# Request请求

**【学习目标】理解、了解、应用、记忆**

通过今天的学习，参训学员能够：（解释的时候说出二级目标的掌握程度）

1. **【应用】能够使用request对象**
2. 能够使用request获取请求行信息和ip地址
3. 能够使用request获取头信息
4. 能够处理request请求参数乱码
5. 能够熟练运用request容器存取删
6. 熟练掌握request请求转发
7. **【应用】能够使用BeanUtils独立完成注册案例**

# request对象介绍（方向：浏览器——> 服务器）

通过前面的servlet学习，我们了解的使用java servlet小程序处理请求和响应的过程中有request对象和response对象帮助我们处理请求响应

那么今天，我们先从请求request对象入手，开始学习：



Request对象学习的思路：

1 如果获取请求 行 头 体

2 请求中文处理

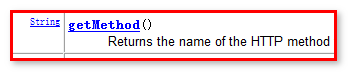
3 请求对象的其他常用方法

## request获得请求行信息和ip地址

请求行格式:

IMG_256

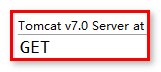
**获得请求方法的类型: 如get 或 post**



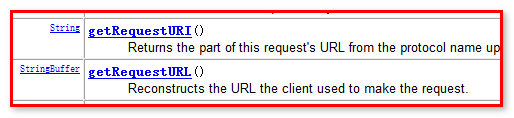
演示代码：

|  |
| --- |
| System.out.println(request.getMethod()); |

效果：



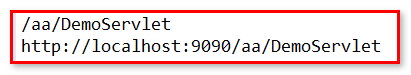
**请求的路径:**



代码演示：

|  |
| --- |
| System.out.println(request.getRequestURI());  System.out.println(request.getRequestURL()); |

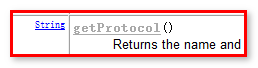
效果：



uri: 统一资源标识符,用来标识一个资源,资源路径。(相当于身份证)

url: 统一资源定位符,是一种具体的URI,可以用来标识一个资源.并且指明了如何定位一个资源.（相当于身份证中的地址）

**获取请求的协议类型: 如 http/1.1**



|  |
| --- |
| System.out.println(request.getProtocol()); |

效果：

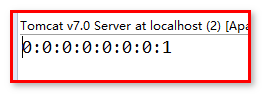


 其他细节：

**获取IP地址**

IMG_260

访问地址：<http://localhost:9090/aa/DemoServlet>



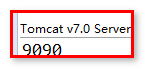
访问地址：<http://127.0.0.1:9090/aa/DemoServlet>



注意，正在服务器解析的时候，效果虽然一致，但是，底层具体内容不一样

**获取服务器的端口号**

IMG_261



## request获取头信息

Accept: text/html, application/xhtml+xml, \*/\*

Referer: http://localhost/day19/post.html

Accept-Language: zh-CN

User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; qdesk 2.4.1272.203; Windows NT 6.1; WOW64; Trident/6.0)

Content-Type: application/x-www-form-urlencoded

Accept-Encoding: gzip, deflate

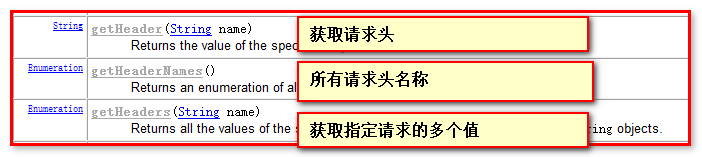
Host: localhost

Content-Length: 30

DNT: 1

Connection: Keep-Alive

Cache-Control: no-cache



需求：获取所有的请求头信息

代码演示：

|  |
| --- |
| **package** cn.igeek.web;  **import** java.io.IOException;  **import** java.util.Enumeration;  **import** javax.servlet.ServletException;  **import** javax.servlet.http.HttpServlet;  **import** javax.servlet.http.HttpServletRequest;  **import** javax.servlet.http.HttpServletResponse;  **public** **class** DemoServlet **extends** HttpServlet {  **public** **void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  Enumeration<String> names = request.getHeaderNames();  **while**(names.hasMoreElements()){  String string = names.nextElement();  System.***out***.println(string+":"+request.getHeader(string));  }  }  **public** **void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  doGet(request, response);  }  } |

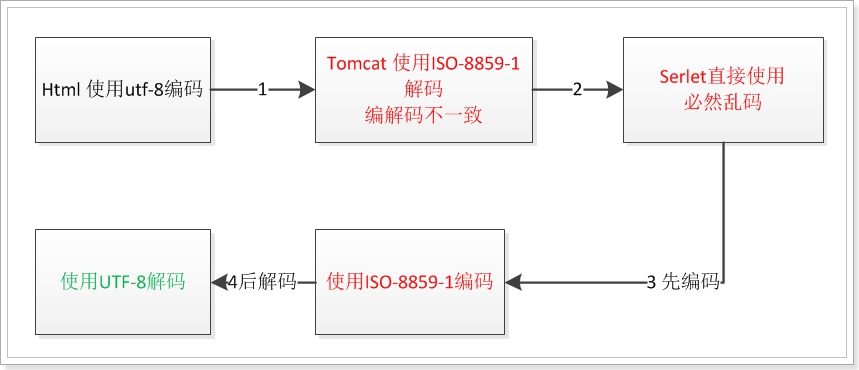
效果：

|  |
| --- |
| host:127.0.0.1:9090  connection:keep-alive  cache-control:max-age=0  user-agent:Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.3163.100 Safari/537.36  upgrade-insecure-requests:1  accept:text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,\*/\*;q=0.8  accept-encoding:gzip, deflate, br  accept-language:zh-CN,zh;q=0.8 |

如果输入中文，你会发现乱码，是不是很不爽？

## 请求参数乱码处理(重点)：

乱码产生的原因：



### Get 提交的中文乱码解决：

<h1>GET方式</h1>

<form method="get" action="/day19/request3">

帐号：<input type="text" name="username"/><br/>

<input type="submit" value="提交"/>

</form>

#### 第一种方案：[修改tomcat默认的编码方式](http://blog.csdn.net/numbibi/article/details/7987883)（不推荐）

默认情况下，tomcat使用的的编码方式：iso-8859-1

修改tomcat下的conf/server.xml文件

找到如下代码：      
<Connector port="8080" protocol="HTTP/1.1" connectionTimeout="20000" redirectPort="8443" />  
这段代码规定了Tomcat监听HTTP请求的端口号等信息。

可以在这里添加一个属性：URIEncoding，将该属性值设置为UTF-8，即可让Tomcat（默认ISO-8859-1编码）以UTF-8的编码处理get请求。

修改完成后：

<Connector port="8080"  protocol="HTTP/1.1" connectionTimeout="20000" redirectPort="8443" URIEncoding="UTF-8" />

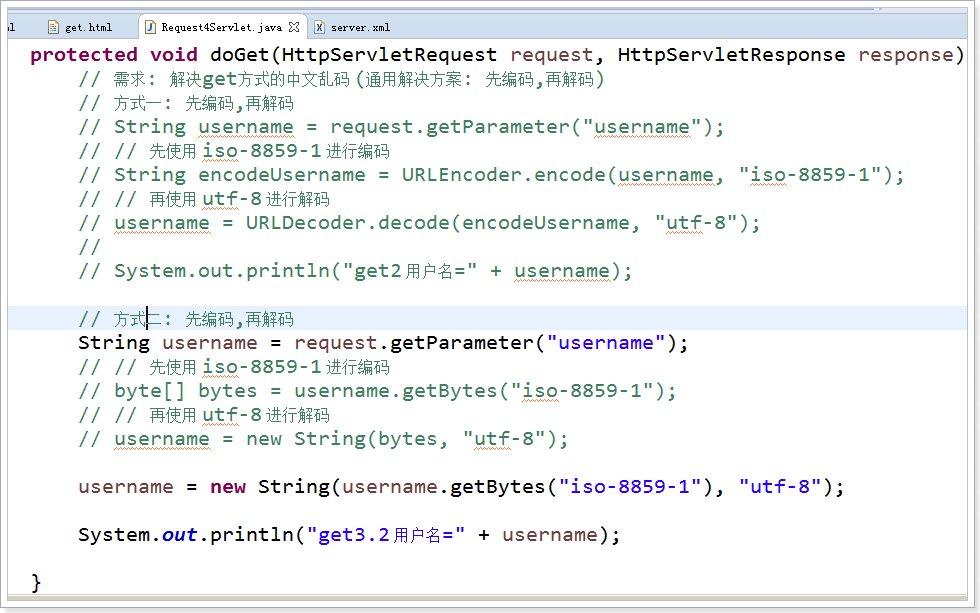
缺点:

方法不是很常用. 修改tomcat的servlet.xml

会很死板.

如果两个项目: 一个是UTF-8,  另一个是GBK

#### 第二种方案：先编码再解码(重点)



### POST提交的中文乱码解决：

<h1>POST方式</h1>

<form method="post" action="/day19/request3">

城市：<input type="text" name="city"/><br/>

<input type="submit" value="提交"/>

</form>

#### 第一种解决方案 先编码再解码(了解)：



#### 第二种解决方案 设置请求编码(重点)：

这种方式只对 请求体 有效，算是post的偷懒方式,开发时最常用



## Request容器（存取删）（域对象）和请求转发（重点）

需求：

使用一个servlet专门处理获取请求参数，比对用户名和密码，如果，成功（登陆成功），那么跳转下一个servlet，在下一个servlet响应页面结果！

请求转发：从一个servlet中启动另一个servlet。客户端浏览器地址栏不会变化，这个请求转发操作，全部在服务器内部执行。

代码演示：演示转发，重点——request.getRequestDispatcher("/my").forward(request, response);

|  |
| --- |
| package cn.igeek.web;  import java.io.IOException;  import javax.servlet.ServletException;  import javax.servlet.http.HttpServlet;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;  public class DemoServlet extends HttpServlet {  public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {    String name = request.getParameter("name");  String pwd = request.getParameter("pwd");  if(name.equals("tom") && "111".equals(pwd)){  request.getRequestDispatcher("/my").forward(request, response);  }  }  public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  doGet(request, response);  }  } |

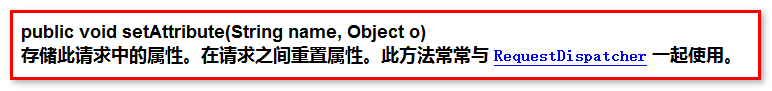
|  |
| --- |
| package cn.igeek.web;  import java.io.IOException;  import java.lang.reflect.Field;  import java.util.Map;  import java.util.Set;  import javax.servlet.ServletConfig;  import javax.servlet.ServletContext;  import javax.servlet.ServletException;  import javax.servlet.http.HttpServlet;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;  public class MyServlet extends HttpServlet {  public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  response.getWriter().write("<h1><font color=\"green\">login OK!!!!</font></h1>");  }  public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  doGet(request, response);  }  } |

效果：

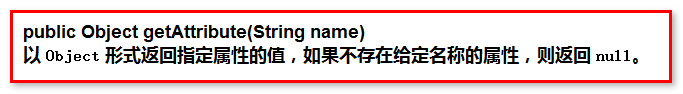


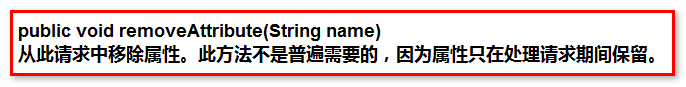
那么如果需要显示当前用户用户名，怎么办？

使用请求对象将获取到的请求参数保存下来：



可以保存以外，还可以，获取和删除：





那么，我们可以修改成：

|  |
| --- |
| package cn.igeek.web;  import java.io.IOException;  import javax.servlet.ServletException;  import javax.servlet.http.HttpServlet;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;  public class DemoServlet extends HttpServlet {  public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {    String name = request.getParameter("name");  String pwd = request.getParameter("pwd");  if(name.equals("tom") && "111".equals(pwd)){  request.setAttribute("name", name);  request.getRequestDispatcher("/my").forward(request, response);  }  }  public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  doGet(request, response);  }  } |

|  |
| --- |
| package cn.igeek.web;  import java.io.IOException;  import java.lang.reflect.Field;  import java.util.Map;  import java.util.Set;  import javax.servlet.ServletConfig;  import javax.servlet.ServletContext;  import javax.servlet.ServletException;  import javax.servlet.http.HttpServlet;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;  public class MyServlet extends HttpServlet {  public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  String name = (String)request.getAttribute("name");    response.getWriter().write("<h1><font color=\"green\">login "+name+" OK!!!!</font></h1>");  }  public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  doGet(request, response);  }  } |

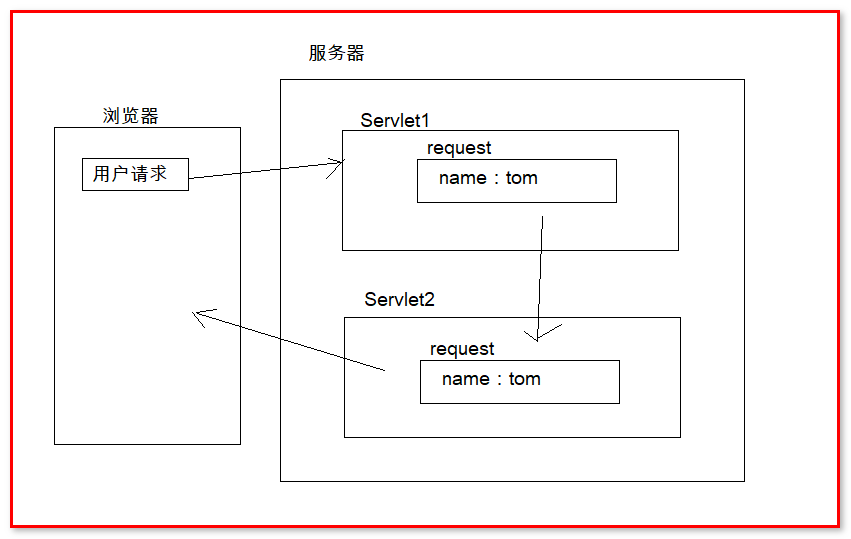
效果:



总结：

当一个Web资源收到客户端的请求后，如果希望服务器通知另外一个资源处理.

可以通过 转发对象 RequestDispatcher 对象的forward（request,response）方法,将当前请求传递给其他的Web资源进行处理，这种方式称为请求转发。



**在这些转发的过程中，所有的Servlet共享同一个请求对象。**

**在转发中，客户端是感觉不到服务器内部在跳转。而且客户端的浏览器的地址栏中是不会发生任何变化的。**

因为在多个Servlet中可以进行转发，导致多个Servlet之间共享同一个request对象，于是在发出转发的Servlet中，可以把request对象当做一个容器，然后给其中保存数据，在其他的Servlet中可以取出前面的Servlet给request对象中保存的数据。

request对象如果当做容器的话，它只是在当前这次请求中有效。当请求响应结束了，这个容器就消失了。

总结：

1 使用request对象，可以获取请求行

2 使用request对象，可以获取请求头

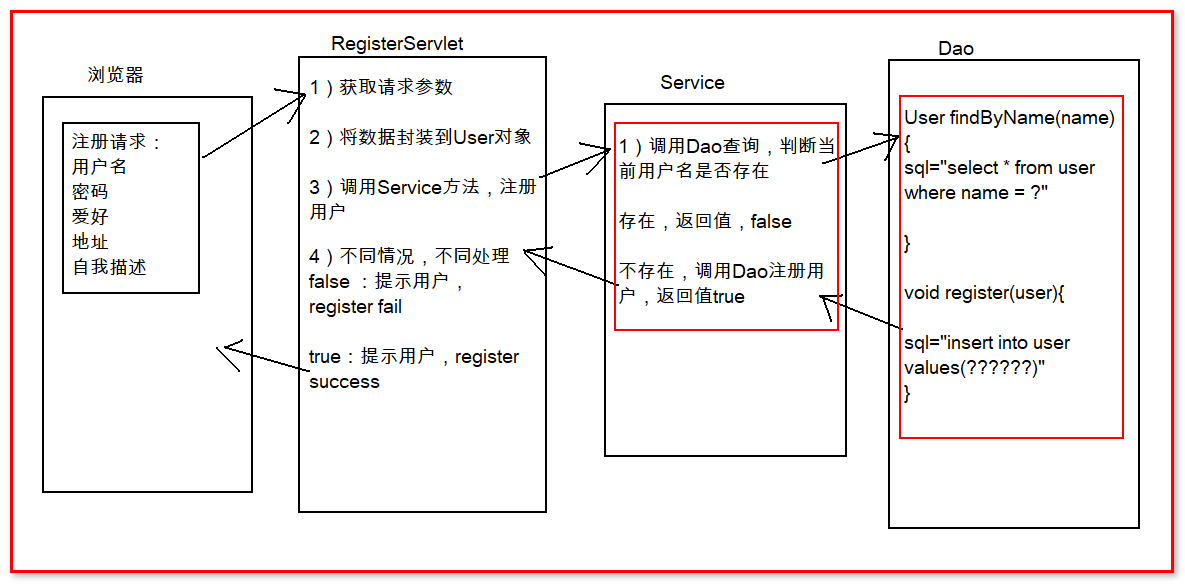
3 使用request对象，可以处理中文乱码

4 使用request对象，调用下一个servlet（请求转发）

5 使用request对象，在一个请求转发过程中，让两个servlet共享数据。

# 注册案例

## 分析

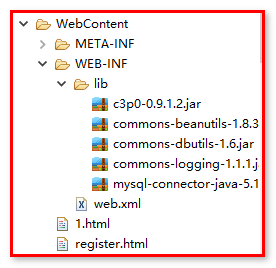


## 实现

### 页面 入口



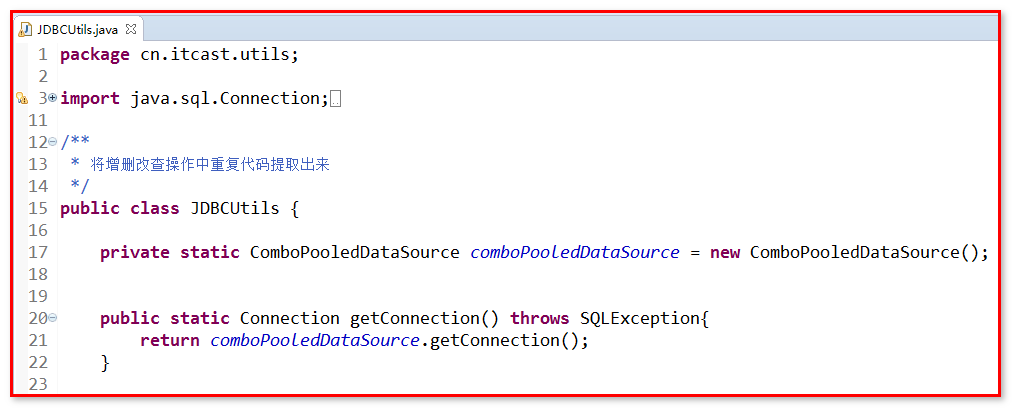
1. 导入jar包



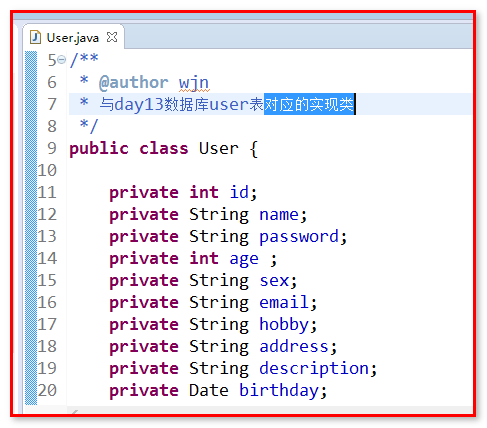
2）配置文件



1. 准备工具类



1. 准备封装数据的实体类（JavaBean 模型）



### web层

在web层，我们获取数据，要进行封装的时候，数据太多了，一次次调用set方法太过麻烦，在这里我们介绍一个技术，给我解决问题：

**BeanUtils：**

学习新技术的思路：

1 百度新技术 获取官网

2 下载资料（jar包 API文档）

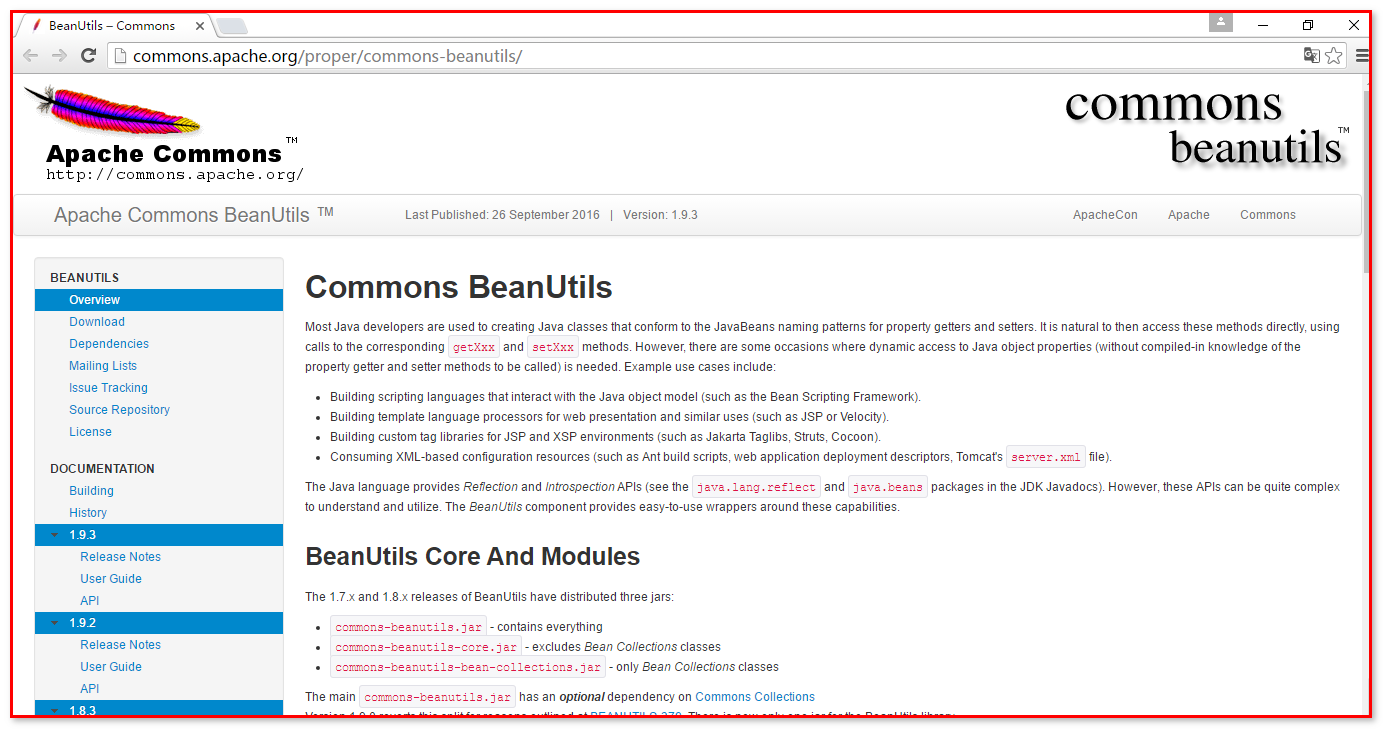
3 创建工程，测试新技术，需要查找到官网的用户指南（user guide，quick start）

4 根据用户指南测试技术

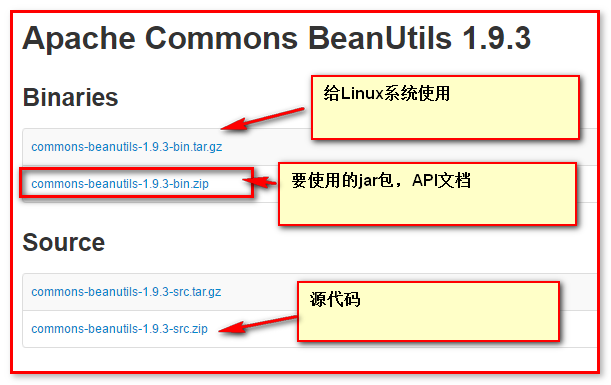
5 笔记，后期，在使用这个技术的是，可以看自己的笔记，效率更高，面试的时候，可以将笔记，作为一个加分项。

作用：帮助程序员更便捷的封装数据到对象中。

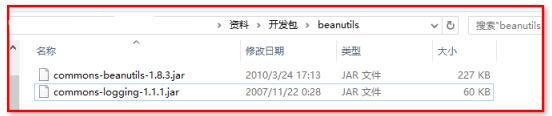
学习新技术第一步：查找官网：<http://commons.apache.org/proper/commons-beanutils/> 找到download下载



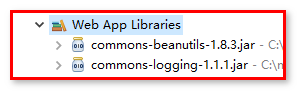
第二步下载资源：



Apache提供的很多技术之间通过都会有jar包的相互依赖。

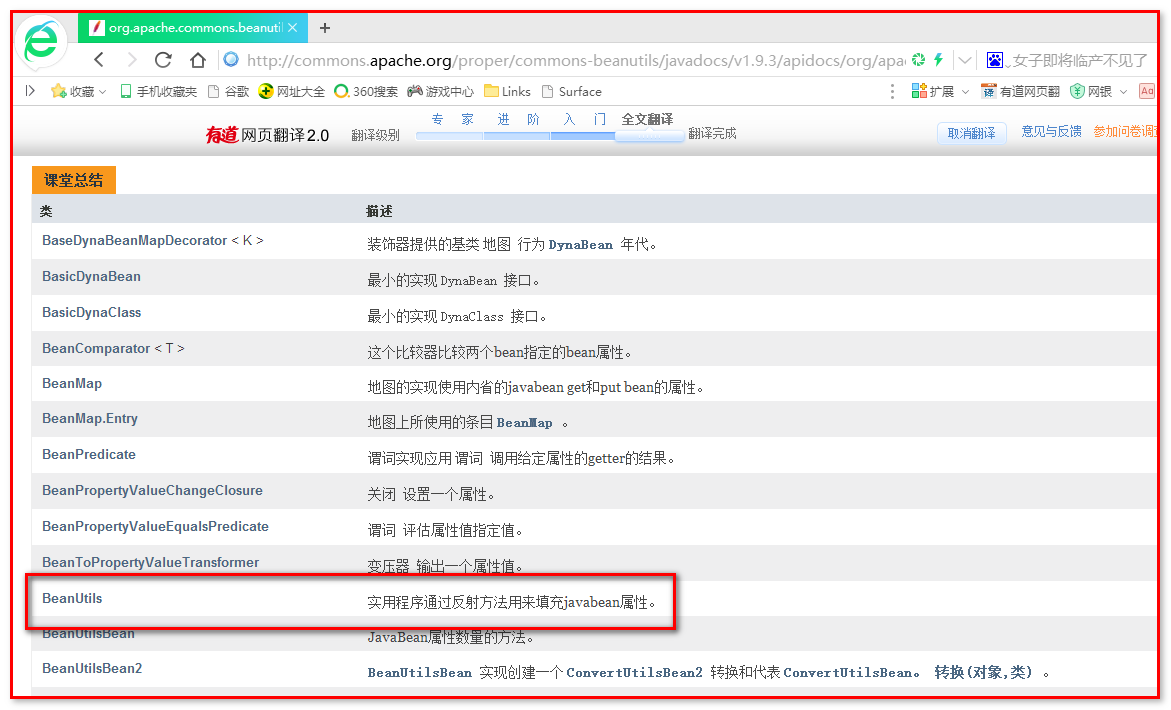


第三步复制到当前项目的WEB-INF下的lib中。

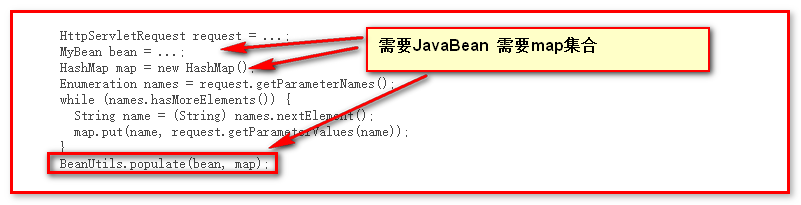


1.2.核心对象BeanUtils类使用

官方API：要使用的核心对象



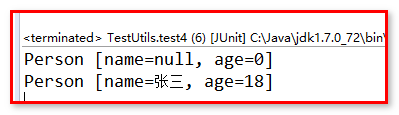
官方提供的例子：



代码演示：

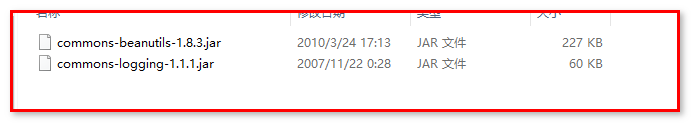
|  |
| --- |
| /\*\*  \* 使用apache公司的beanUtils来封装数据  \*  \* **@throws** Exception  \*/  @Test  **public** **void** test4() **throws** Exception{  Map<String,Object > map = **new** HashMap<String,Object >();  map.put("name", "张三");  map.put("age", 18);    Person p = **new** Person();  System.***out***.println(p);    org.apache.commons.beanutils.BeanUtils.*populate*(p, map);  System.***out***.println(p);  } |

效果：



## BeanUtils的使用总结

1、需要导入两个jar包



2、使用BeanUtils类里面的populate方法

第一参数：数据要封装到的对象（Person）

第二个参数：要被封装数据，这个数据存放在一个map集合中

最后就会把map中的数据，封装到bean对象中。

应用：

|  |
| --- |
| package cn.igeek.web;  import java.io.IOException;  import java.lang.reflect.InvocationTargetException;  import java.util.Map;  import javax.servlet.ServletException;  import javax.servlet.http.HttpServlet;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;  import org.apache.commons.beanutils.BeanUtils;  import cn.igeek.domain.User;  import cn.igeek.service.UserService;  public class RegisterServlet extends HttpServlet {  public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  request.setCharacterEncoding("utf-8");  Map<String, String[]> map = request.getParameterMap();  User user = new User();  try {  BeanUtils.populate(user, map);  } catch (Exception e) {  e.printStackTrace();  }  System.out.println(user);  String[] values = request.getParameterValues("hobby");  String hobby = "";  for (String string : values) {  hobby = hobby +","+string;  }  hobby = hobby.substring(1);  user.setHobby(hobby);  UserService service = new UserService();  boolean info = service.register(user);  if(info){  response.getWriter().write("register OK!!!");  }else{  response.getWriter().write("register error!!!");  }    }  public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  doGet(request, response);  }  } |

### 业务层

|  |
| --- |
| package cn.igeek.service;  import cn.igeek.dao.UserDao;  import cn.igeek.domain.User;  public class UserService {  private UserDao userdDao = new UserDao();  public boolean register(User user) {  User findByname = userdDao.findByName(user.getName());  if(findByname == null){  userdDao.register(user);  return true;  }else{  return false;  }  }  } |

### 数据层

|  |
| --- |
| package cn.igeek.dao;  import java.sql.SQLException;  import java.util.ArrayList;  import java.util.List;  import org.apache.commons.dbutils.QueryRunner;  import org.apache.commons.dbutils.handlers.BeanHandler;  import cn.igeek.domain.User;  import cn.igeek.utils.JDBCUtils;  public class UserDao {  private QueryRunner qr = new QueryRunner(JDBCUtils.getDataSource());  public User findByName(String name) {  String sql = "select \* from user where name = ?";  try {  return qr.query(sql, new BeanHandler<User>(User.class), name);  } catch (SQLException e) {  e.printStackTrace();  return null;  }  }  public void register(User user) {  String sql = "insert into user values(null,?,?,?,?,?,?,?,?,?)";  List<Object> list= new ArrayList<>();  list.add(user.getName());  list.add(user.getPassword());  list.add(user.getAge());    list.add(user.getSex());  list.add(user.getEmail());  list.add(user.getHobby());    list.add(user.getAddress());  list.add(user.getDescription());  list.add(user.getBirthday());    try {  qr.update(sql, list.toArray());  } catch (SQLException e) {  e.printStackTrace();  }  }  } |

注意：注册的时候必须使用日期格式—— yyyy-MM-dd（BeanUtils默认支持的日期转换格式）

