* Servlet基础

1.1. Web应用的演变

1.1.1. 单机程序

软件简单分为单机程序和网络程序。发展到今时今日仍有大量的不依赖网络的单机程序被我们使用，如记事本、Excel、PPT、ZIP压缩等软件都是大家熟知的装机必备软件。

1.1.2. 网络程序

最早的网络程序是基于主机+终端模式的，也就是整个应用中只有一台大型主机，各个操作地点都是使用一条专线与主机相连，终端不提供任何运算和界面，类似于Unix形式，所有的运算和处理都由主机来完成。主机一般处理能力非常强大，并且稳定，主要机型都是由IBM这样的大公司提供。但主机的高昂的价格以及扩展难、维护费用高等弊端并不是一般企业所能承受，所以除银行、航空订票、证券等大企业在使用以外，大多数企业开始转投CS架构的程序，即客户端服务器架构。

1.1.3. 网络程序-CS架构

CS架构的发展过程经历了两层CS架构，三层CS架构以及多层CS架构的演变。

两层的CS架构是由客户端和后面的数据库组成的。数据库用于存放数据，并且使用数据库编程语言编写业务逻辑，客户端则使用VB、VC、Delphi这样的可视化编程方便的语言来开发客户端的输入输出界面。

三层的CS架构指的是客户端+应用服务器+数据库，即将混合在数据库端的业务逻辑从中分离出来，放入到应用服务器中，数据库只负责数据的管理、存储及检索。客户端负责界面。

这样的程序结构虽然在一定程度上降低了对数据库编程的依赖，并且能够适应大型的应用程序，但数据通信模块的增加却提升了开发的难度以及整体架构的复杂度。

1.1.4. 网络程序-BS架构

为了降低三层CS架构中与通信有关的复杂度，BS架构开始成为了网络程序中一大重要的架构类型。

BS架构即Browser + Web Server + DB。

由于三层CS架构中，自定义协议提升了整体的复杂度，那么就将自定义协议变成标准的HTTP协议。于是客户端使用HTTP协议进行数据打包拆包的程序即各厂商依据标准开发的浏览器，Web服务器也是基于HTTP协议由一些厂商提供，如IIS，Apache等。这样基于浏览器和服务器的架构中，由于协议已被限定，所以与通信有关的数据打包拆包的过程都不用我们开发人员来编写程序，只需要考虑将HTTP协议解析出来的数据进行业务处理，以及将什么样的结果提供给响应即可。也就是开发过程中只需要考虑7,8,9这三个步骤即可。于是大大降低了网络程序的开发难度，所以这种架构得到了大量的应用。

* CS和BS的区别

CS

•Client Server

•客户端需要单独开发单独安装

BS

•Browser Server

•由浏览器充当客户端,不需要单独开发及安装

1.2. 什么是Servlet

1.2.1. 扩展Web服务器功能

•Servlet是sun推出的用来在服务器端处理HTTP协议的组件.Servlet技术是使用Java语言开发的一套组件规范，不再像CGI技术那样需要分配单独的进程来处理请求，而是单独分配一个线程来处理，于是大大提升了处理效率。并且Java语言是跨平台的语言，也提升了Web服务器扩展程序的可移植性，已经取代了CGI技术，成为BS架构中的主流技术。所有后续的BS架构中的主流框架本质上都是基于Servlet来实现的。

1.2.2. 组件规范

组件规范是依靠一套API来实现的，也就是说开发中只要基于Sun公司提供的这套API，按照一定的规则来编写程序，那么就可以实现针对Web服务器的功能扩展。

但是组件只是对部分功能的一个实现，不能单独运行，必须放在一定的环境中才能运行。而这个针对各个组件进行管理、创建、销毁的运行环境即容器。

1.2.3. Servlet组件运行原理

Servlet作为补充Web服务器功能的组件，需要依赖于Servlet容器才能运行

在浏览器中输入请求地址后，浏览器会依据IP地址及端口号找到对应的Web服务器，如果请求的是静态资源，Web服务器直接提供响应；如果请求的是动态资源，Web服务器的通信模块会将该请求传递给Servlet容器的通信模块，Servlet容器负责创建Servlet实例，并将请求中的数据解析出来传递给Servlet。在Servlet处理完数据之后，响应结果也是由容器的通信模块负责返回给Web服务器。后续的Servlet的销毁及管理都由容器来负责。

能够充当Servlet容器这个角色的有很多软件，如Tomcat、Weblogic、JBoss等。而这些Servlet容器不仅仅具备了管理Servlet组件的功能，也具备了Web服务器的一些功能，所以很多时候只要安装一个Tomcat软件就同时具备了Web服务器及Servlet容器的双重功能。

1.3. 如何开发一个Servlet

1.3.1. 安装Tomcat

Servlet是Web服务器功能的补充，要想能够运行必须依赖Servlet容器的管理才可以，为了进行下一步的Servlet的具体开发，则需要先准备用于运行Servlet的环境，在学习过程中我们选择的是主流的Tomcat作为Web服务器及Servlet容器。

下面，简单介绍下Tomcat的安装步骤：

步骤一：下载并解压安装文件

在浏览器中输入 tomcat.apache.org，在Download菜单中选择Tomcat7.0后会得到如图- 4 所示的下载链接。

选择Core下面符合当前操作系统的版本后下载，下载后的文件为压缩文件，解压到硬盘即可。假定Tomcat解压后的路径为 c:\java\tomcat7.0

步骤二：配置环境变量

选择“我的电脑”右键 (“属性”( “高级系统设置” ( “环境变量” 用于新建及修改一些环境变量，以保证系统记录解压的Tomcat软件的位置。

点击系统变量下的“新建”按钮，创建“变量名”为 CATALINA\_HOME ， “变量值“ 为 c:\java\tomcat7.0的环境变量后点击确定按钮，

选中系统变量中的“Path“后点击编辑，在原有内容的最前面添加 “ %CATALINA\_HOME%\bin; ” 后点击确定按钮。注意：修改值的时候光标放在最前面，并且要用分号将路径与原有Path的值分开。

若安装jdk时已配置了CLASS\_PATH变量，则选中系统变量中的“CLASS\_PATH“后点击编辑，在原有内容的最前面添加 “ %CATALINA\_HOME%\lib\servlet-api.jar; ” 后点击确定按钮。注意：修改值的时候光标放在最前面，并且要用分号将路径与原有CLASS\_PATH的值分开。

至此，环境变量配置完毕。

步骤三：启动Tomcat

为了检测环境变量是否配置成功，以及Tomcat是否能够成功启动，首先进入命令行，windows系统下可以使用 “cmd“命令进入。在命令行窗口下输入“startup”命令，

打开浏览器，输入 “ http://localhost:8080 ”地址

至此，看到该页面即代表Tomcat启动成功。

步骤四：关闭Tomcat

在命令提示符下输入“shutdown”命令后，Tomcat的命令提示符窗口会关闭，在浏览器中再次输入http://localhost:8080，代表关闭Tomcat

1.3.2. 开发Servlet的步骤

步骤一：编写一个实现Servlet接口或继承HttpServlet的Java类

在硬盘上新建一个文件名为“HelloServlet.java”的源文件，并编写代码如下：

01.package web;

12.public class HelloServlet extends HttpServlet {

13. protected void service(HttpServletRequest req, HttpServletResponse res)throws ServletException, IOException{

14. resp.setContentType("text/html");

15. PrintWriter out = resp.getWriter();

16. out.write("<h1>Hello Servlet");

17. out.close();

18. }

19.}

步骤二：使用javac命令编译源文件为字节码文件

在命令行提示符下，进入到HelloServlet.java文件所在的位置，输入如下命令：

01.javac HelloServlet.java –cp c:\java\tomcat7.0\lib\servlet-api.jar

在没有报错的情况下会在java源文件的同级目录内出现HelloServlet.class的文件，代表编译成功。

步骤三：将编译完的组件打包

将组建打包，并新建web.xml文件，内容如下：

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

02.<web-app version="2.5"

03. xmlns="http://java.sun.com/xml/ns/javaee"

04. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

05. xsi:schemaLocation="http://java.sun.com/xml/ns/javaee

06. http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd">

07. <servlet>

08. <servlet-name>helloServlet</servlet-name>

09. <servlet-class>web.HelloServlet</servlet-class>

10. </servlet>

11. <servlet-mapping>

12. <servlet-name>helloServlet</servlet-name>

13. <url-pattern>/sayHi</url-pattern>

14. </servlet-mapping>

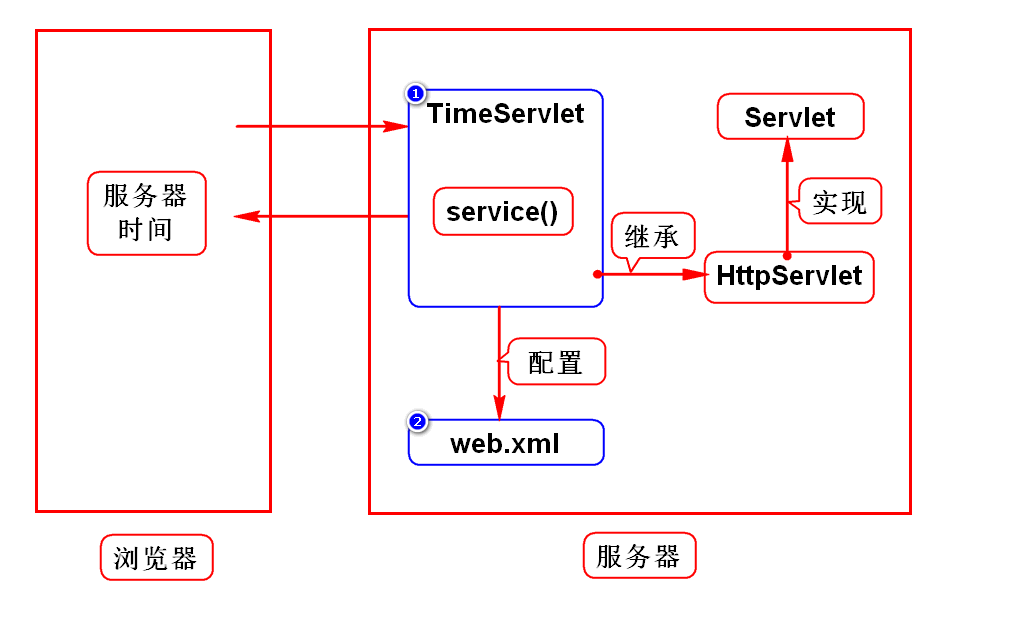
15.</web-app>

步骤四：部署

将整个firstweb文件夹拷贝到c:\java\tomcat7.0\webapps

步骤五：启动Tomcat，并访问Servlet

在命令提示符下输入“startup”命令启动Tomcat，启动成功后，打开浏览器，在地址栏中输入 “ http://localhost:8080/firstweb/sayHi “回车，得到第一个Web应用程序的运行结果。



1.4. 常见错误及解决方法

1.4.1. 404错误产生原因及解决方法

404产生的原因为Web服务器（容器）根据请求地址找不到对应资源，以下情况都会出现404的错误提示：

1.输入的地址有误（应用名大小写不正确，名称拼写不正确）

2.在web.xml文件中<servlet>和<servlet-mapping>中的两个<servlet-name>不一致

3.没有将工程部署到Tomcat的webapps下面

4.Web工程部署时的结构没有遵循Servlet的规范

具体的解决办法就是根据上面提到的4种情况，逐条进行检查。

1.4.2. 容器如何找到service（）方法

当在浏览器中输入 http://localhost:8080/firstweb/sayHi 这个地址后，容器是如何找到 HelloServlet.class这个文件并执行的呢？

首先容器会根据firstweb这个应用名找到位于webapps下面对应的文件夹，然后根据地址中的“/sayHi”到web.xml文件中寻找与之匹配的<url-pattern>节点，找到匹配的节点后会找到与该节点紧邻的<servlet-name>节点，获取名称并在此寻找与该名称相等的<servlet-name>节点，找到相等的节点后，搜寻该节点下面紧邻的<servlet-class>节点，于是获取到了与该地址相对应的类名，借助于ClassLoader加载该类文件，创建对象并调用service（）方法，客户端即看到了与该地址匹配的运行结果。

1.4.3. 405错误产生原因及解决方法

405这个错误的产生原因是容器找不到service方法来处理请求。以下情况容器都将视为没有找到service（）方法

1.service方法名写错，字母顺序不对或大小写错误

2.service方法参数类型与标准不一致，或参数顺序与标准不一致。一定是先HttpServletRequest，然后是HttpServletResponse类型

3.service方法异常、返回值类型与标准不一致，一定是抛出ServletException、IOException。返回值类型是void。

解决405错误的方法即检查service方法是否存在，签名（方法名、参数、返回值、异常类型）是否与覆盖的父类中的方法一致。

1.4.4. 500错误产生原因及解决方法

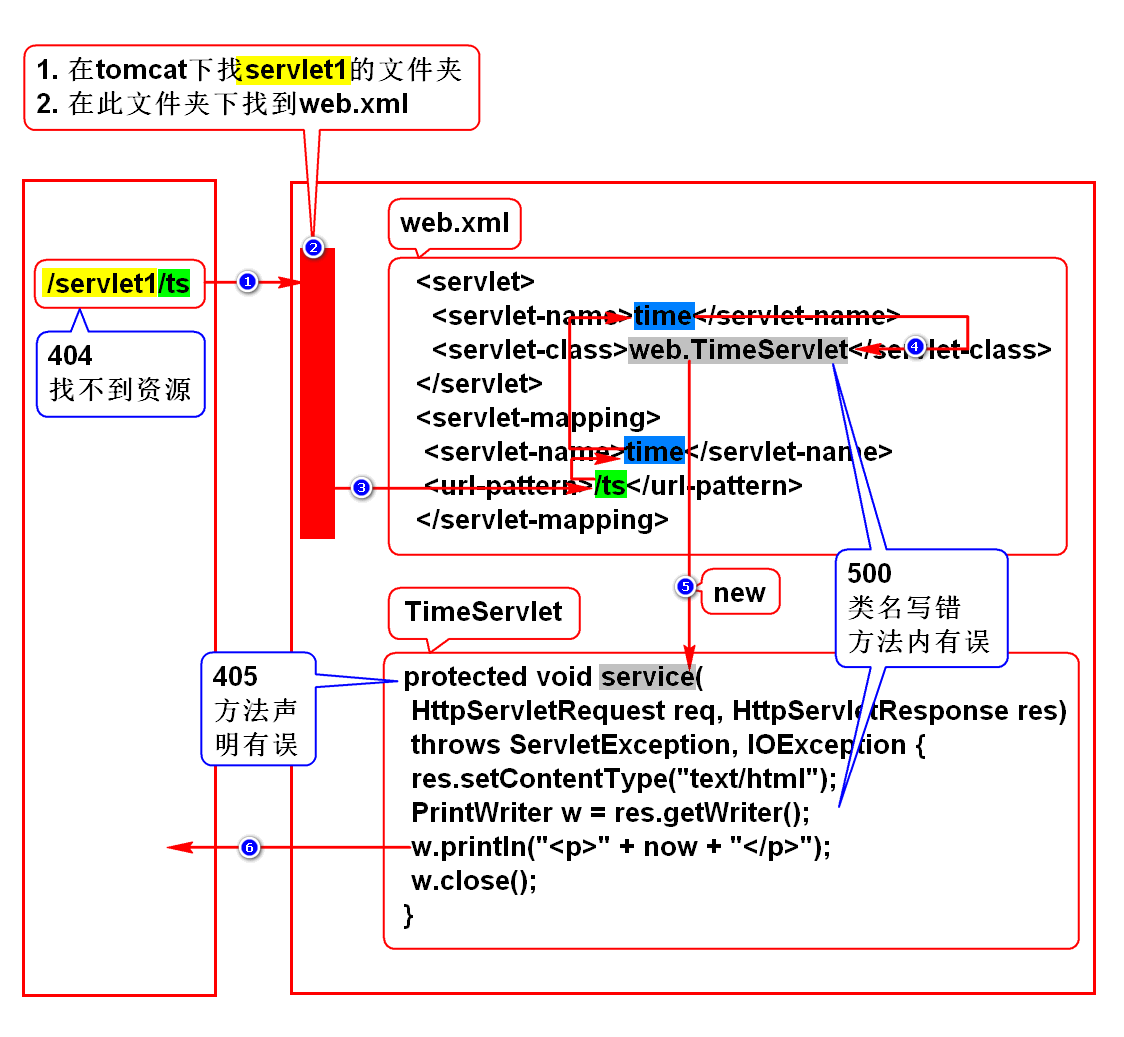
500的错误原因是请求的Servlet在执行service方法的过程中出现了错误，程序不能继续运行了。以下情况容器都将视为500的一种情况，而返回给客户端这个错误说明。

1.Servlet没有继承HttpServlet或实现Servlet接口

2.web.xml文件中的<servlet-class>写错，与真正的包名类名不符

3.service方法中的代码运行时抛出异常

解决500的方法为依据上面三种情况依次进行检测，逐条排除。



1.5. HTTP协议

1.5.1. 什么是HTTP协议

HTTP协议是HyperText Transfer Protocol的缩写，即超文本传输协议。是由w3c（万维网联盟）制定的一种应用层协议，用来定义浏览器与web服务器之间如何通信以及通信的数据格式。

因为BS架构中的通信模块就是以HTTP这个协议作为标准协议的，所以对该协议有所了解可以更好的编写程序。

1.5.2. HTTP协议是如何通信的

HTTP协议的通信过程可以分为以下四个步骤：

1.浏览器根据IP地址和端口号与服务器建立连接

2.向Web服务器发送请求数据包

3.Web服务器接收请求数据包后，发送相应的响应数据包

4.浏览器接收响应数据后关闭连接

在这个过程中，一次请求对应一次连接，当浏览器再次发请求给服务器时，Web服务器并不知道这就是上次发请求的客户端，这也是HTTP协议的一个特点-无状态协议。这种需要时建立连接，使用结束后立即断开连接的方式使得Web服务器可以利用有限的连接为尽可能多的客户提供服务。也正是具备了这样的特点，才使得BS结构能够承载企业级应用的大量访问。

1.5.3. HTTP协议中的数据格式

在HTTP协议控制的数据走向中，既包括客户端发送给服务器端的请求数据也包括服务器端返回给客户端的响应信息，而具有一定规范的数据格式是保证通信标准的第一要素。借助于一些浏览器的插件或者浏览器本身内嵌的功能模块，可以实现对请求数据、响应数据的抓取，熟练掌握这些数据的主要部分，可以有利于理解BS模型中的一些处理问题的方式。通常情况下的请求数据包及响应数据包结

1.5.4. 请求数据包组成

请求数据包包含三个部分：

1.请求行，包括请求方式，请求资源路径，以及协议版本

2.消息头是由若干的键值对组成，主要用于传递给Web服务器一些信息。如，浏览器版本、接受资源的类型等

3.在消息头的下面会跟着一个空白行，这样就可以区分出消息头与实体内容这两个部分。而在请求数据包中，大部分情况实体内容为空，只有当请求方式为POST时，实体内容才有要提交的表单数据

1.5.5. 响应数据包组成

响应数据包也包含三个部分：

1.状态行，包括协议类型、版本、状态码、状态描述

2.消息头（若干），也是多组键值对组成的信息说明。如返回的Content-Type消息头可以告诉浏览器返回的结果数据类型和字符集

3.实体内容，位于消息头下面的一个空白行之后。空白行意在通知浏览器消息头的结束以及结果内容的开始，浏览器就从这里开始数据的逐行解析

1.6. Servlet如何处理HTTP协议

1.6.1. 如何控制通信数据

HTTP协议在Web容器这端主要表现为通信数据的到达以及响应数据的返回。于是Web容器将这两部分数据解释为两个对象，一个是与请求数据对应的HttpServletRequest对象，一个是与响应数据对应的HttpServletResponse对象。对于Servlet来讲，主要的业务逻辑过程就是从请求对象中获取数据，经过加工后将结果附着在响应对象中发送回客户端。

1.6.2. HttpServletRequest对象

借助于HttpServletRequest对象可以实现很多操作，如读取请求行、消息头信息，取得路径信息等。使用如下代码可以实现相关功能。

01.package web;

14.public class RequestServlet extends HttpServlet {

16. protected void service(HttpServletRequest req, HttpServletResponse resp)

17. throws ServletException, IOException {

18. Enumeration e = req.getHeaderNames ();

19. while (e.hasMoreElements ()){

20. String headerName = e.nextElement ().toString ();

21. System.out.println (headerName+":"+req.getHeader (headerName));

22. }

23. System.out.println (req.getMethod ());

24. System.out.println (req.getProtocol ());

25. System.out.println (req.getRequestURI ());

26. System.out.println (req.getRequestURL ());

27. System.out.println (req.getServletPath ());

28. }

29.}

1.6.3. HttpServletResponse对象

HttpServletResponse对象最主要的作用即设置给浏览器的响应内容及浏览器的解码方式，此外设置cookie及重定向也都是响应对象的功能。这些功能对应的代码实现如下所示。

01.package web;

12.public class ResponseServlet extends HttpServlet {

14. protected void service(HttpServletRequest req, HttpServletResponse resp)

15. throws ServletException, IOException {

16. resp.setContentType("text/html;charset=gbk");

17. resp.setStatus(404);

18. }

19.

20.}

* 注意:

1)很多事情不需要我们处理

•通信的步骤由浏览器和服务器自动实现

•请求数据中的请求行,消息头由浏览器自动填写

•响应数据中的状态行,消息头通常由服务器自动填写

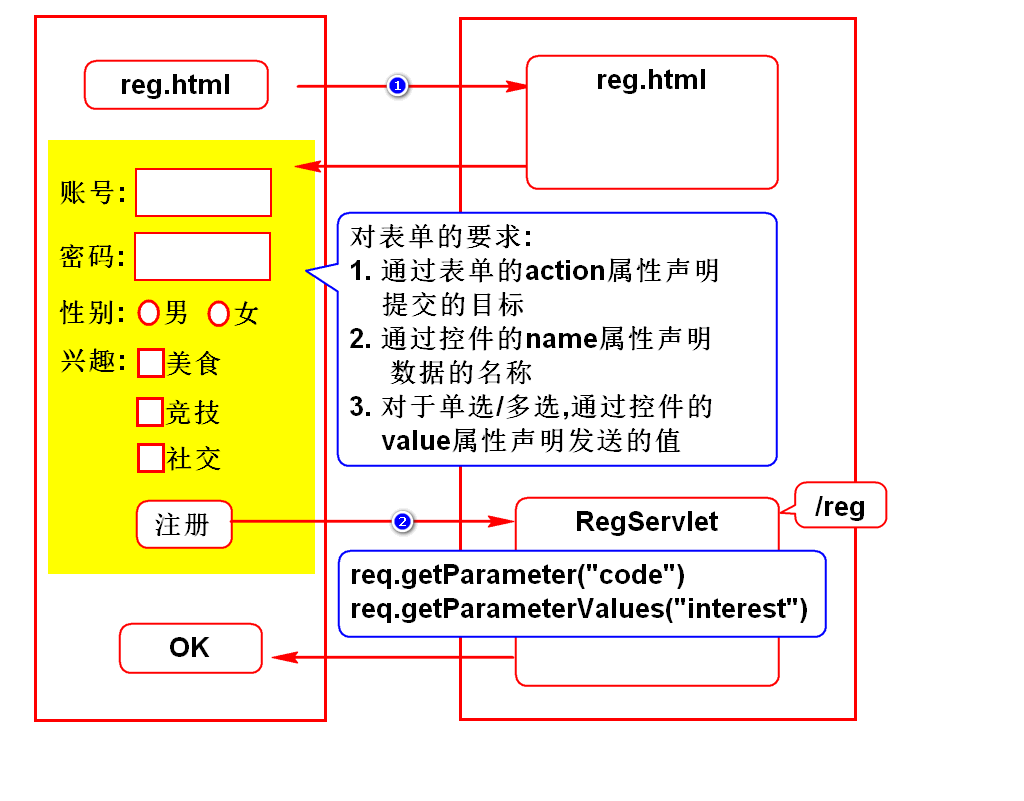
2.少量事情需要我们处理

•请求数据中的实体内容由我们提供

•响应数据中的实体内容由我们提供

通过request处理请求数据,通过response处理响应数据

* 注册案例:



* Servlet核心（I）

1.1. 请求方式

1.1.1. 请求方式的种类

请求数据包中的请求行的第一个参数就是请求方式，是客户端向Web服务器发送请求时的意向说明，以此告知服务器该如何处理及解析提交的这些数据，而客户端提交的数据也会因为采用的提交方式不同来处理。

区分请求种类也会为浏览器采用不同的缓存方式处理后续请求提供一个依据，从而提升响应速度。所以对于编程人员来讲，使用正确的请求方式是Web应用的一个重要方面。

1.1.2. GET请求方式的特点

使用GET提交数据时，form中的数据将编码到URL中。在使用上，当且仅当请求幂等时使用GET。重复访问时使用GET方法请求的页面，浏览器会使用缓存处理后续请求。

•GET是把参数数据队列加到提交表单的ACTION属性所指的URL中，值和表单内各个字段一一对应，在URL中可以看到。

•GET传送的数据量较小，不能大于2KB。

•GET安全性非常低

•<form method="get" action="xxx?b=b">跟<form method="get" action="xxx">是一样的，也就是说，action页面后边带的参数列表会被忽视；而<form method="post" action="xxx?b=b">跟<form method="post" action="xxx">是不一样的。

如下情况中浏览器会发送GET类型的请求：

•在地址栏输入一个地址

•点击页面中的链接

•表单的默认提交方式

1.1.3. POST请求方式的特点

POST方式提交的数据是在HTTP协议的Hearder中进行传输的。在使用上，当请求会改变服务器数据或状态时（更新数据，上传文件）应该使用POST。

从使用经验上有如下总结：

•POST是将表单内各个字段与其内容放置在实体内容中内一起传送到ACTION属性所指的URL地址。用户看不到这个过程。

•POST传送的数据量较大，一般被默认为不受限制

•POST安全性较高

将表单的Method属性设置为POST时，浏览器会发送POST请求。

1.2. Servlet如何运行

1.2.1. Servlet运行的步骤

Servlet作为Web服务器的补充功能在运行时需要受到Servlet容器的管理，其运行的流程如下：

1.浏览器依据IP建立与容器的连接

2.浏览器将请求数据打包

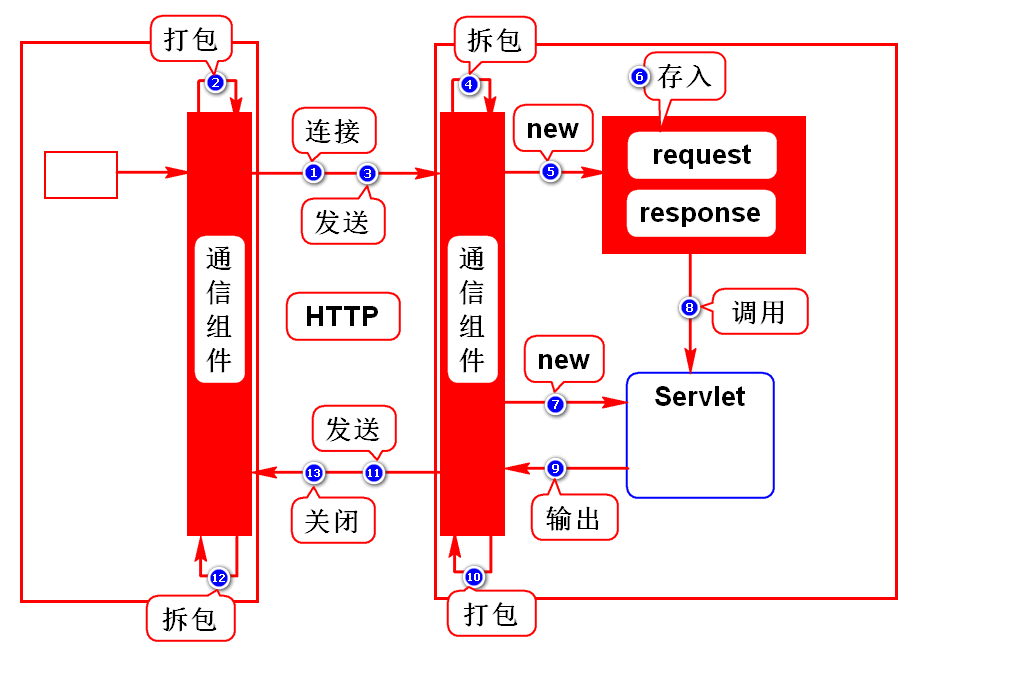
3.容器解析请求数据包，封装request和response对象

4.容器依据请求资源路径找到Servlet创建对象

5.容器将request和response对象作为参数传入service方法，并调用

6.容器将响应数据打包发给浏览器

7.浏览器取出结果，生成页面



1.3. Servlet如何输出中文

1.3.1. 为什么返回的页面会有乱码

Java语言在内存当中保存的每一个字符使用的都是Unicode字符集。一个中文字在内存中使用2个字节来表示。

编码：将Unicode字符集对应的字节数组转换成某种本地字符集（比如UTF-8，GBK）对应的字节数组的过程叫做编码

解码：将某种本地字符集对应的字节数组转换成Unicode字符集对应的字节数组的过程。

如果想输出就涉及到编码问题。编码和解码使用的字符集不一致就产生了乱码问题。

out.println（“”）这行代码是编码过程。现在这个out要输出的内容使用的是Unicode字符集，于是转换为默认的iso-8859-1。之所以使用这个格式是因为tomcat默认是英文的。所以调用out.println时，容器采用默认情况下的ISO-8859-1字符集去编码带有中文的输出内容时就产生了乱码。

1.3.2. 如何解决输出内容的乱码问题

在service（）方法中第一行的位置上添加如下代码：

01.response.setContentType(“text/html;charset=UTF-8”);

其中charset可以使用其他支持中文的字符集，如GBK。setContentType（）有两个作用：

1.通知容器，在调用out.println方法输出时，使用指定的字符集

2.生成一个消息头（content-type），通知浏览器，服务器返回的数据类型和字符集

使用该段代码修改默认的编码方式时，一定要保证在调用print之前编写，所以该段代码尽量放在service方法的第一行的位置。在charset之前使用的是分号隔开，如果写错，则会出现保存文件的界面，原因是浏览器不能识别消息头的值，于是让用户来处理。

能够输出带有中文内容页面的完整代码如下：

09.public class HelloServlet extends HttpServlet{

10. public void service(HttpServletRequest request,

11.HttpServletResponse response)throws ServletException,

12.IOException{

13. response.setContentType("text/html;charset=utf-8");

14. //获得输出流

15. PrintWriter out = response.getWriter();

16. out.println(“达内科技欢迎你”);

17. //关闭流

18. out.close();

19. }

20.}

1.4. Servlet如何处理表单

1.4.1. 如何获得请求参数值

对于表单提交的数据，Servlet可以从容器构建的request对象中获取，如下两个方法可以在不同情况下获取表单数据：

•String getParameter（String paramName）

•String[] getParameterValues（String paramName）

1.4.2. getParameter方法

主要用于获取表单中控件的数据。其中参数名一定要与客户端表单中的控件name属性相一致。所以在构建表单各元素时，name属性一定要有。而name属性和id属性的区别就在于，id属性一般是作为客户端区分控件的标识，name属性是服务器端区分各控件的标识。如果参数名写错，则该方法会返回null。

用于提供用户输入的HTML页面代码如下：

01.<html>

02. <head>

03. </head>

04. <body>

05. <form action="hello" method="post">

06. 用户名:<input name="username"/><br/>

07. <input type="submit" value="提交"/>

08. </form>

09. </body>

10.</html>

使用该方法获取表单数据的完整代码如下：

01.package web;

09.public class HelloServlet extends HttpServlet{

10. public void service(HttpServletRequest

11. request,HttpServletResponse response)

12. throws ServletException,IOException

14. \* request.getParameter方法：

15. \* 1.作用：依据请求参数名获取请求参数值。

17. String username = request.getParameter("username");

18. //处理请求

19. String msg = "<div style='font-size:80px'>" +

20. "你好 " + username + "</div>";

21. response.setContentType("text/html;charset=utf-8");

22. //获得输出流

23. PrintWriter out = response.getWriter();

24. out.println(msg);

25. //关闭流

26. out.close();

27. }

28.}

1.4.3. getParameterValues方法

用于获取表单中一组参数名相同的控件提交的数据组，如复选框，所以该方法返回的是字符串数组。如果用户没有选择这一组同名控件的任何一个，则该方法返回null。

用于提供用户输入用户名和爱好选项的HTML页面完整代码如下：

01.<html>

02. <head>

03. </head>

04. <body>

05. <form action="hello" method="post">

06. 用户名:<input name="username"/><br/>

07. 爱好:

08.吃饭<input type="checkbox" name="hobby" value="eating" checked="checked"/>

09. 睡觉<input type="checkbox" name="hobby" value="sleeping"/>

10. 打豆豆<input type="checkbox" name="hobby" value="fighting"/>

11.<br/>

12. <input type="submit" value="提交"/>

13. </form>

14. </body>

15.</html>

使用该方法获取复选框值的完整代码如下：

08.public class HelloServlet extends HttpServlet{

09. public void service(HttpServletRequest

10. request,HttpServletResponse response)throws ServletException,IOException{

11. String username =request.getParameter("username");

12. String[] hobby= request.getParameterValues("hobby");

13. //处理请求

14. String msg = "<div style='font-size:80px'>" +

15. "你好 " + username + "</div>";

16. response.setContentType("text/html;charset=utf-8");

17. PrintWriter out = response.getWriter();

18. out.println(msg);

19. for(int i=0;i<hobby.length;i++){

20. out.println("<h1>" + hobby[i] +"</h1>");

21. }

22. //关闭流

23. out.close();

24. }

25.}

1.4.4. 为什么表单提交中文会出现乱码

当表单提交时，浏览器会对表单中的中文参数值进行编码，而使用的编码是打开页面时所使用的字符集，如当前页面使用的UTF-8的字符集进行显示的，那么表单提交的数据就会以UTF-8的方式进行编码后传输，而Web服务器在默认情况下对提交的表单数据会使用ISO-8859-1的字符集来解码，编码与解码的方式不一致就产生了表单提交时的中文乱码问题。

1.4.5. 如何解决表单提交时的中文乱码问题

步骤一、确保表单所在的页面按照指定的字符集打开

在HTML页面中使用meta标记可以确保浏览器按照指定的字符集进行解码页面，并限定表单提交时的数据编码方式

01.<meta http-equiv="content-type" content="text/html;charset=utf-8">

在服务器端需要在调用getParameter方法读取参数之前，告诉浏览器如何解码，使用如下代码即可完成该设置：

01.request.setCharacterEncoding("utf-8");

注意该方法一定要要放在所有request.getParameter方法之前。

这种方式只针对POST请求有效。

1.4.6. 解决GET方式提交时的中文乱码问题

步骤一、确保表单所在的页面按照指定的字符集打开

在HTML页面中使用meta标记可以确保浏览器按照指定的字符集进行解码页面，并限定表单提交时的数据编码方式

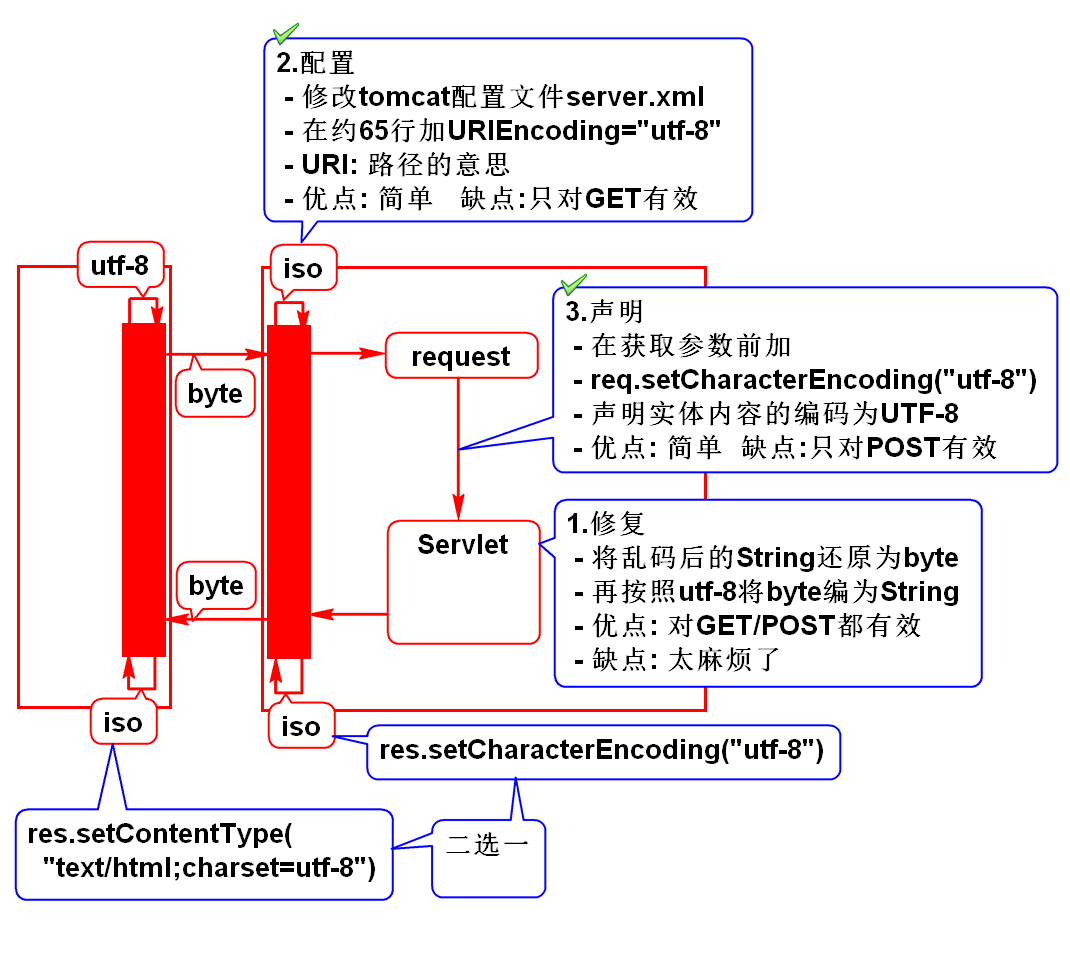
01.<meta http-equiv="content-type" content="text/html;charset=utf-8">

步骤二、完成ISO-8859-1到UTF-8格式的转换

01.String username = request.getParameter(“username”);

02.username = new String(username.getBytes(“iso-8859-1”),“UTF-8”);

由于GET方式提交的任何数据在服务器端一定会以ISO-8859-1的方式进行解码，所以服务器端可以先按ISO-8859-1的方式获取字节数组，在将该数组转变成UTF-8对应的字符串形式。



* Servlet开发要点

1.1. 重定向

1.1.1. 什么是重定向

在服务器为浏览器提供响应时，回传的数据包中的状态行里面是302状态码，同时在消息头内会增加一个键值对，名称为Location，值是一个新的URL地址。当这个响应到达浏览器的时候，这一次的请求响应过程并未结束，浏览器遇见302状态码之后，会立即按照Location头信息中指定的URL地址发送新的一个请求，这样一个在接到响应后又立即发出请求的过程叫做重定向。对于客户端用户来讲，中间的变化过程不会被察觉，因为这个过程是由浏览器自动完成的。

1.1.2. 重定向原理

在重定向的过程中，影响浏览器做出动作的关键点即响应中的状态码及Location这个消息头。302状态就像一道命令一样，使得浏览器做出新的一次请求，而请求的地址会从头信息中查找。由于这个新的请求动作是由浏览器发出的，所以浏览器的地址栏上的地址会变成Location消息头中的地址。

1.1.3. 如何重定向

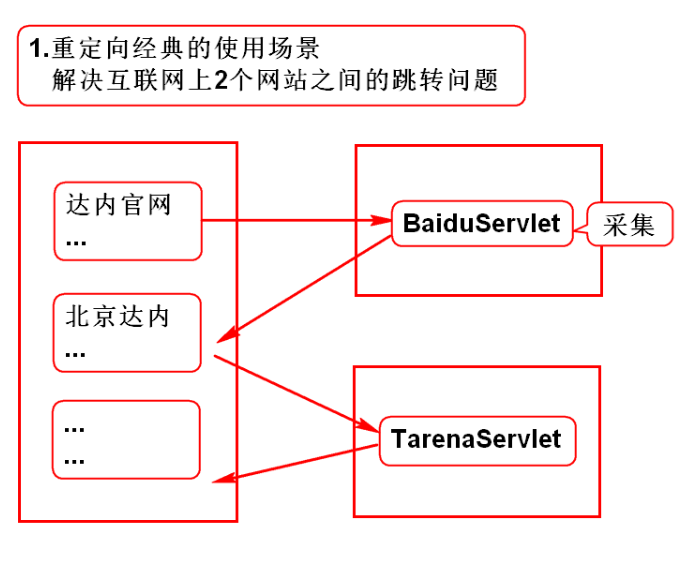
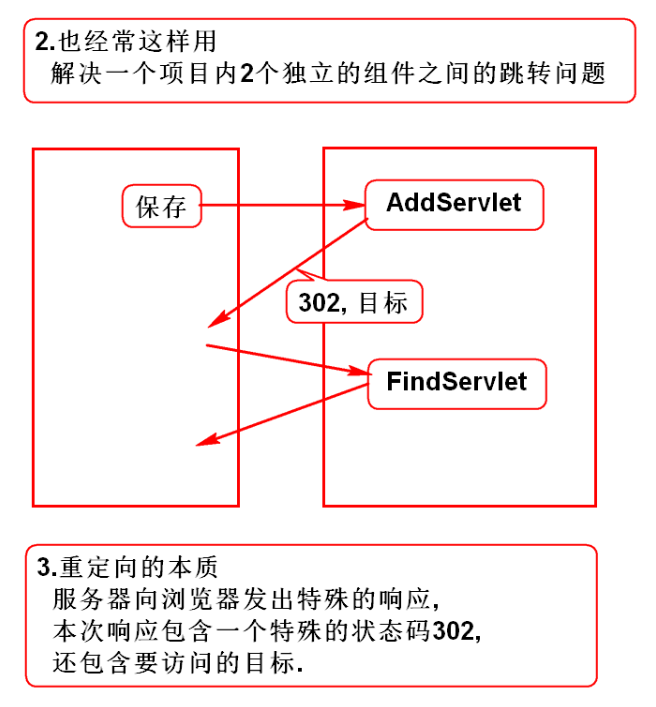
由于发回的响应信息由response对象控制，所以使用如下代码即可实现重定向的过程：

01.response.sendRedirect(String url);

该方法的参数值url即Location消息头中的重定向地址。注意，该段代码后面如果还有其他代码的话也会被继续执行的。

1.1.4. 重定向特点

由于重定向动作的执行者为浏览器，所以请求的地址可以是任意地址，哪怕是当前应用以外的应用；浏览器发出请求时一定会保持地址栏与目标地址的一致，所以发生重定向时可以从地址栏中看到地址的改变；由于整个跳转过程是在浏览器收到响应后重新发起请求，所以涉及到的Web组件并不会共享同一个request和response。

1.2. Servlet容器如何处理请求资源路径

1.2.1. 什么是请求资源路径

在地址栏中输入的请求地址中，端口号之后的部分都是请求资源路径。紧跟端口号的是部署到Web服务器上的应用名（appName），紧跟应用名的则是具体的应用内的组件路径。

1.2.2. Web服务器对请求地址的处理过程

浏览器依据地址中的IP和端口号与Web服务器建立连接，服务器会获取到请求资源路径信息。根据端口号后面的应用名找到服务器上对应的应用。默认情况下容器会认为应用名后面的是一个Servlet，所以回到web.xml文件中所有是否有与该值匹配的<url-pattern>，找到匹配的值之后再按照<servlet-name>完成对应关系的查找，进而找到要执行的Servlet。如果没有找到匹配的资源服务器就会返回404错误。

1.2.3. 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。这些原则主要有：

精确匹配

即具体资源名称与web.xml文件中的url-pattern严格匹配相等才执行。

通配符匹配

使用“\*”这个符号来匹配0个或多个字符，已达到路径的批量匹配的效果。

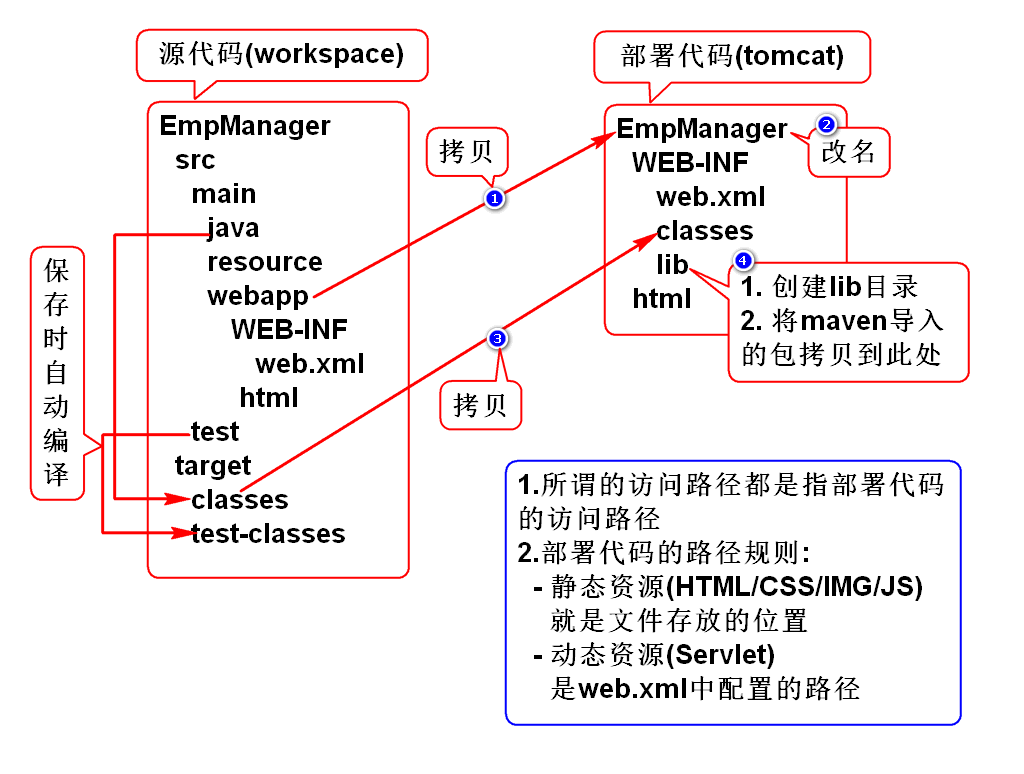
在地址栏中输入任何地址时都是匹配成功的。

后缀匹配

在配置url-pattern节点时，不使用斜杠开头，用“\*.”开头来匹配任意多个字符的模式叫做后缀匹配。

在地址栏中输入任何地址时都是匹配成功的。

在这三种匹配方式中，优先级最高的是精确匹配。如果容器在使用以上原则都不能找到相匹配的资源来执行时，就按照地址到应用中查找对应的文件。此时如果找到文件则返回，找不到资源来执行就返回404错误。



1.3. 一个Servlet实现多请求

1.3.1. 为什么要将多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。

要想完成多个Servlet合并为一个Servlet，需要完成以下两个步骤：

•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

1.3.2. 使用后缀匹配模式完成请求资源路径的匹配

修改web.xml文件，将更多的servlet配置节点删除，只保留一个节点即可，代码如下：

07. <servlet>

08. <servlet-name>someServlet</servlet-name>

09. <servlet-class>web.SomeServlet</servlet-class>

10. </servlet>

11. <servlet-mapping>

12. <servlet-name>someServlet</servlet-name>

13. <url-pattern>\*.do</url-pattern>

14. </servlet-mapping>

15.</web-app>

1.3.3. 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。使用如下代码逻辑来完成分发动作。

10.public class SomeServlet extends HttpServlet{

11. public void service(HttpServletRequest request,

12. HttpServletResponse response) throws

13.ServletException,IOException{

14. //获得请求资源路径

15. String uri = request.getRequestURI();

16. System.out.println("uri:" + uri);

17. if(uri.equals("/test/list.do")){

18. System.out.println("进行员工列表的处理...");

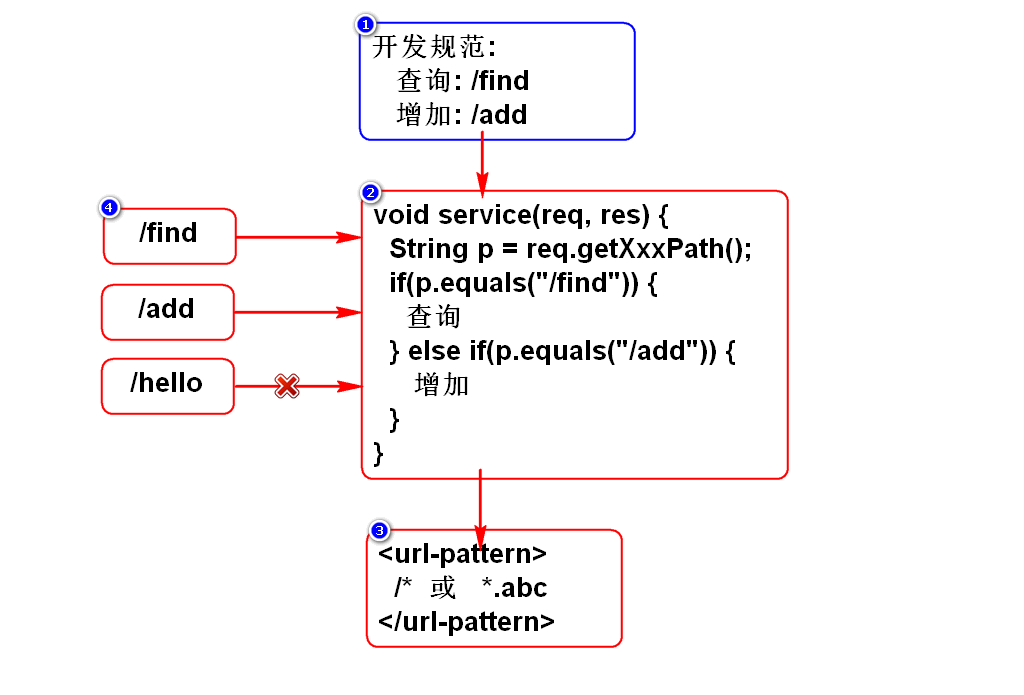
19. }else if(uri.equals("/test/add.do")){

20. System.out.println("添加员工的处理...");

21. }

22. }

23.}



1.4. Servlet的生命周期

1.4.1. 什么是Servlet生命周期

Servlet容器如何创建Servlet对象、如何为Servlet对象分配、准备资源、如何调用对应的方法来处理请求以及如何销毁Servlet对象的整个过程即Servlet的生命周期。

1.4.2. 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。

01.<servlet>

02. <servlet-name>someServlet</servlet-name>

03. <servlet-class>test/SomeServlet</servlet-class>

04. <load-on-startup>1</load-on-startup>

05.</servlet>

06.<servlet-mapping>

07. <servlet-name>someServlet</servlet-name>

08. <url-pattern>/\*</url-pattern>

09.</servlet-mapping>

配置文件中的load-on-startup节点用于设置该Servlet的创建时机。

当其中的值大于等于0时，表示容器在启动时就会创建实例

小于0时或没有指定时，代表容器在该Servlet被请求时再执行创建

正数的值越小，优先级越高，应用启动时就越先被创建。

* **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。

注：init方法只被执行一次。

以下代码为在servlet配置中，增加初始化参数

07. <servlet>

08. <servlet-name>someServlet</servlet-name>

09. <servlet-class>test/SomeServlet</servlet-class>

10. <init-param>

11. <param-name>debug</param-name>

12. <param-value>true</param-valule>

13. </init-param>

14. </servlet>

15. <servlet-mapping>

16. <servlet-name>someServlet</servlet-name>

17. <url-pattern>/\*</url-pattern>

18. </servlet-mapping>

19.</web-app>

使用以下代码可以读取Servlet配置中增加的初始化参数

11.public class SomeServlet extends HttpServlet{

13. public void service(HttpServletRequest request,

14. HttpServletResponse response)

15. throws ServletException,IOException{

16. System.out.println("SomeServlet's service...");

17. ServletConfig config = getServletConfig();

18. String debug = config.getInitParameter("debug");

19. System.out.println("debug:" + debug);

20. }

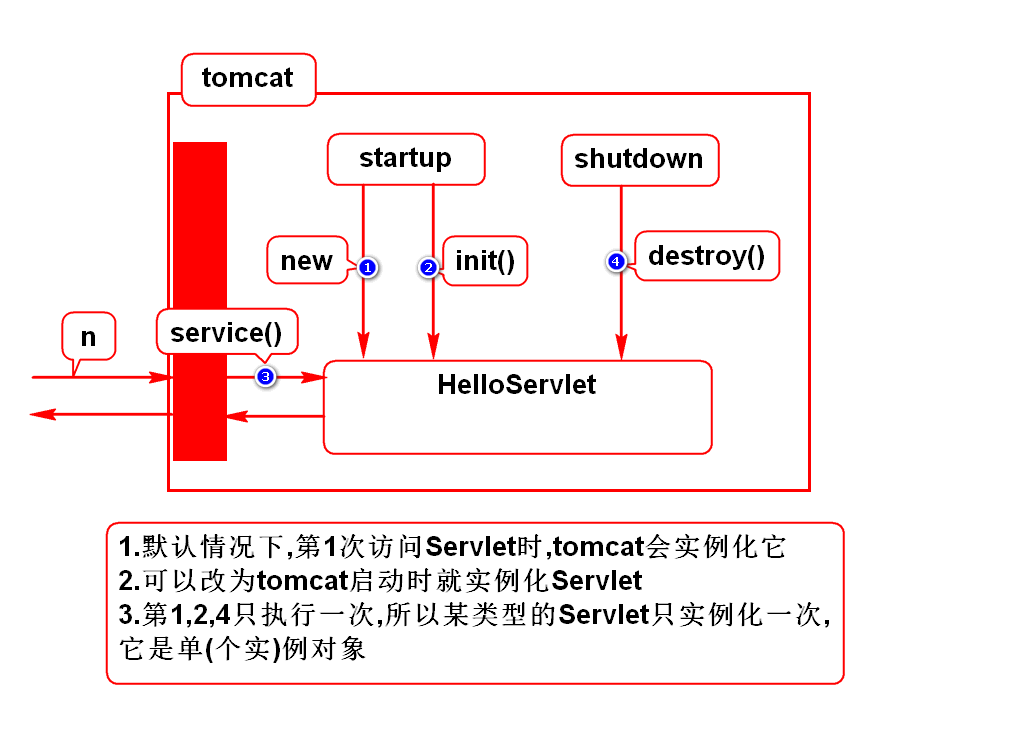
21.}

* **阶段三、就绪**

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。

* **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。



1.4.3. 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。

Servlet接口包括以下四个方法：

•init(ServletConfig config)

•service(ServletRequest req,ServletResponse res)

•destroy( )

1.4.4. Servlet涉及到的抽象类

Servlet API中另一个重要的类就是GenericServlet这个抽象类，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。

01.GenericServlet{

02. ServletConfig config；

03. public void init（）

04. { } //此方法什么也没做，可以说是为编程人员预留的接口

05. public void init（ServletConfig config{

07. this.config=config;

08. this.init();

09. }

10. getServletConfig（）{

12. return config；

13. }

14.}

1.5. ServletConfig和ServletContext

1.5.1. 什么是Servlet上下文

WEB容器在启动时，它会为每个WEB应用程序都创建一个对应的ServletContext对象，它代表当前web应用，是一个全局的环境变量。该应用中的任何组件，在任何时候都可以访问到该对象，所以Servlet上下文具有唯一性。

1.5.2. 如何获得Servlet上下文

获取该对象的方式有以下四种：

•GenericeServlet（HttpServlet）提供的 getServletContext（）

•ServletConfig提供的 getServletContext（）

•HttpSession提供的 getServletContext（）

•FilterConfig提供的 getServletContext（）

1.5.3. Servlet上下文的作用及特点

Servlet上下文的作用：

•在Web应用范围内存取共享数据:如 setAttribute(),getAttribute()

•访问Web应用的静态资源：如getRealPath(String path)

•跨多个请求、用户和Servlet

1.5.4.ServletConfig和ServletContext

1.它们的作用

•都能够读取web.xml中为Servlet预置的参数

2.它们的区别

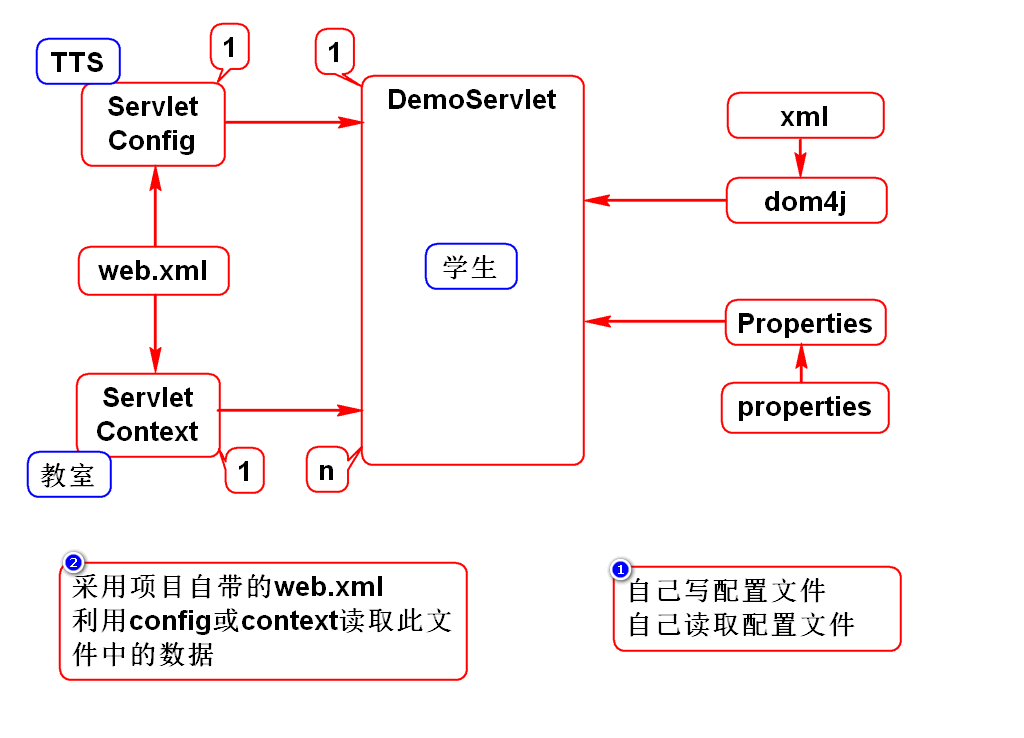
•config和Servlet是1对1的关系

•context和Servlet是1对多的关系

•若数据只给某个Servlet使用,则用config

•若数据给多个Servlet使用,则用context

它们的关系由服务器来保障



3.config使用场景

•假设要开发一个网页游戏,若超过人数上限则要排队

•开发登录功能LoginServlet

•人数上限应该是一个可配置的参数maxOnline

•该参数由LoginServlet使用

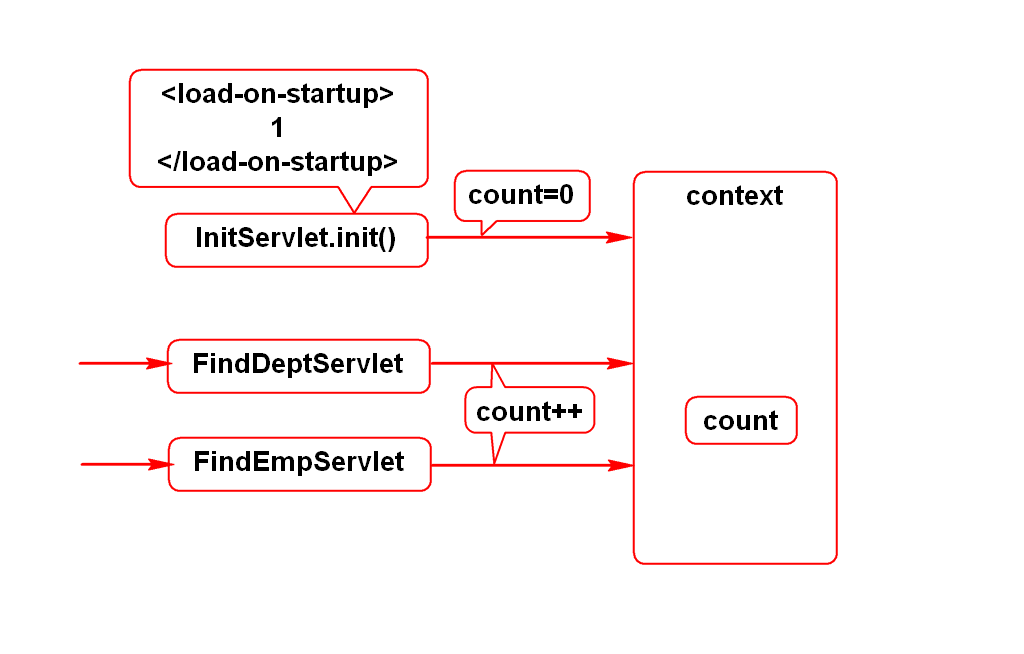
由于该参数只是LoginServlet使用,由config读取即可

4.context使用场景

•大部分的查询都具备分页功能

•分页需要一个参数:每页显示几条数据size

•该参数一般可配置,由于被众多查询功能复用,使用context读取



1.6. Servlet线程安全问题

1.6.1. 为什么会有线程安全问题

当浏览器访问服务器的通讯模块SomeServlet时，会启动一个线程T1来进行一系列的创建动作来处理这个请求。一般的web服务器的编程模型如下：

01. while（flag）{

02. Socket s = ss.accept();

03. Thread t = new Thread(s);

04. t.start();}

如果刚好同时也有一个请求来访问SomeServlet，但是服务器只有一个servlet实例，所以服务器会启动线程T2，此时就有可能产生T1和T2同时访问someservlet的情况，如果要修改属性就会有安全隐患

1.6.2. 如何保证Servlet线程安全

使用synchronized对代码加锁即可。

