Servlet

课前默写

1、写出对数据库表t_student(id,sname,sage,ssex)
的新增、修改、删除、和分页查询的SQL语句
2、请写出HTML常用的表单标签
3、写出XML语法
4、写出Dom4J解析的步骤
5、写出SAX解析的步骤
6、写出JSON语法
7、写出JSON的三种解析

课程回顾

1、Web项目结构
2、Tomcat的配置
3、XML语法
4、Dom4J解析
5、SAX解析

今日内容

1、Http协议
2、Servlet入门
3、Servlet获取请求参数
4、Servlet实现页面跳转
5、Servlet的生命周期
6、Servlet的三种创建方式
7、Servlet的两种配置方式
8、ServletContext的使用
9、Servlet线程安全

教学目标

1、掌握Servlet的基本使用
2、熟悉Http协议
3、掌握Servlet获取请求参数
4、掌握Servlet实现页面跳转
5、掌握Servlet的生命周期
6、掌握Servlet的三种创建方式
7、掌握Servlet的两种配置方式
8、掌握ServletContext的使用
9、掌握Servlet线程安全

第四章 HTTP协议回顾

4.1 HTTP协议简介

千锋JAVA教学部

HTTP是一个属于应用层的面向对象的协议，由于其简捷、快速的方式，适用于分布式超媒体信息系统。它于1990年提出，经过几年的使用与发展，得到不断地完善和扩展。目前在WWW中使用的是HTTP/1.0的第六版，HTTP/1.1的规范化工作正在进行之中，而且HTTP-NG(Next Generation of HTTP)的建议已经提出。

HTTP协议的主要特点可概括如下：

1.支持客户/服务器模式。

2.简单快速：客户向服务器请求服务时，只需传送请求方法和路径。请求方法常用的有GET、HEAD、POST。每种方法规定了客户与服务器联系的类型不同。由于HTTP协议简单，使得HTTP服务器的程序规模小，因而通信速度很快。

3.灵活：HTTP允许传输任意类型的数据对象。正在传输的类型由Content-Type加以标记。

4.无连接：无连接的含义是限制每次连接只处理一个请求。服务器处理完客户的请求，并收到客户的应答后，即断开连接。采用这种方式可以节省传输时间。

5.无状态：HTTP协议是无状态协议。无状态是指协议对于事务处理没有记忆能力。缺少状态意味着如果后续处理需要前面的信息，则它必须重传，这样可能导致每次连接传送的数据量增大。另一方面，在服务器不需要先前信息时它的应答就较快。

4.2 HTTP协议详解之URL篇

http（超文本传输协议）是一个基于请求与响应模式的、无状态的、应用层的协议，常基于TCP的连接方式，HTTP1.1版本中给出一种持续连接的机制，绝大多数的Web开发，都是构建在HTTP协议之上的Web应用。

HTTP URL (URL是一种特殊类型的URI，包含了用于查找某个资源的足够的信息)的格式如下：
[http://host[":"port]abs_path] http表示要通过HTTP协议来定位网络资源；host表示合法的Internet主机域名或者IP地址；port指定一个端口号，为空则使用缺省端口80；abs_path指定请求资源的URI；如果URL中没有给出abs_path，那么当它作为请求URI时，必须以“/”的形式给出，通常这个工作浏览器自动帮我们完成。

4.3 HTTP协议详解之请求篇

http请求由三部分组成，分别是：请求行、消息报头、请求正文

1、请求行以一个方法符号开头，以空格分开，后面跟着请求的URI和协议的版本，格式如下：Method Request-URI HTTP-Version CRLF
其中 Method表示请求方法；Request-URI是一个统一资源标识符；HTTP-Version表示请求的HTTP协议版本；CRLF表示回车和换行（除了作为结尾的CRLF外，不允许出现单独的CR或LF字符）。

请求方法（所有方法全为大写）有多种，各个方法的解释如下： GET 请求获取Request-URI所标识的资源 POST 在Request-URI所标识的资源后附加新的数据 HEAD 请求获取由Request-URI所标识的资源的响应消息报头 PUT 请求服务器存储一个资源，并用Request-URI作为其标识 DELETE 请求服务器删除Request-URI所标识的资源 TRACE 请求服务器回送收到的请求信息，主要用于测试或诊断 CONNECT 保留将来使用 OPTIONS 请求查询服务器的性能，或者查询与资源相关的选项和需求 应用举例： GET方法：在浏览器的地址栏中输入网址的方式访问网页时，浏览器采用GET方法向服务器获取资源，eg:GET /form.html HTTP/1.1 (CRLF)

POST方法要求被请求服务器接受附在请求后面的数据，常用于提交表单。 eg：POST /reg.jsp HTTP/ (CRLF) Accept:image/gif,image/x-xbit,... (CRLF) Content-Length:22 (CRLF) Connection:Keep-Alive (CRLF) Cache-Control:no-cache (CRLF) (CRLF) //该CRLF表示消息报头已经结束，在此之前为消息报头 user=jeffrey&pwd=1234 //此行以下为提交的数据

千锋JAVA教学部

HEAD方法与GET方法几乎是一样的，对于HEAD请求的回应部分来说，它的HTTP头部中包含的信息与通过GET请求所得到的信息是相同的。利用这个方法，不必传输整个资源内容，就可以得到Request-URI所标识的资源的信息。该方法常用于测试超链接的有效性，是否可以访问，以及最近是否更新。 2、请求报头后述 3、请求正文(略)

4.4 HTTP协议详解之响应篇

在接收和解释请求消息后，服务器返回一个HTTP响应消息。

HTTP响应也是由三个部分组成，分别是：状态行、消息报头、响应正文 1、状态行格式如下： HTTP-Version Status-Code Reason-Phrase CRLF 其中，HTTP-Version表示服务器HTTP协议的版本；Status-Code表示服务器发回的响应状态代码；Reason-Phrase表示状态代码的文本描述。 状态代码有三位数字组成，第一个数字定义了响应的类别，且有五种可能取值： 1xx：指示信息--表示请求已接收，继续处理 2xx：成功--表示请求已被成功接收、理解、接受 3xx：重定向--要完成请求必须进行更进一步的操作 4xx：客户端错误--请求有语法错误或请求无法实现 5xx：服务器端错误--服务器未能实现合法的请求 常见状态代码、状态描述、说明： 200 OK //客户端请求成功 400 Bad Request //客户端请求有语法错误，不能被服务器所理解 401 Unauthorized //请求未经授权，这个状态代码必须和WWW-Authenticate报头域一起使用 403 Forbidden //服务器收到请求，但是拒绝提供服务 404 Not Found //请求资源不存在，eg：输入了错误的URL 500 Internal Server Error //服务器发生不可预期的错误 503 Server Unavailable //服务器当前不能处理客户端的请求，一段时间后可能恢复正常 eg：HTTP/1.1 200 OK （CRLF)

2、响应报头后述

3、响应正文就是服务器返回的资源的内容

4.5 HTTP协议详解之消息报头篇

HTTP消息由客户端到服务器的请求和服务器到客户端的响应组成。请求消息和响应消息都是由开始行（对于请求消息，开始行就是请求行，对于响应消息，开始行就是状态行），消息报头（可选），空行（只有CRLF的行），消息正文（可选）组成。

HTTP消息报头包括普通报头、请求报头、响应报头、实体报头。 每一个报头域都是由名字+“：”+空格+值 组成，消息报头域的名字是大小写无关的。

1、普通报头 在普通报头中，有少数报头域用于所有的请求和响应消息，但并不用于被传输的实体，只用于传输的消息。 eg： Cache-Control 用于指定缓存指令，缓存指令是单向的（响应中出现的缓存指令在请求中未必会出现），且是独立的（一个消息的缓存指令不会影响另一个消息处理的缓存机制），HTTP1.0使用的类似的报头域为Pragma。 请求时的缓存指令包括：no-cache（用于指示请求或响应消息不能缓存）、no-store、max-age、max-stale、min-fresh、only-if-cached; 响应时的缓存指令包括：public、private、no-cache、no-store、no-transform、must-revalidate、proxy-revalidate、max-age、s-maxage. eg：为了指示IE浏览器（客户端）不要缓存页面，服务器端的JSP程序可以编写如下：response.sehHeader("Cache-Control","no-cache"); //response.setHeader("Pragma","no-cache");作用相当于上述代码，通常两者//合用 这句代码将在发送的响应消息中设置普通报头域：Cache-Control:no-cache

Date普通报头域表示消息产生的日期和时间

Connection普通报头域允许发送指定连接的选项。例如指定连接是连续，或者指定“close”选项，通知服务器，在响应完成后，关闭连接

千锋JAVA教学部

2、请求报头 请求报头允许客户端向服务器端传递请求的附加信息以及客户端自身的信息。 常用的请求报头 Accept Accept请求报头域用于指定客户端接受哪些类型的信息。eg：Accept：image/gif，表明客户端希望接受GIF图象格式的资源；Accept：text/html，表明客户端希望接受html文本。 Accept-Charset Accept-Charset请求报头域用于指定客户端接受的字符集。eg：Accept-Charset:iso-8859-1,gb2312.如果在请求消息中没有设置这个域，缺省是任何字符集都可以接受。 Accept-Encoding Accept-Encoding请求报头域类似于Accept，但是它是用于指定可接受的内容编码。eg：Accept-Encoding:gzip.deflate.如果请求消息中没有设置这个域服务器假定客户端对各种内容编码都可以接受。 Accept-Language Accept-Language请求报头域类似于Accept，但是它是用于指定一种自然语言。eg：Accept-Language:zh-cn.如果请求消息中没有设置这个报头域，服务器假定客户端对各种语言都可以接受。 Authorization Authorization请求报头域主要用于证明客户端有权查看某个资源。当浏览器访问一个页面时，如果收到服务器的响应代码为401（未授权），可以发送一个包含Authorization请求报头域的请求，要求服务器对其进行验证。 Host（发送请求时，该报头域是必需的） Host请求报头域主要用于指定被请求资源的Internet主机和端口号，它通常从HTTP URL中提取出来的，eg：我们在浏览器中输入：http://www.1000phone.com 浏览器发送的请求消息中，就会包含Host请求报头域，如下： Host：www.1000phone.com 此处使用缺省端口号80，若指定了端口号，则变成：Host：www.1000phone.com:指定端口号 User-Agent 我们上网登陆论坛的时候，往往会看到一些欢迎信息，其中列出了你的操作系统的名称和版本，你所使用的浏览器的名称和版本，这往往让很多人感到很神奇，实际上，服务器应用程序就是从User-Agent这个请求报头域中获取到这些信息。User-Agent请求报头域允许客户端将它的操作系统、浏览器和其它属性告诉服务器。不过，这个报头域不是必需的，如果我们自己编写一个浏览器，不使用User-Agent请求报头域，那么服务器端就无法得知我们的信息了。 请求报头举例： GET /form.html HTTP/1.1 (CRLF) Accept:image/gif,image/x-xbitmap,image/jpeg,application/x-shockwave-flash,application/vnd.ms-excel,application/vnd.ms-powerpoint,application/msword,/ (CRLF) Accept-Language:zh-cn (CRLF) Accept-Encoding:gzip,deflate (CRLF) If-Modified-Since:Wed,05 Jan 2007 11:21:25 GMT (CRLF) If-None-Match:W/"80b1a4c018f3c41:8317" (CRLF) User-Agent:Mozilla/4.0(compatible;MSIE6.0;Windows NT 5.0) (CRLF) Connection:Keep-Alive (CRLF) (CRLF)

3、响应报头 响应报头允许服务器传递不能放在状态行中的附加响应信息，以及关于服务器的信息和对Request-URI所标识的资源进行下一步访问的信息。 常用的响应报头 Location Location响应报头域用于重定向接受者到一个新的位置。Location响应报头域常用在更换域名的时候。 Server Server响应报头域包含了服务器用来处理请求的软件信息。与User-Agent请求报头域是相对应的。下面是 Server响应报头域的一个例子： Server：Apache-Coyote/1.1 WWW-Authenticate WWW-Authenticate响应报头域必须被包含在401（未授权的）响应消息中，客户端收到401响应消息时候，并发送Authorization报头域请求服务器对其进行验证时，服务端响应报头就包含该报头域。 eg：WWW-Authenticate:Basic realm="Basic Auth Test!" //可以看出服务器对请求资源采用的是基本验证机制。

4、实体报头 请求和响应消息都可以传送一个实体。一个实体由实体报头域和实体正文组成，但并不是说实体报头域和实体正文要在一起发送，可以只发送实体报头域。实体报头定义了关于实体正文（eg：有无实体正文）和请求所标识的资源的元信息。 常用的实体报头 Content-Encoding Content-Encoding实体报头域被用作媒体类型的修饰符，它的值指示了已经被应用到实体正文的附加内容的编码，因而要获得Content-Type报头域中所引用的媒体类型，必须采用相应的解码机制。Content-Encoding这样用于记录文档的压缩方法，eg：Content-Encoding：gzip Content-Language Content-Language实体报头域描述了资源所用的自然语言。没有设置该域则认为实体内容将提供给所有的语言阅读 者。eg：Content-Language:da Content-Length Content-Length实体报头域用于指明实体正文的长度，以字节方式存储的十进制数字来表示。 Content-Type Content-Type实体报头域用语指明发送给接收者的实体正文的媒体类型。eg： Content-Type:text/html;charset=ISO-8859-1 Content-Type:text/html;charset=GB2312 Last-Modified Last-Modified实体报头域用于指示资源的最后修改日期和时间。Expires Expires实体报头域给出响应过期的日期和时间。为了让代理服务器或浏览器在一段时间以后更新缓存中(再次访问曾访问过的页面时，直接从缓存中加载，缩短响应时间和降低服务器负载)的页面，我们可以使用Expires实体报头域指定页面过期的时间。eg：Expires：Thu，15 Sep 2006 16:23:12 GMT HTTP1.1的客户端和缓存必须将其他非法的日期格式（包括0）看作已经过期。eg：为了让浏览器不要缓存页面，我们也可以利用Expires实体报头域，设置为0，jsp中程序如下：response.setDateHeader("Expires","0");

千锋JAVA教学部

第五章 Servlet概述

5.1 什么是servlet

Servlet（Server Applet）是Java Servlet的简称，称为小服务程序或服务连接器，用Java编写的服务器端程序，主要功能在于交互式地浏览和修改数据，生成动态Web内容

ervlet运行于支持Java的应用服务器中。从实现上讲，Servlet可以响应任何类型的请求，但绝大多数情况下Servlet只用来扩展基于HTTP协议的Web服务器。

Servlet编程需要使用到javax.servlet 和 javax.servlet.http两个包下面的类和接口，在所有的类和接口中，javax.servlet.servlet 接口最为重要。所有的servlet程序都必须实现该接口或者继承实现了该接口的类。
javax.servlet.ServletConfig;
javax.servlet.ServletException;
javax.servlet.http.HttpServlet;
javax.servlet.http.HttpServletRequest;
javax.servlet.http.HttpServletResponse;
javax.servlet.http.HttpSession;
javax.servlet.http.Cookie;

5.2 Servlet的基本使用

1. 在src创建package
2. 选中刚刚创建的包，右键-->New-->Servlet
3. 第一个Servlet代码如下：

```
/**
 *第一个Servlet程序
 */
@WebServlet("/HelloServlet")
public class HelloServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /**
     * @see 构造函数
     */
    public HelloServlet() {
        super();
        // TODO Auto-generated constructor stub
    }

    /**
     * @see 处理Get请求
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        // 响应内容到浏览器
        response.getWriter().print("Hello Word");;
    }
```

千锋JAVA教学部

```
    /**
     * @see 处理Post请求
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        // TODO Auto-generated method stub
        doGet(request, response);
    }

}
```

1. 发布右键-->Run As-->Run On Server
2. 浏览器输入网址访问：http://localhost:8080/项目名称/HelloServlet

5.3 常见错误解决方法

```
HTTP Status 404   找不到
解决方法：查看Tomcat的webapps目录下找到当前项目在WEB-INF下的classes内能否找到刚刚的class文件
如果有，重新启动Tomcat
如果没有，在Eclipse中选择Project-->clean让Eclipse清空缓存并重新构建项目，再次运行
```

5.4 HTTP协议

5.4.1 什么是HTTP协议

```
超文本传输协议（HTTP，HyperText Transfer Protocol)是互联网上应用最为广泛的一种网络协议,是一个基于请求与响应模式的、无状态的、应用层的协议，常基于TCP的连接方式
```

```
HTTP协议的主要特点如下：
1.支持客户/服务器模式。
2.简单快速：客户向服务器请求服务时，只需传送请求方法和路径。请求方法常用的有GET、POST。每种方法规定了客户与服务器联系的类型不同。由于HTTP协议简单，使得HTTP服务器的程序规模小，因而通信速度很快。
3.灵活：HTTP允许传输任意类型的数据对象。传输的类型由Content-Type加以标记。
4.无连接：无连接的含义是限制每次连接只处理一个请求。服务器处理完客户的请求，并收到客户的应答后，即断开连接。采用这种方式可以节省传输时间。
5.无状态：HTTP协议是无状态协议。无状态是指协议对于事务处理没有记忆能力。缺少状态意味着如果后续处理需要前面的信息，则它必须重传，这样可能导致每次连接传送的数据量增大。另一方面，在服务器不需要先前信息时它的应答就较快。
```

5.4.2 Http协议的通信

```
HTTP通信机制是在一次完整的HTTP通信过程中，Web浏览器与Web服务器之间将完成下列7个步骤：
1、 建立TCP连接
在HTTP工作开始之前，Web浏览器首先要通过网络与Web服务器建立连接，该连接是通过TCP来完成的，该协议与IP协议共同构建Internet，即著名的TCP/IP协议族，因此Internet又被称作是TCP/IP网络。HTTP是比TCP更高层次的应用层协议，根据规则，只有低层协议建立之后才能进行更低层协议的连接。因此，首先要建立TCP连接，一般TCP连接的端口号是80
2、 浏览器向Web服务器发送请求命令
一旦建立了TCP连接，Web浏览器就会向Web服务器发送请求命令
例如：GET/sample/hello.html HTTP/1.1
3、 浏览器发送请求头信息
```

千锋JAVA教学部

浏览器发送其请求命令之后，还要以头信息的形式向Web服务器发送一些别的信息，之后浏览器发送了一空白行来通知服务器，它已经结束了该头信息的发送。
4、 Web服务器应答
客户机向服务器发出请求后，服务器会客户机回送应答，
HTTP/1.1 200 OK
应答的第一部分是协议的版本号和应答状态码
5、 Web服务器发送应答头信息
正如客户端会随同请求发送关于自身的信息一样，服务器也会随同应答向用户发送关于它自己的数据及被请求的文档。
6、 Web服务器向浏览器发送数据
Web服务器向浏览器发送头信息后，它会发送一个空白行来表示头信息的发送到此为结束，接着，它就以Content-Type应答头信息所描述的格式发送用户所请求的实际数据
7、 Web服务器关闭TCP连接
一般情况下，一旦Web服务器向浏览器发送了请求数据，它就要关闭TCP连接，然后如果浏览器或者服务器在其头信息加入了这行代码
Connection:keep-alive
TCP连接在发送后将仍然保持打开状态，于是，浏览器可以继续通过相同的连接发送请求。保持连接节省了为每个请求建立新连接所需的时间，还节约了网络带宽

5.4.3 数据格式

HTTP请求格式

当浏览器向Web服务器发出请求时，它向服务器传递了一个数据块，也就是请求信息，HTTP请求信息由3部分组成：
l 请求方法URI协议/版本
l 请求头(Request Header)
l 请求正文
下面是一个HTTP请求的例子：

```
GET/hello HTTP/1.1
Accept:text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Accept-Language:zh-CN,zh;q=0.8,en-GB;q=0.6,en;q=0.4
Connection:Keep-Alive
Host:localhost
User-Agent:Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/59.0.3071.115 Safari/537.36
Accept-Encoding:gzip, deflate, br

name=zhangsan
```

1、请求方法URI协议/版本
请求的第一行是“方法URL议/版本”：GET/hello HTTP/1.1
以上代码中“GET”代表请求方法，“/hello”表示URI，“HTTP/1.1代表协议和协议的版本。
根据HTTP标准，HTTP请求可以使用多种请求方法。例如：HTTP1.1支持7种请求方法：GET、POST、HEAD、OPTIONS、PUT、DELETE和TARCE。在Internet应用中，最常用的方法是GET和POST。
URL完整地指定了要访问的网络资源，通常只要给出相对于服务器的根目录的相对目录即可，因此总是以“/”开头，最后，协议版本声明了通信过程中使用HTTP的版本。
2、请求头(Request Header)
请求头包含许多有关的客户端环境和请求正文的有用信息。例如，请求头可以声明浏览器所用的语言，请求正文的长度等。

```
Accept:text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Accept-Language:zh-CN,zh;q=0.8,en-GB;q=0.6,en;q=0.4
Connection:Keep-Alive

Host:localhost
```

千锋JAVA教学部

```
User-Agent:Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/59.0.3071.115 Safari/537.36
Accept-Encoding:gzip, deflate, br
3、请求正文
请求头和请求正文之间是一个空行，这个行非常重要，它表示请求头已经结束，接下来的是请求正文。请求正文中可以
包含客户提交的查询字符串信息：
name=zhangsan
```

第六章 Servlet使用

6.1 Servlet核心接口和类

```
Servlet接口
在ServletAPI中最重要的是Servlet接口，所有Servlet都会直接或间接的与该接口发生联系，或是直接实现该接口，
或间接继承自实现了该接口的类。
该接口包括以下五个方法：

init(ServletConfig config)
ServletConfig getServletConfig()
service(ServletRequest req,ServletResponse res)
String getServletInfo()
destroy( )
```

```
GenericServlet抽象类
它对Servlet接口中的部分方法（init和destroy）添加了实现，使得开发时只需要考虑针对service方法的业务实现即
可。
```

```
HttpServlet类
是继承GenericServlet的基础上进一步的扩展，
扩展方法：
public voidinit(ServletConfig config),
public void init()。他们有如下的关系： init（ServletConfig config）方法由tomcat自动调用，它读取web工
程下的web.xml，将读取的信息打包传给此参数，此方法的参数同时将接收的信息传递给GenericServlet类中的成员变
量config，同时调用init()。以后程序员想重写init方法可以选择init（ServletConfig config）或者init()，但
是选择init（ServletConfig config）势必会覆盖此方法已实现的内容，没有为config变量赋值，此后若是调用
getServletConfig（）方法返回config时会产生空指针异常的，所以想重写init（ServletConfig config）方法，
必须在方法体中第一句写上 super.init（config），为了防止程序员忘记重写super.init（config）方法sun公司自
动为用户生成一个public void init()的方法
```

6.2 Servlet的三种创建方式

Servlet的第一种创建方式：继承HttpServlet

```
/**
 * Servlet implementation class HelloServlet
 * 演示Servlet的第一种创建方式，继承HttpServlet.也是开发中推荐的
 *
 */
@WebServlet("/hs1")
public class HelloServlet extends HttpServlet {

    private static final long serialVersionUID = 1L;
```

千锋JAVA教学部

```
    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {

        response.getWriter().print("我是Servlet创建的第一种方式");
    }
    /**
     * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        // TODO Auto-generated method stub
        doGet(request, response);
    }
}
```

Servlet创建的第二种方式：实现接口Servlet

```
/**
 * Servlet创建的第二种方式：实现接口Servlet
 * */
@WebServlet("/hs2")
public class HelloServlet2 implements Servlet{

    @Override
    public void destroy() {
        // TODO Auto-generated method stub
    }

    @Override
    public ServletConfig getServletConfig() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public String getServletInfo() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public void init(ServletConfig arg0) throws ServletException {
        // TODO Auto-generated method stub
    }
    @Override
    public void service(ServletRequest request, ServletResponse response) throws
ServletException, IOException {
        // TODO Auto-generated method stub
```

千锋JAVA教学部

```
        System.out.println("OK");
        response.setContentType("text/html;charset=UTF-8");
        response.getWriter().println("我是第二种创建方式");
    }
}
```

Servlet的第三种创建方式：继承GenericServlet类

```
/**
 * Servlet的第三种创建方式：继承GenericServlet类*/
@WebServlet(value="/hs3")
public class HelloServlet3 extends GenericServlet{

    @Override
    public void service(ServletRequest request, ServletResponse response) throws
ServletException, IOException {
        // TODO Auto-generated method stub
        System.out.println("OK");
        response.setContentType("text/html;charset=UTF-8");
        response.getWriter().println("我是第三种创建方式");
    }
}
```

6.3 Servlet的2种配置方式

第一种注解式配置 Servlet3.0及以后：

```
/**
 * Servlet implementation class HelloServlet
 * 演示Servlet注解式配置
 */
@WebServlet(value="/hello",loadOnStartup=1)
public class HelloServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {

        response.getWriter().print("OK");
    }
    /**
     * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        // TODO Auto-generated method stub
        doGet(request, response);
    }
}
```

千锋JAVA教学部

第二种web.xml配置 Servlet所有版本都支持：

```
/**
 * Servlet implementation class HelloServlet
 * 演示Servlet的web.xml配置
 */
public class HelloServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {

        response.getWriter().print("OK");
    }
    /**
     * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        // TODO Auto-generated method stub
        doGet(request, response);
    }
}
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-
app_3_1.xsd" version="3.1">
  <display-name>Web_Day11</display-name>
  <!--Servlet的第二种配置  -->
  <!--Servlet配置  -->
  <servlet>
  <!--名称  -->
    <servlet-name>hello2</servlet-name>
    <!--Servlet的全称类名  -->
    <servlet-class>com.qf.web.servlet.HelloServlet</servlet-class>
    <!--启动的优先级，数字越小越先起作用  -->
    <load-on-startup>1</load-on-startup>
  </servlet>
  <!--映射配置  -->
  <servlet-mapping>
  <!--名称  -->
    <servlet-name>hello2</servlet-name>
    <!--资源的匹配规则：精确匹配  -->
    <url-pattern>/hello2</url-pattern>

  </servlet-mapping>
```

千锋JAVA教学部

```
  <welcome-file-list>
    <welcome-file>login.html</welcome-file>
  </welcome-file-list>
</web-app>
```

6.4 Servlet生命周期

6.4.1 什么是生命周期

生命周期也就是生命历程，就像人都是从怀胎-->出生-->婴儿-->儿童-->少年-->成年-->中年-->老年-->死亡

6.4.2 生命周期的四个阶段

阶段一、实例化
实例化阶段是Servlet生命周期中的第一步，由Servlet容器调用Servlet的构造器创建一个具体的Servlet对象的过程。而这个创建的时机可以是在容器收到针对这个组件的请求之后，即用了才创建；也可以在容器启动之后立刻创建实例，而不管此时Servlet是否使用的上。使用如下代码可以设置Servlet是否在服务器启动时就执行创建

阶段二、初始化
Servlet在被加载实例化之后，必须要初始化它。在初始化阶段，init（）方法会被调用。这个方法在javax.servlet.Servlet接口中定义。其中，方法以一个ServletConfig类型的对象作为参数。ServletConfig对象由Servlet引擎负责创建，从中可以读取到事先在web.xml文件中通过<init-param>节点配置的多个name-value名值对。ServletConfig对象还可以让Servlet接受一个ServletContext对象。
一般情况下，init方法不需要编写，因GenericServlet已经提供了init方法的实现，并且提供了getServletConfig方法来获得ServletConfig对象。
注：init方法只被执行一次

阶段三、就绪/服务
Servlet被初始化以后就处于能够响应请求的就绪状态。每个对Servlet的请求由一个ServletRequest对象代表，Servlet给客户端的响应由一个ServletResponse对象代表。当客户端有一个请求时，容器就会将请求与响应对象转给Servlet，以参数的形式传给service方法。service方法由javax.servlet.Servlet定义，由具体的Servlet实现HttpServlet将service方法拆分了。doGet和doPost

阶段四、销毁
Servlet容器在销毁Servlet对象时会调用destroy方法来释放资源。通常情况下Servlet容器停止或者重新启动都会引起销毁Servlet对象的动作，但除此之外，Servlet容器也有自身管理Servlet对象的准则，整个生命周期并不需要人为进行干预

Servlet代码如下：

```
/**
 * Servlet implementation class LifeServlet
 * 演示Servlet的生命周期：
 * 1、实例化
 * 2、init:初始化
 * 3、service：服务
 * 4、destory：销毁
 */
@WebServlet("/LifeServlet")

public class LifeServlet extends HttpServlet {
```

千锋JAVA教学部

```
    private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public LifeServlet() {
        super();
        System.out.println("1、完成了实例化");
        // TODO Auto-generated constructor stub
    }
    @Override
    public void init() throws ServletException {
        // TODO Auto-generated method stub
        super.init();
        System.out.println("2、完成了初始化");
    }

    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        // TODO Auto-generated method stub
        System.out.println("3、就绪中");
        response.getWriter().append("Served at: ").append(request.getContextPath());
    }

    /**
     * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        // TODO Auto-generated method stub
        doGet(request, response);
    }

    @Override
    public void destroy() {
        // TODO Auto-generated method stub
        super.destroy();
        System.out.println("4、销毁了");
    }
}
```

6.5 获取请求参数

html页面：

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">

<title>欢迎页面</title>
```

千锋JAVA教学部

```
</head>
<body>
<h1>欢迎你</h1>
<div>
<form action="HelloServlet">
<label>姓名：</label><input name="name"><br/>
<label>年龄：</label><input  name="age"><br/>
<input type="submit" value="提交">
</form>
</div>
</body>
</html>
```

Servlet代码：

```
/**
 * Servlet implementation class HelloServlet
 * 演示Servlet的获取请求参数
 *
 */
@WebServlet("/HelloServlet")
public class HelloServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        //获取表单提交的姓名
        String name=request.getParameter("name");
        //获取年龄
        String age=request.getParameter("age");
        //服务端输出打印
        System.out.println(request.getRemoteAddr()+"发来信息：姓名："+name+"---->年龄："+age);
    }

    /**
     * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        // TODO Auto-generated method stub
        doGet(request, response);
    }
}
```

6.6 请求方式

默认的请求方式是GET请求

6.6.1 GET请求

千锋JAVA教学部

GET提交的数据会放在URL之后，以?分割URL和传输数据，参数之间以&相连
GET提交的数据大小有限制（因为浏览器对URL的长度有限制）
GET方式提交数据，会带来安全问题
效率高

对应的Servlet的方法是doGet

6.6.2 POST请求

POST方法是把提交的数据放在HTTP包的Body中
POST方法提交的数据没有限制
POST提交的数据相对安全
效率相对没有GET高

对应的Servlet的方法是doPost

6.7 如何处理中文参数

6.7.1 为什么表单中会产生中文乱码

产生乱码，就是因为服务器和客户端沟通的编码不一致造成的，因此解决的办法是：在客户端和服务器之间设置一个统一的编码，之后就按照此编码进行数据的传输和接收

6.7.2 GET中文乱码

在Tomcat7及以下
客户端以UTF-8的编码传输数据到服务器端，而服务器端的request对象使用的是ISO8859-1这个字符编码来接收数据，服务器和客户端沟通的编码不一致因此才会产生中文乱码的。解决办法：在接收到数据后，先获取request对象以ISO8859-1字符编码接收到的原始数据的字节数组，然后通过字节数组以指定的编码构建字符串，解决乱码问题。

Tomcat8的版本中GET基本就不会乱码了，因为服务器对url的编码格式可以进行自动转换

解决get中文乱码的代码：

```
/**
 * Servlet implementation class HelloServlet
 * 演示Servlet的GET请求,中文乱码的问题
 *
 */
@WebServlet("/GETServlet")
public class GetServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        //获取表单提交的姓名
        String name=request.getParameter("name");

        name=new String(name.getBytes("ISO8859-1"),"UTF-8");
```

千锋JAVA教学部

```
        //获取年龄
        String age=request.getParameter("age");
        //服务端输出打印
        System.out.println(request.getRemoteAddr()+"发来信息：姓名："+name+"---->年龄："+age);
    }
    /**
     * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        // TODO Auto-generated method stub
        doGet(request, response);
    }
}
```

6.7.3 POST乱码

由于客户端是以UTF-8字符编码将表单数据传输到服务器端的，因此服务器也需要设置以UTF-8字符编码进行接收，要想完成此操作，服务器可以直接使用从ServletRequest接口继承而来的"setCharacterEncoding(charset)"方法进行统一的编码设置。

解决POST中文乱码代码如下：

```
/**
 * Servlet implementation class HelloServlet
 * 演示Servlet的GET请求,中文乱码的问题
 *
 */
@WebServlet("/GETServlet")
public class GetServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        }

    /**
     * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        //设置请求参数的编码格式--对GET无效
        request.setCharacterEncoding("UTF-8");
        //获取表单提交的信息
        String name=request.getParameter("msg");
        //服务端输出打印
        System.out.println(request.getRemoteAddr()+"发来信息："+msg);
    }

}
```

千锋JAVA教学部

6.8 Servlet输出中文

6.8.1 页面返回乱码原因

浏览器识别不到返回的中文是什么编码格式，就会默认使用GB2312,如果返回的是UTF-8格式的那么在浏览器上就会显示乱码的问题

6.8.2 如何解决内容中的乱码

```
response.setContentType("text/html;charset=UTF-8");
```

第七章 Servlet路径处理和特性

7.1 重定向

重定向就是通过各种方法将各种网络请求重新定个方向转到其它位

7.1.1 重定向原理和特点

客户浏览器发送http请求----》web服务器接受后发送302状态码响应及对应新的location给客户浏览器--》客户浏览器发现是302响应，则自动再发送一个新的http请求，请求url是新的location地址----》服务器根据此请求寻找资源并发送给客户。在这里location可以重定向到任意URL，既然是浏览器重新发出了请求，则就没有什么request传递的概念了。在客户浏览器路径栏显示的是其重定向的路径，客户可以观察到地址的变化的。

特点：
1，重定向是客户端行为。
2，重定向是浏览器做了至少两次的访问请求的。
3，重定向浏览器地址改变。
4，重定向2次跳转之间传输的信息会丢失（request范围）。
5，重定向可以指向任何的资源，包括当前应用程序中的其他资源，同一个站点上的其他应用程序中的资源，其他站点的资源。注意：传递给HttpServletResponse.sendRedirect 方法的相对URL以“/”开头，它是相对于整个WEB站点的根目录

7.1.2 实现重定向

html页面：

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>登录页面</title>
</head>
<body>
<div>
<form action="login" method="post">
<label>账号：</label><input name="un"/><br/>
<label>密码：</label><input type="password" name="pw"/><br/>
<input type="submit" value="登录"/>
</form>

</div>
```

千锋JAVA教学部

```
<h1><a href="register.html">还没账号请注册</a></h1>
</body>
</html>
```

Servlet代码：

```
/**
 * Servlet implementation class LoginServlet
 * 实现登录信息操作
 */
@WebServlet(value="/LoginServlet",initParams={@WebInitParam(name="version",value="v10")})
public class LoginServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    /**
     * @see HttpServlet#HttpServlet()
     */
    public LoginServlet() {
        super();
        // TODO Auto-generated constructor stub
    }

    @Override
    public void init(ServletConfig config) throws ServletException {
        // TODO Auto-generated method stub
        super.init(config);
    }
    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        // TODO Auto-generated method stub
        //response.getWriter().append("Served at: ").append(request.getContextPath());
    }

    /**
     * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        // TODO Auto-generated method stub
        //doGet(request, response);
        //获取表单提交的用户名
        String username=request.getParameter("un");
        //获取密码
        String password=request.getParameter("pw");
        //服务端输出打印
        System.out.println("POST:用户名："+username+"---->密码："+password);
        //重定向.可以跳转到指定的页面
        response.sendRedirect("register.html");
    }
}
```

千锋JAVA教学部

7.2 请求转发

Servlet除了支持重定向之外还支持请求转发

7.2.1 转发的原理和特点

原理：
客户浏览器发送http请求----》web服务器接受此请求--》调用内部的一个方法在容器内部完成请求处理和转发动作----》将目标资源发送给客户。在这里，转发的路径必须是同一个web容器下的url，其不能转向到其他的web路径上去，中间传递的是自己的容器内的request。在客户浏览器路径栏显示的仍然是其第一次访问的路径，也就是说客户是感觉不到服务器做了转发的

特点：
1，转发是服务器行为
2，转发是浏览器只做了一次访问请求
3，转发浏览器地址不变
4，转发2次跳转之间传输的信息不会丢失，所以可以通过request进行数据的传递
5，转发只能将请求转发给同一个WEB应用中的组件
注意：如果创建RequestDispatcher 对象时指定的相对URL以“/”开头，它是相对于当前WEB应用程序的根目录。

7.2.2 实现请求转发

```
package com.qf.web.servlet;

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

/**
 * Servlet implementation class RegisterServlet
 请求转发
 */
@WebServlet("/RegisterServlet")
public class RegisterServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public RegisterServlet() {
        super();
        // TODO Auto-generated constructor stub
    }

    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse
     *      response)
     */

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
```

千锋JAVA教学部

```
            throws ServletException, IOException {
        // TODO Auto-generated method stub
        response.getWriter().append("Served at: ").append(request.getContextPath());
    }

    /**
     * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse
     *      response)
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        // 获取表单提交的用户名
        String username = request.getParameter("un");
        // 获取密码
        String password = request.getParameter("pw");
        // 服务端输出打印
        System.out.println("POST:注册：用户名：" + username + "---->密码：" + password);
        //请求转发
        request.getRequestDispatcher("LoginServlet").forward(request, response);
    }
}
```

7.3 请求资源路径

7.3.1 什么是请求资源路径

```
在浏览器输入：http://127.0.0.1:8080/Web_Day12/HelloServlet
Web_Day12/HelloServlet就是请求资源路径
```

7.3.2 容器对请求的处理过程

```
在浏览器输入：http://127.0.0.1:8080/Web_Day12/HelloServlet
浏览器依据ip，port建立与Servlet容器之间的连接，然后将请求资源路径Web_Day12/HelloServlet 发送过去给容器
容器依据应用名“/Web_Day12”找到应用所在的文件夹，容器会默认请求的是一个Servlet，
如果使用的是XML配置，那么查找web.xml文件中所有的Servlet配置url-pattern,看是否有匹配的Servlet
如果使用的是注解式配置，那么就直接检索注解@WebServlet的值是否匹配
```

7.3.3 匹配Servlet的规则

```
容器在进行url-pattern比对的时候是遵循一定的匹配原则的
```

匹配规则：

```
url-pattern定义匹配规则，取值说明：
精确匹配
/具体的名称                    只有url路径是具体的名称的时候才会触发Servlet
后缀匹配
*.xxx                        只要是以xxx结尾的就匹配触发Servlet
通配符匹配
/*                                  匹配所有请求，包含服务器的所有资源
```

千锋JAVA教学部

7.4 单Servlet实现多请求

7.4.1 为什么要将Servlet合并

Servlet作为Web应用中最核心的环节是因为这个组件不仅能接受请求，还能够为该请求提供响应，所以Servlet一般都会充当整个应用的控制器来进行请求的分发，为不同的请求找到对应的资源。于是程序中大多只需要一个Servlet完成这个分发工作即可，合并多个Servlet为一个Servlet会让程序的处理逻辑更加明确

实现Servlet的合并:
使用后缀模式完成请求资源路径的匹配
分析请求资源路径中的请求目标，完成分发的动作

7.4.2 使用后缀模式完成路径匹配

2种方式:
1、在web.xml中
可以设置url-pattern的值为：*.do，这样只要是以do结尾的请求都可以进行匹配
2、在定义的Servlet类上
在注解@WebServlet("*.do")

7.4.3 分析请求资源后分发

配置完web.xml文件后，不同请求都会发送到Web.SomeServlet来处理，要想起到分发的作用，则需要分析调过来的请求中具体的请求目标是什么。

7.8 ServletContext

ServletContext:Servlet上下文

7.8.1 什么是Servlet上下文

Servlet上下文又叫做：ServletContext。
当WEB服务器启动时，会为每一个WEB应用程序(webapps下的每个目录就是一个应用程序)创建一块共享的存储区域
ServletContext也叫做“公共区域”，也就是同一个WEB应用程序中，所有的Servlet和JSP都可以共享同一个区域。
ServletContext在WEB服务器启动时创建，服务器关闭时销毁
容器在启动的时候，会为每一个web应用创建唯一的一个符合ServletContext接口的对象，该对象一般称之为"servlet上下文"。

7.8.2 如何获得Servlet上下文

方式一:GenericServlet提供了getServletContext()方法。
方式二:ServletConfig提供了getServletContext()方法。
方式三:HttpSession提供了getServletContext()方法。
方式四:HttpServletRequest提供了getServletContext()方法。

7.8.3 Servlet上下文的作用及特点

作用:
1、获取真实路径

获取当前项目的发布路径

千锋JAVA教学部

```
request.getSelectContext().getRealPath("/");
2、获取容器的附加信息
System.out.println(request.getServletContext().getServerInfo());
System.out.println(request.getServletContext().getContextPath());
3、全局容器
//设置信息到全局容器中
request.getServletContext().setAttribute("msg", "共享信息");
//获取数据
System.out.println(request.getServletContext().getAttribute("msg"));
//移除数据
request.getServletContext().removeAttribute("msg");
```

特点:
唯一性：一个应用对应一个servlet上下文。
一直存在：只要容器不关闭或者应用不卸载，servlet上下文就一直存在。

7.9 Servlet的路径

绝对路径：绝对路径就是你的主页上的文件或目录在硬盘上真正的路径，(URL和物理路径)例如：
C:/xyz/test.txt 代表了test.txt文件的绝对路径。http://www.sun.com/index.htm也代表了一个URL绝对路径

相对路径：相对与某个基准目录的路径。包含Web的相对路径（HTML中的相对目录），例如：在Servlet中，"/"代表Web应用的跟目录。和物理路径的相对表示。例如："./" 代表当前目录，"../"代表上级目录。这种类似的表示，也是属于相对路径

服务器端的相对地址指的是相对于你的web应用的地址，这个地址是在服务器端解析的（不同于html和javascript中的相对地址，他们是由客户端浏览器解析的）也就是说这时候在jsp和servlet中的相对地址应该是相对于你的web应用，即相对于http://127.0.0.1:8080/test/的

7.10 Servlet线程安全问题

7.10.1 线程安全问题

因为每次请求都会创建一个线程，如果多人同时请求，那么就会存在多个线程操作同一个Servlet对象，那么如果在对应的方法中操作了成员变量，就有可能产生线程安全的问题。

7.10.2 如何保证线程安全

1、synchronized
将存在线程安全问题的代码放到同步代码块中
2、实现singleThreadModel接口
servlet实现singleThreadModel接口后，每个线程都会创建servlet实例，这样每个客户端请求就不存在共享资源的问题，但是servlet响应客户端请求的效率太低，所以已经淘汰。
3、尽可能只使用局部变量

7.11 综合案例

实现个人信息详细信息的提交

千锋JAVA教学部

页面：

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>个人信息的收集</title>
</head>
<body>
<h3>个人信息</h3>
<form action="quest" method="post">
<label>姓名：</label><input type="text" name="name"><br/>
<label>年龄：</label><input type="text" name="age"><br/>
<label>性别：</label><input type="radio" name="sex" id="female" checked/><label for="female">女
</label>
<input type="radio" name="sex" id="male"/><label for="male">男</label><br/>
<label>联系方式：</label><input type="text" name="phone"><br/>
<label>电子邮件:</label><input type="email" name="email"><br/>
<input type="submit">
</form>
</body>
</html>
```

Servlet:

```
/**
 * Servlet implementation class Servlet1
 */
@WebServlet("/quest")
public class DBWriteServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public DBWriteServlet() {
        super();
    }

    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    }

    /**
     * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
     */


    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
```

千锋JAVA教学部

```
ServletException, IOException {
        request.setCharacterEncoding("utf-8");
        String name = request.getParameter("name");
        int age =Integer.parseInt(request.getParameter("age")) ;
        String sex = request.getParameter("sex");
        String phone = request.getParameter("phone");
        String email = request.getParameter("email");
        System.out.println(name);
        System.out.println(age);
        System.out.println(sex);
        System.out.println(phone);
        System.out.println(email);
    }
}
```

作业题

1、请结合之前学习的前端技术，实现一个页面输入内容存储到数据库的功能
要求页面尽可能的包含自己会的标签，使用Servlet接收页面数据，通过JDBC存储到数据库

面试题

1、说一说Servlet生命周期
2、forward和redirect的区别

千锋JAVA教学部