**什么是servlet**

servlet是一门用于开发动态web资源的技术，可以运行在Web服务器中的小型Java程序，有时也叫做服务器端的小应用程序。servlet 可以通过 HTTP协议接收和响应来自 Web 客户端的请求。

**Servlet生命周期**

初始化阶段  调用init()方法

响应客户请求阶段　　调用service()方法

终止阶段　　调用destroy()方法

servlet的特征

Servlet是单例多线程的，只创建一个servlet对象，但是每次请求都会起一个线程并在自己线程栈内存中执行service方法。

一个 Servlet 实例只会执行一次无参构造器与 init()方法，并且是在第一次访问时执行。

用户每提交一次对当前 Servlet 的请求，就会执行一次 service()方法。

一个 Servlet 实例只会执行一次 destroy()方法，在应用停止时执行。

由于 Servlet 是单例多线程的，所以为了保证其线程安全性，一般情况下是不建议在 Servlet类中定义可修改的成员变量，因为每个线程均可修改这个成员变量，会出现线程安全问题。

默认情况下，Servlet 在 Web 容器启动时是不会被实例化的。

添加load-on-startup的作用是，标记是否在 Tomcat启动时创建并初始化这个 Servlet实例。它的值必须是一个整数。

当值大于等于 0 时，表示容器在启动时就加载并初始化这个 Servlet，数值越小，该 Servlet的优先级就越高，其被创建的也就越早；

当值相同时，容器会自己选择创建顺序。

**ServletConfig的特点**

每一个servlet都对应一个ServletConfig用于封装各自的配置信息，即有几个servlet就会产生几个ServletConfig对象。

**ServletContext接口简介**

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，ServletContext对象包含Web应用中所有 Servlet 在 Web 容器中的一些数据信息。ServletContext随着Web应用的启动而创建，随着 Web 应用的关闭而销毁。一个 Web 应用只有一个ServletContext 对象。

ServletContext中不仅包含了 web.xml 文件中的配置信息，还包含了当前应用中所有Servlet可以共享的数据。可以这么说， ServeltContext 可以代表整个应用，所以ServletContext有另外一个名称：application。

**ServletContext中常用方法**

ServletConfig对象中维护了ServletContext对象的引用，开发人员在编写servlet时，可以通过ServletConfig.getServletContext()方法获得ServletContext对象。

String getInitParameter ()：获取 web.xml 文 件 的 中 指 定 名 称 的上下文参数值 。

Enumeration getInitParameterNames()：获取 web.xml 文件的中的所有的上下文参数名称。其返回值为枚举类型 Enumeration。

void setAttribute(String name, Object object)：在 ServletContext 的公共数据空间中，也称为域属性空间，放入数据。这些数据对于 Web应用来说，是全局性的，与整个应用的生命周期相同。当然，放入其中的数据是有名称的，通过名称来访问该数据。

Object getAttribute(String name)：从 ServletContext 的域属性空间中获取指定名称的数据。

void removeAttribute(String name)：从 ServletContext 的域属性空间中删除指定名称的数据。

String getRealPath(String path)：获取当前 Web 应用中指定文件或目录在本地文件系统中的路径。

String getContextPath()：获取当前应用在 Web 容器中的名称。

**设置欢迎页面**

在浏览器地址栏中直接通过项目名称访问时，默认显示的页面就是欢迎页面，可以是.html.jsp，可以通过welcome-file-list进行设置。可以为应用设置多个欢迎页面，但只会有一个起作用，系统加载这些欢迎页面的顺序与其代码的顺序相同，即由上到下逐个查找，一旦找到，则马上显示，不会再向下查找。

如果当前应用没有指定欢迎页面，则系统会从当前项目的根目录下依次查找 index.html、 index.htm

及 index.jsp 文件，如果这些文件不存在的话，浏览器会报出 404 错误。

**url-pattern的设置**

url-pattern标签用于对请求进行筛选匹配，对当前注册的 Servlet 所要处理的请求类

型进行筛选。对于url-pattern中路径的写法，有多种不同模式，表示不同的意义，一个Servlet可以对应多个url-pattern.

**精确路径模式**

请求路径必须与url-pattern的值完全相同才可被当前 Servlet 处理。

**通配符路径模式**

该模式中的路径由两部分组成：精确路径部分与通配符部分。请求路径中只有携带了url-pattern值中指定的精确路径部分才可被当前 Servlet 处理。

**全路径模式**

提交的所有请求全部可被当前的 Servlet 处理。其值可以指定为/\*，也可指定为/。

**后辍名模式**

请求路径最后的资源名称必须携带中指定的后辍名，其请求才可被当前Servlet 处理

/\*与/表示所有请求均会被当前 Servlet 所处理。如果一个servlet的url-pattern是/\*或/，则该servlet表示默认映射，当一个请求找不到相应的url的servlet时，系统会调用这个默认映射的servlet。

这两个路径的不同之处是：

使用/\*，表示当前的 Servlet 会拦截用户对于静态资源（.css、.js、.html、.jpg、.png…..）

与动态资源（.jsp）的请求。即用户不会直接获取到这些资源文件，而是将请求交给当前 Servlet

来处理了。

使用/，表示当前的 Servlet 会拦截用户对于静态资源（.css、.js、.html、.jpg、.png…..），

但对于动态资源的请求，是不进行拦截的。即用户请求的静态资源文件是不能直接获取到的。

综上所述，对于 Servlet 的url-pattern的设置，我们一般是不会将其指定为/\*或/的。

一旦有一个 Servlet 的url-patter被设置为了/\*或/，则整个应用的静态资源将可能无法正常显示。

**url-pattern路径优先级**

**路径优先后辍匹配原则**

**精确路径优先匹配原则**

**最长路径优先匹配原则**

**实际开发中经常使用继承HttpServlet类的方式创建一个servlet**

**HttpServlet的子类不需要重写service方法，倘若重写了该方法后可能会导致所编写的Servlet无法正常工作。**

**创建servlet的三种方式**

定一个类实现javax.servlet.Servlet接口

定义一个类继承javax.servet.GenericServlet类

定义一个类继承javax.servlet.http.HttpServlet类

**HttpServletRequest简介**

Web服务器收到客户端的http请求，会针对每一次请求，创建一个用于代表请求的HttpServletRequest类型的request对象，并将"HTTP请求协议"的完整内容封装到该对象中。开发者获拿到request对象后就可以获取客户端发送给服务器的请求数据了。

**HttpServletRequest的生命周期**

当客户端浏览器向服务器发送请求后，服务器会根据HTTP请求协议的格式对请求进行解析。同时，服务器会创建 HttpServletRequest类型的对象，即请求对象，然后将解析出的数据封装到该请求对象中。此时HttpServletRequest实例就创建并初始化完毕了，也就是说，请求对象是由服务器创建。当服务器向客户端发送响应结束后，HttpServletRequest 实例对象被服务器销毁，HttpServletRequest对象的生命周期很短暂。

一次请求对应一个请求对象， 另外一次请求对应另外一个请求对象，即每次请求都会创建一个HttpServletRequest类型的对象，这些对象之间没有关系。

**HttpServletRequest中常用的方法**

Map getParameterMap()

获取包含所有请求参数及值的 Map 对象。需要注意，该 Map 的 value 为 String[]，即一个参数所对应的值为一个数组。说明一个参数可以对应多个值。

Enumeration getParameterNames()

获取请求参数 Map 的所有 key,即获取所有请求参数名。

String[] getParameterValues(String name)

根据指定的请求参数名称，获取其对应的所有值。这个方法一般用于获取复选框(checkbox)数据。

String getParameter(String name)

根据指定的请求参数名称，获取其对应的值。若该参数名称对应的是多个值，则该方法获取到的是第一个值。这个方法是最常用的方法。

**获取客户端信息的方法：**

getRequestURL方法返回客户端发出请求时的完整URL。

getRequestURI方法返回请求行中的资源名部分。

getQueryString 方法返回请求行中的参数部分。

getRemoteAddr方法返回发出请求的客户机的IP地址

getRemoteHost方法返回发出请求的客户机的完整主机名

getRemotePort方法返回客户机所使用的网络端口号

getLocalAddr方法返回WEB服务器的IP地址。

getLocalName方法返回WEB服务器的主机名

getMethod得到客户机请求方式

**乱码的产生原因**

当用户通过浏览器提交一个包含 UTF-8 编码格式的两个中文请求时，浏览器会将这两个中文字符变为六个字节（一般一个 UTF-8 汉字占用三个字节），即形成六个类似%8E 的字节表示形式，并将这六个字节上传至 Tomcat 服务器。

Tomcat 服务器在接收到这六个字节后，并不知道它们原始采用的是什么字符编码。而Tomcat 默认的编码格式为 ISO-8859-1。所以会将这六个字节按照 ISO-8859-1 的格式进行解码，解码后在控制台显示，所以在控制台会显示乱码。

**针对 POST 提交乱码的解决方式**

在接收请求参数之前先通过 request 的 setCharacterEncoding()方法，指定请求体的字符编码格式。这样的话，在接收到请求中的参数后，就可按照指定的字符编码进行解码。

注意，request 的 setCharacterEncoding()方法只能解决 POST 提交方式中的乱码问题，对

于 GET 提交方式的不起作用。因为该方法设置的是请求体中的字符编码， GET 提交中的参数不出现在请求体中，而出现在请求行。

**针对get提交乱码的解决方式**

可以通过修改 Tomcat 默认字符编码的方式来解决 GET 提交方式中携带中文的乱码问题。在 Tomcat 安装目录的 conf/server.xml 中，找到端口号为 8080 的标签，在其中添加 URIEncoding=”UTF-8″的设置，即可将 Tomcat 默认字符编码修改为 UTF-8。

**万能解决方案**



**代码量较大，开发中使用较少。**

**HttpServletResponse简介**

Web服务器收到客户端的http请求，会针对每一次请求，创建一个用于代表响应的HttpServletResponse类型的response对象，开发者可以将要向客户端返回的数据封装到response对象中。

**HttpServletResponse向客户端发送数据**

ServletResponse 接口有一个方法 getWriter()，用于获取到一个输出流对象 PrintWriter，

该输出流对象是专门用于向客户端浏览器中输出字符数据的，称为标准输出流。

**HttpServletResponse响应乱码的解决方案**

响应时会产生乱码的原因是在 HTTP 协议中规定，默认响应体的字符编码为ISO-8859-1。所以，若要解决乱码问题，就需要修改响应体的默认编码。一般情况下，有两种方式可以修改：

**方法一**：HttpServletResponse 的 setCharacterEncoding(“utf-8”)方法，将编码修改为utf-8，然后再通过setHead(“Content-type”,”text/html;charset=UTF-8″);方法告诉客户端浏览器的编码方式。

**方法二**：为了简便操作，开发者可以直接使用HttpServletResponse 的 setContentType(“text/html;charset=utf-8”)方法，告诉浏览器的编码方式，该方法相当于方法一种的两条代码。

**注意：设置响应编码时必须在 PrintWriter 对象产生之前先设置，否则将不起作用。**

**Jsp与servlet的区别**

jsp的本质就是servlet，是servlet的一种简化，jsp经编译后就变成了Servlet

jsp更擅长表现于页面显示,servlet更擅长于逻辑控制.

Servlet中没有内置对象，Jsp中的内置对象都是必须通过HttpServletRequest对象，HttpServletResponse对象以及HttpServlet对象得到.

**JSP**

**隐式对象**

1、request            请求对象　         作用域 Request

　2、response   响应对象           作用域  Page

　3、pageContext    页面上下文对象      作用域    Page

　4、session            会话对象               作用域    Session

　5、application       应用程序对象          作用域    Application

   6、out                 输出对象                 作用域    Page

　7、config               配置对象                 作用域    Page

　8、page                页面对象                  作用域    Page

　9、exception 例外对象                 作用域    page

**jsp有四种属性范围：**

|  |  |
| --- | --- |
| 名称 | 作用域 |
| application | 在所有应用程序中有效 |
| session | 在当前会话中有效 |
| request | 在当前请求中有效 |
| page | 在当前页面有效 |

**page -> 页面级别，显然只有在一个页面内可用。**

page：当前页面从打开到关闭这段时间。

**request -> 请求级别 服务器跳转，一次请求之后消失。**

request：HTTP请求开始到结束这段时间。

**session -> 会话级别 客户端跳转（服务器跳转）**

session：HTTP会话开始到结束这段时间。

**application = 应用级别，当重启服务器时才会消失**

application：服务器启动到停止这段时间。

### 页面间对象传递的方法

request，session，application，cookie

**转发与重定向的区别**

转发是服务器行为，重定向是客户端行为。

转发的速度快;重定向速度慢

转发地址栏没有变化;重定向地址栏有变化

转发的是同一次请求;重定向是两次不同请求

转发不会执行转发后的代码;重定向会执行重定向之后的代码

转发地址栏没有变化;重定向地址栏有变化

转发必须是在同一台服务器下完成;重定向可以在不同的服务器下完成

转发页面和转发到的页面可以共享request里面的数据. 重定向不能共享数据

### servlet的特征

* Servlet是单例多线程的，只创建一个servlet对象，但是每次请求都会起一个线程并在自己线程栈内存中执行service方法。
* 一个 Servlet 实例只会执行一次无参构造器与 init()方法，并且是在第一次访问时执行。
* 用户每提交一次对当前 Servlet 的请求，就会执行一次 service()方法。
* 一个 Servlet 实例只会执行一次 destroy()方法，在应用停止时执行。
* 由于 Servlet 是单例多线程的，所以为了保证其线程安全性，一般情况下是不建议在 Servlet类中定义可修改的成员变量，因为每个线程均可修改这个成员变量，会出现线程安全问题。
* 默认情况下，Servlet 在 Web 容器启动时是不会被实例化的。

load-on-startup标签的作用是，标记是否在 Tomcat启动时创建并初始化这个 Servlet实例。它的值必须是一个整数。

* 当值大于等于 0 时，表示容器在启动时就加载并初始化这个 Servlet，数值越小，该 Servlet的优先级就越高，其被创建的也就越早；
* 当值相同时，容器会自己选择创建顺序。

简述web.xml的作用  
属于部署描述符，在整个JAVA中只要是容器都会存在部署描述符，此部署描述符可以控制整个WEB中各个组件的运行状态，也可以配置整个窗口的状态

### 创建servlet的三种方式

* 定一个类实现javax.servlet.Servlet接口
* 定义一个类继承javax.servet.GenericServlet类
* 定义一个类继承javax.servlet.http.HttpServlet类

**cookie 和session 的区别**

1、数据存放位置不同：

cookie数据存放在客户的浏览器上，session数据放在服务器上。

2、安全程度不同：

cookie不是很安全，别人可以分析存放在本地的COOKIE并进行COOKIE欺骗,考虑到安全应当使用session。

3、性能使用程度不同：

session会在一定时间内保存在服务器上。当访问增多，会比较占用你服务器的性能,考虑到减轻服务器性能方面，应当使用cookie。

4、数据存储大小不同：

单个cookie保存的数据不能超过4K，很多浏览器都限制一个站点最多保存20个cookie，而session则存储与服务端，浏览器对其没有限制。

### 动态引入和静态引入的区别

静态引入会生成一个java文件，两个jsp文件中可以共享同一个变量，但不能定义重名的变量。

动态引入会生成两个java文件，两个jsp文件中不可以共享同一个变量，可以定义重名的变量。

EL表达式只能从 pageConext、request、session、application 四大域属性空间中获取数据。

**request.getAttribute()和 request.getParameter()有何区别**

getParameter 返回的是String,用于读取提交的表单中的值;（获取之后会根据实际需要转换为自己需要的相应类型，比如整型，日期类型啊等等）

getAttribute 返回的是Object，需进行转换,可用setAttribute 设置成任意对象，使用很灵活，可随时用

**什么是MVC，对MVC的理解？**

MVC是一种设计模式，它强制的将应用程序的输入、处理、和输出分开。

使用了MVC的应用程序被分为3个核心部件:试图(View)、模型(Model)、控制器(Controller)。它们各司其职，既分工明确又相互合作。

首先视图接受用户输入请求，然后将请求传递给控制器，控制器再调用某个模型来处理用户的请求，在控制器的控制下，再将处理后的结果交给某个视图进行格式化输出给用户.这是经典的MVC设计执行的基本流程。MVC不仅实现了功能模块和显示模块的分离，同时它还提高了系统的可维护性、可扩展性和组件的可复用性，是一个优秀的创建软件的途径。

J2EE把MVC引入了web领域，并在此基础上架构出了一套称为Model2的体系。Model2体系及jsp作为视图显示输出和接受用户输入数据，Servlet作为控制器处理用户请求并根据用户请求选择相应的jsp返回给用户，javaBean作为模型组件用作业务逻辑处理、管理数据的存储和一致性。**MVC模式的缺点：**增加了程序源码的复杂性。

**MVC分层的优点**

1、开发人员可以只关注整个结构中的其中某一层；

2、可以很容易的用新的实现来替换原有层次的实现；

3、可以降低层与层之间的依赖；

4、有利于标准化；

5、利于各层逻辑的复用。

6、结构更加的明确

7、在后期维护的时候，极大地降低了维护成本和维护时间

**MVC分层的缺点**

1、降低了系统的性能。这是不言而喻的。如果不采用分层式结构，很多业务可以直接造访数据库，以此获取相应的数据，如今却必须通过中间层来完成。

2、有时会导致级联的修改。这种修改尤其体现在自上而下的方向。如果在表示层中需要增加一个功能，为保证其设计符合分层式结构，可能需要在相应的业务逻辑层和数据访问层中都增加相应的代码。

3、增加了开发成本。

MVC分层

1.

• 视图层view：用于展示数据，与用户进行交互。

• 控制层controller：用于分发控制到来的请求,并调用模型层与数据库进行交互,以及将数据返回给视图层展示。

• 模型层model：数据模型，它与数据库进行交互，进行CURD操作。

当请求来了，controller就会将相应的请求分发到相应的service层，在service层中再调用dao层进行数据库交互。这里的dao层其实就是之前的model层，封装了对数据库的操作。这样一来，就把业务处理逻辑从controller中分离出来，从而实现了解耦。

# 三层架构

# 三层架构(3-tier application) 通常意义上的三层架构就是将整个业务应用划分为：表现层（UI）、业务逻辑层（BLL）、数据访问层（DAL）。区分层次的目的即为了“高内聚，低耦合”的思想。

# 1、表现层（UI）：通俗讲就是展现给用户的界面，即用户在使用一个系统的时候他的所见所得。

# 2、业务逻辑层（BLL）：针对具体问题的操作，也可以说是对数据层的操作，对数据业务逻辑处理。

# 3、数据访问层（DAL）：该层所做事务直接操作数据库，针对数据的增添、删除、修改、更新、查找等。

三层架构的实现：

        表现层：UI层只能作为一个外壳，不能包含任何的逻辑处理过程。

        业务逻辑层：是系统架构中体现核心价值的部分，主要负责对数据层的操作。也就是说把一些数据层的操作进行组合。

         数据访问层：该层所做事务直接操作数据库，针对数据的增添、删除、修改、查找等具体为业务逻辑层或表示层提供数据服务。

**三层与MVC的区别：**

MVC是 Model-View-Controller，严格说这三个加起来以后才是三层架构中的UI层，也就是说，MVC把三层架构中的UI层再度进行了分化，分成了控制器、视图、实体三个部分，控制器完成页面逻辑，通过实体来与界面层完成通话；而C层直接与三层中的BLL进行对话。

mvc可以是三层中的一个表现层框架，属于表现层。三层和mvc可以共存。

三层是基于业务逻辑来分的，而mvc是基于页面来分的。

MVC主要用于表现层，3层主要用于体系架构，3层一般是表现层、中间层、数据层，其中表现层又可以分成M、V、C，(Model View Controller)模型－视图－控制器

MVC是表现模式（Presentation Pattern）

三层架构是典型的架构模式（Architecture Pattern）

三层架构的分层模式是典型的上下关系，上层依赖于下层。但MVC作为表现模式是不存在上下关系的，而是相互协作关系。即使将MVC当作架构模式，也不是分层模式。MVC和三层架构基本没有可比性，是应用于不同领域的技术。