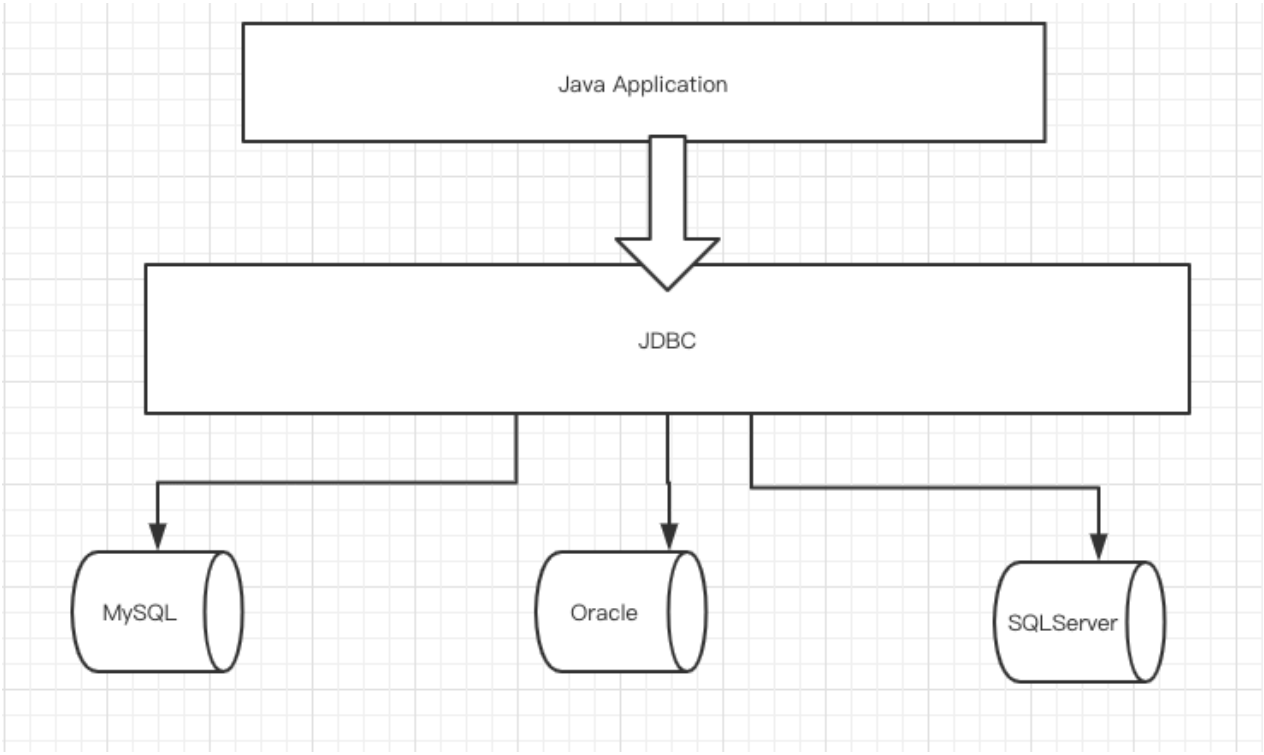
```
        List<String> cities = new ArrayList<>();
        cities.add("西安市");
        cities.add("宝鸡市");
        cities.add("渭南市");
        provinceMap.put("陕西省",cities);
        cities = new ArrayList<>();
        cities.add("郑州市");
        cities.add("洛阳市");
        cities.add("开封市");
        provinceMap.put("河南省",cities);
        cities = new ArrayList<>();
        cities.add("南京市");
        cities.add("苏州市");
        cities.add("南通市");
        provinceMap.put("江苏省",cities);
    }

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {

    }

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        String type = req.getParameter("type");
        resp.setCharacterEncoding("UTF-8");
        String id = req.getParameter("id");
        switch (type){
            case "city":
                List<String> areas = cityMap.get(id);
                JSONArray jsonArray = JSONArray.fromObject(areas);
                resp.getWriter().write(jsonArray.toString());
                break;
            case "province":
                List<String> cities = provinceMap.get(id);
                String city = cities.get(0);
                List<String> cityAreas = cityMap.get(city);
                Location location = new Location();
                location.setCities(cities);
                location.setAreas(cityAreas);
                JSONObject jsonObject = JSONObject.fromObject(location);
                resp.getWriter().write(jsonObject.toString());
                break;
        }

    }
}
```

```java
package com.southwind.entity;

import java.util.List;

public class Location {
    private List<String> cities;
    private List<String> areas;

    public List<String> getCities() {
        return cities;
    }

    public void setCities(List<String> cities) {
        this.cities = cities;
    }

    public List<String> getAreas() {
        return areas;
    }

    public void setAreas(List<String> areas) {
        this.areas = areas;
    }
}
```

JDBC

Java DataBase Connectivity 是一个独立于特定数据库的管理系统，通用的 SQL 数据库存取和操作的公共接口。

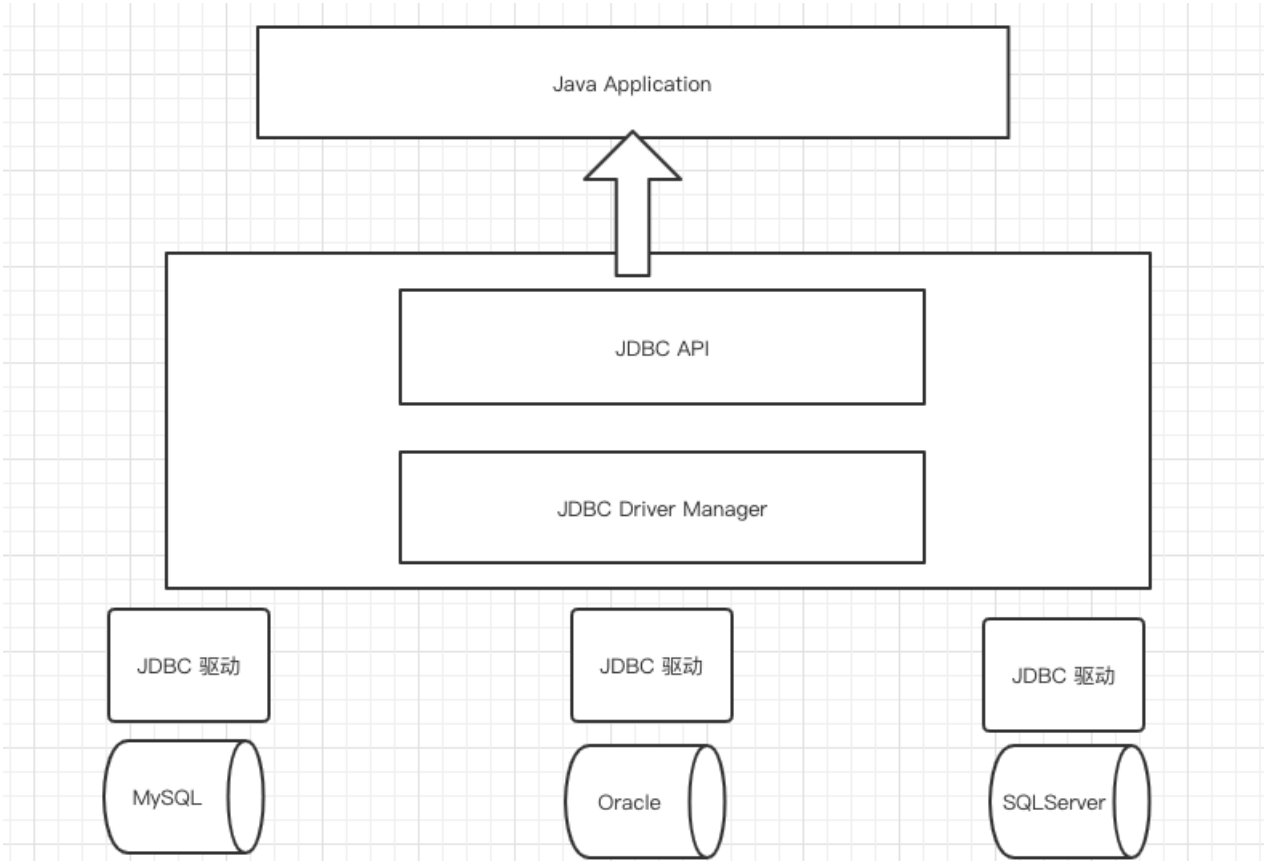
定义了一组标准，为访问不同数据库提供了统一的途径。



JDBC 体系结构

JDBC 接口包括两个层面：

- 面向应用的 API，供程序员调用
- 面向数据库的 API，供厂商开发数据库的驱动程序



JDBC API

提供者：Java 官方

内容：供开发者调用的接口

java.sql 和 javax.sql

- DriverManager 类
- Connection 接口
- Statement 接口
- ResultSet 接口

DriverManager

提供者：Java 官方

作用：管理不同的 JDBC 驱动

JDBC 驱动

提供者：数据库厂商

作用：负责连接不同的数据库

JDBC 的使用

1、加载数据库驱动，Java 程序和数据库之间的桥梁。

2、获取 Connection，Java 程序与数据库的一次连接。

3、创建 Statement 对象，由 Connection 产生，执行 SQL 语句。

4、如果需要接收返回值，创建 ResultSet 对象，保存 Statement 执行之后所查询到的结果。

```
package com.southwind.test;

import java.sql.*;
import java.util.Date;

public class Test {
    public static void main(String[] args) {
        try {
            //加载驱动
            Class.forName("com.mysql.cj.jdbc.Driver");
            //获取连接
            String url = "jdbc:mysql://localhost:3306/test?
useUnicode=true&characterEncoding=UTF-8";
            String user = "root";
            String password = "root";
            Connection connection =
DriverManager.getConnection(url,user,password);
//            String sql = "insert into student(name,score,birthday) values('李
四',78,'2019-01-01')";
//            String sql = "update student set name = '李四'";
//            String sql = "delete from student";
//            Statement statement = connection.createStatement();
//            int result = statement.executeUpdate(sql);

            String sql = "select * from student";
            Statement statement = connection.createStatement();
            ResultSet resultSet = statement.executeQuery(sql);
            while (resultSet.next()){
                Integer id = resultSet.getInt("id");
                String name = resultSet.getString(2);
                Double score = resultSet.getDouble(3);
                Date date = resultSet.getDate(4);
                System.out.println(id+"-"+name+"-"+score+"-"+date);
            }
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (SQLException e){
            e.printStackTrace();
        }
    }
}
```

PreparedStatement

Statement 的子类，提供了 SQL 占位符的功能

使用 Statement 进行开发有两个问题：

1、需要频繁拼接 String 字符串，出错率较高。

2、存在 SQL 注入的风险。

SQL 注入：利用某些系统没有对用户输入的信息进行充分检测，在用户输入的数据中注入非法的 SQL 语句，从而利用系统的 SQL 引擎完成恶意行为的做法。

```
String url = "jdbc:mysql://localhost:3306/test?
useUnicode=true&characterEncoding=UTF-8";
String user = "root";
String password = "root";
Connection connection = DriverManager.getConnection(url,user,password);
String username = "lisi";
String mypassword = "000";
String sql = "select * from t_user where username = ? and password = ?";
System.out.println(sql);
PreparedStatement preparedStatement = connection.prepareStatement(sql);
preparedStatement.setString(1,username);
preparedStatement.setString(2,mypassword);
ResultSet resultSet = preparedStatement.executeQuery();
if(resultSet.next()){
  System.out.println("登录成功");
}else{
  System.out.println("登录失败");
}
} catch (ClassNotFoundException e) {
  e.printStackTrace();
} catch (SQLException e){
  e.printStackTrace();
}
```

数据库连接池

JDBC 开发流程

- 加载驱动（只需要加载一次）
- 建立数据库连接（Connection）
- 执行 SQL 语句（Statement）
- ResultSet 接收结果集（查询）
- 断开连接，释放资源

数据库连接对象是通过 DriverManager 来获取的，每次获取都需要向数据库申请获取连接，验证用户名和密码，

执行完 SQL 语句后断开连接，这样的方式会造成资源的浪费，数据连接资源没有得到很好的重复利用。

可以使用数据库连接池解决这一问题。

数据库连接池的基本思想就是为数据库建立一个缓冲池，预先向缓冲池中放入一定数量的连接对象，当需要获取数据库连接的时候，只需要从缓冲池中取出一个对象，用完之后再放回到缓冲池中，供下一次请求使用，做到了资源的重复利用，允许程序重复使用一个现有的数据库连接对象，而不需要重新创建。

当数据库连接池中没有空闲的连接时，新的请求就会进入等待队列，等待其他线程释放连接。

数据库连接池实现

JDBC 的数据库连接池使用 javax.sql.DataSource 接口来完成的，DataSource 是 Java 官方提供的接口，使用的时候开发者并不需要自己来实现该接口，可以使用第三方的工具，C3P0 是一个常用的第三方实现，实际开发中直接使用 C3P0 即可完成数据库连接池的操作。

1、导入 jar 包。

传统方式拿到的 Connection

```
com.mysql.cj.jdbc.ConnectionImpl@557caf28
```

C3P0 拿到的 Connection

```
com.mchange.v2.c3p0.impl.NewProxyConnection@4988d8b8
```

2、代码实现

```
package com.southwind.test;

import com.mchange.v2.c3p0.ComboPooledDataSource;
```

```
import java.beans.PropertyVetoException;
import java.sql.Connection;
import java.sql.SQLException;

public class DataSourceTest {
    public static void main(String[] args) {
        try {
            //创建C3P0
            ComboPooledDataSource dataSource = new ComboPooledDataSource();
            dataSource.setDriverClass("com.mysql.cj.jdbc.Driver");
            dataSource.setJdbcUrl("jdbc:mysql://localhost:3306/test?
useUnicode=true&characterEncoding=UTF-8");
            dataSource.setUser("root");
            dataSource.setPassword("root");
            Connection connection = dataSource.getConnection();
            System.out.println(connection);
            //还回到数据库连接池中
            connection.close();
        } catch (PropertyVetoException e) {
            e.printStackTrace();
        } catch (SQLException e){
            e.printStackTrace();
        }
    }
}
```

实际开发，将 C3P0 的配置信息定义在 xml 文件中，Java 程序只需要加载配置文件即可完成数据库连接池的初始化操作。

1、配置文件的名字必须是 c3p0-config.xml

2、初始化 ComboPooledDataSource 时，传入的参数必须是 c3p0-config.xml 中 named-config 标签的 name 属性值。

```
<?xml version="1.0" encoding="UTF-8"?>
<c3p0-config>

    <named-config name="testc3p0">

        <!-- 指定连接数据源的基本属性 -->
        <property name="user">root</property>
        <property name="password">root</property>
        <property name="driverClass">com.mysql.jdbc.Driver</property>
        <property name="jdbcUrl">jdbc:mysql://localhost:3306/library?
useUnicode=true&amp;characterEncoding=UTF-8</property>

        <!-- 若数据库中连接数不足时，一次向数据库服务器申请多少个连接 -->
```

```
        <property name="acquireIncrement">5</property>
        <!-- 初始化数据库连接池时连接的数量 -->
        <property name="initialPoolSize">20</property>
        <!-- 数据库连接池中的最小的数据库连接数 -->
        <property name="minPoolSize">2</property>
        <!-- 数据库连接池中的最大的数据库连接数 -->
        <property name="maxPoolSize">40</property>

    </named-config>

</c3p0-config>
```

```
package com.southwind.test;

import com.mchange.v2.c3p0.ComboPooledDataSource;

import java.beans.PropertyVetoException;
import java.sql.Connection;
import java.sql.SQLException;

public class DataSourceTest {
    public static void main(String[] args) {
        try {
            //创建C3P0
            ComboPooledDataSource dataSource = new
ComboPooledDataSource("testc3p0");
            Connection connection = dataSource.getConnection();
            System.out.println(connection);
            //还回到数据库连接池中
            connection.close();
        } catch (SQLException e){
            e.printStackTrace();
        }
    }
}
```

DBUtils

DBUtils 可以帮助开发者完成数据的封装（结果集到 Java 对象的映射）

1、导入 jar 包

ResultHandler 接口是用来处理结果集，可以将查询到的结果集转换成 Java 对象，提供了 4 种实现类。

- BeanHandler 将结果集映射成 Java 对象 Student
- BeanListHandler 将结果集映射成 List 集合 List
- MapHandler 将结果集映射成 Map 对象
- MapListHandler 将结果集映射成 MapList 结合

```
public static Student findByDBUtils(Integer id){
  Connection connection = null;
  Student student = null;
  try {
    connection = dataSource.getConnection();
    String sql = "select * from student";
    QueryRunner queryRunner = new QueryRunner();
    List<Map<String,Object>> list = queryRunner.query(connection,sql,new
MapListHandler());
    for (Map<String,Object> map:list){
      System.out.println(map);
    }
  } catch (SQLException e) {
    e.printStackTrace();
  } finally {
    try {
      connection.close();
    } catch (SQLException e) {
      e.printStackTrace();
    }
  }
  return student;
}
```