知识汇总

目录

[知识汇总 1](#_Toc520213216)

[JAVA技术 3](#_Toc520213217)

[Servlet 3](#_Toc520213218)

[工作模式 3](#_Toc520213219)

[HttpServlet 3](#_Toc520213220)

[Servlet声明周期及工作原理 3](#_Toc520213221)

[JavaEE 4](#_Toc520213222)

[为什么需要JavaEE 4](#_Toc520213223)

[什么是JavaEE 4](#_Toc520213224)

[JavaEE包括的技术 5](#_Toc520213225)

[什么框架 5](#_Toc520213226)

[JavaEE与框架的关系 5](#_Toc520213227)

[JavaEE的体系结构 5](#_Toc520213228)

[JavaEE技术内容 6](#_Toc520213229)

[HTTP（HyperText Transfer Protocol，超文本传输协议）协议 7](#_Toc520213230)

[使用MyEclipse提供的数据监控工具TCP/IP Monitor来进行数据监控。 7](#_Toc520213231)

[get和post发送方式 8](#_Toc520213232)

[Java注解 8](#_Toc520213233)

[Web应用 9](#_Toc520213234)

[JAVA远程通讯 10](#_Toc520213235)

[1、 远程方法调用(RMI，英文：Remote Method Invocation) 10](#_Toc520213236)

[2.远程过程调用(RPC，英文：Remote Procedure Call) 12](#_Toc520213237)

[3.Java消息服务(JMS，英文：Java Message Service) 12](#_Toc520213238)

[数据结构 13](#_Toc520213239)

[排序 13](#_Toc520213240)

[前端相关 13](#_Toc520213241)

[JSP 13](#_Toc520213242)

[HTML 15](#_Toc520213243)

[Ajax 15](#_Toc520213244)

[JavaScript 15](#_Toc520213245)

[Worker 15](#_Toc520213246)

[服务器技术 15](#_Toc520213247)

[WebSphere 15](#_Toc520213248)

[Tomcat 15](#_Toc520213249)

[知识点 15](#_Toc520213250)

[负载均衡 15](#_Toc520213251)

[代理服务器 16](#_Toc520213252)

[Linux知识 16](#_Toc520213253)

[Linux标准输入、标准输出、标准错误与重定向介绍 16](#_Toc520213254)

[常用命令 18](#_Toc520213255)

[rm 18](#_Toc520213256)

[mkdir和touch 19](#_Toc520213257)

[tar 19](#_Toc520213258)

[vim 20](#_Toc520213259)

[mv、cp和rm 21](#_Toc520213260)

[sudo 22](#_Toc520213261)

[find和grep 22](#_Toc520213262)

[ls 23](#_Toc520213263)

[stat 24](#_Toc520213264)

[free和top 24](#_Toc520213265)

[ps -ef |grep和kill 24](#_Toc520213266)

[iptables 24](#_Toc520213267)

[cat 24](#_Toc520213268)

[lsof 25](#_Toc520213269)

[netstat 26](#_Toc520213270)

[分布式知识 26](#_Toc520213271)

[网络编程(计算机网络) 26](#_Toc520213272)

[TCP/IP协议 26](#_Toc520213273)

[OSI模型 27](#_Toc520213274)

[TCP/IP与OSI比较 28](#_Toc520213275)

[socket编程 28](#_Toc520213276)

[Socket编程模型 29](#_Toc520213277)

[前沿技术 29](#_Toc520213278)

[区块链 29](#_Toc520213279)

[比特币 29](#_Toc520213280)

[人工智能 29](#_Toc520213281)

[压力测试 29](#_Toc520213282)

[设计模式 29](#_Toc520213283)

[数据库技术 29](#_Toc520213284)

[存储过程 29](#_Toc520213285)

# JAVA技术

## Servlet

全称Java Servlet，未有中文译文。是用Java编写的服务器端程序。狭义的Servlet是指Java语言实现的一个接口，广义的Servlet是指任何实现了这个Servlet接口的类。

### 工作模式

客户端发送请求至服务器；

服务器启动并调用Servlet，Servlet根据客户端请求生成响应内容并将其传给服务器；

服务器将响应返回客户端；

其他；

### HttpServlet

javax.servlet.http.HttpServlet实现了专门用于响应HTTP请求的Servlet，提供了响应对应HTTP标准请求的doGet()、doPost()等方法。

### Servlet声明周期及工作原理

Servlet生命周期分为三个阶段：

　　1，初始化阶段  调用init()方法

　　2，响应客户请求阶段　　调用service()方法

　　3，终止阶段　　调用destroy()方法

Servlet初始化阶段：

　　在下列时刻Servlet容器装载Servlet：

　　　　1，Servlet容器启动时自动装载某些Servlet，实现它只需要在web.XML文件中的<Servlet></Servlet>之间添加如下代码：

　　　　2，在Servlet容器启动后，客户首次向Servlet发送请求

　　　　3，Servlet类文件被更新后，重新装载Servlet

　　Servlet被装载后，Servlet容器创建一个Servlet实例并且调用Servlet的init()方法进行初始化。在Servlet的整个生命周期内，init()方法只被调用一次。

Servlet工作原理：

　　首先简单解释一下Servlet接收和响应客户请求的过程，首先客户发送一个请求，Servlet是调用service()方法对请求进行响应的，通过源代码可见，service()方法中对请求的方式进行了匹配，选择调用doGet,doPost等这些方法，然后再进入对应的方法中调用逻辑层的方法，实现对客户的响应。在Servlet接口和GenericServlet中是没有doGet,doPost等等这些方法的，HttpServlet中定义了这些方法，但是都是返回error信息，所以，我们每次定义一个Servlet的时候，都必须实现doGet或doPost等这些方法。

　　每一个自定义的Servlet都必须实现Servlet的接口，Servlet接口中定义了五个方法，其中比较重要的三个方法涉及到Servlet的生命周期，分别是上文提到的init(),service(),destroy()方法。GenericServlet是一个通用的，不特定于任何协议的Servlet,它实现了Servlet接口。而HttpServlet继承于GenericServlet，因此HttpServlet也实现了Servlet接口。所以我们定义Servlet的时候只需要继承HttpServlet即可。

　　Servlet接口和GenericServlet是不特定于任何协议的，而HttpServlet是特定于HTTP协议的类，所以HttpServlet中实现了service()方法，并将请求ServletRequest,ServletResponse强转为HttpRequest和HttpResponse。

## JavaEE

### 为什么需要JavaEE

我们编写的JSP代码中，由于大量的显示代码和业务逻辑混淆在一起，彼此嵌套，不利于程序的维护和扩展。当业务需求发生变化的时候，对于程序员和美工都是一个很重的负担。

为了程序的易维护性和可扩展性，这就需要我们使用JavaEE技术来进行项目开发

### 什么是JavaEE

JavaEE是一个开发分布式企业级应用的规范和标准。

Java语言的平台有3个版本：适用于小型设备和智能卡的JavaME（Java Platform Micro Edition，Java微型版）、适用于桌面系统的JavaSE（Java Platform Micro Edition，Java标准版）、适用于企业应用的JavaEE（Java Platform Enterprise Edition，Java企业版）。

### JavaEE包括的技术

JavaEE应用程序是由组件构成的。J2EE组件是具有独立功能的单元，他们通过相关的类和文件组装成JavaEE应用程序，并与其他组件交互。

JavaEE包括的技术有：Web Service、Struts、Hibernate、Spring、JSP、Servlet、JSF、EJB、JavaBean、JDBC、JNDI、XML、JavaSE。

### 什么框架

框架即Framework，我们可以把框架理解为某种应用的半成品，就是一组组件，供用户选用，完成用户自己的系统。简单地说就是使用别人准备好的组件，来实施自己的项目。

### JavaEE与框架的关系

框架技术是在JavaEE的基础上形成的，而应用程序是在框架的基础上创建的。



### JavaEE的体系结构

应用程序的三层结构如下：

表示层：由用户界面和用户生成界面的代码组成。

中间层：包含系统的业务和功能代码。

数据层：负责完成存取数据库的数据和对数据进行封装。

三层体系结构的优点：一个组件的更改不会影响其他两个组件。例如：如果用户需要更换数据库，那么只有数据层组件需要修改代码。同样，如果更改了用户界面设计，那么只有表示层组件需要修改。

由于表示层和数据层相互独立，因而可以方便地扩充表示层，使系统具有良好的可扩展性。

代码重复减少，因为在3个组件之间尽可能地共享代码。

良好的分工与协作。这将使不同的小组能够独立地开发应用程序的不同部分，并充分发挥各自的长处和优势。

### JavaEE技术内容

JavaEE技术体系结构可分为：表示层技术、中间层技术、数据层技术。JavaEE还涉及到系统集成的一些技术。

1、表示层技术

        HTML

        JavaScript

        Ajax

Ajax是几种技术的整合。Ajax的主要功能是异步地向服务器端发送请求，处理数据或者根据返回的数据重新显示页面。

2、中间层技术

       JSP

       显示动态内容的服务器网页。

       Servlet

       接收客户端请求，并做出响应的Java程序。

       Servlet是中间层技术的重要组成部分，它控制着其他的组件。

       JSTL

       辅助JSP显示动态内容的标准标签库。

       JavaBean

       JavaEE的模型组件。

       中间层的框架技术

       Struts框架：Struts主要是扩展了Servlet。

3、数据层技术

       JDBC技术

       JDBC（Java Database Connectivity）：Java数据库连接。

       使用JDBC操作数据库中的表和数据。

       数据层框架技术

       数据层框架，Hibernate提供了以对象的形式操作关系型数据库数据的功能。

4、系统集成技术

在一些大型项目中，整个应用程序被分成若干个模块，由不同的开发团队同步进行开发，有可能他们使用的技术也是各不相同的，在每个功能模块开发结束后，所要做的事情就是把这些零散的模块，整合成一个整体，这才是我们也是客户所要的东西，这时我们就要用到有关系统集成的技术（Web Service）。

JAX-WS技术

JAX-WS（Java API for XML Web Service）是JavaEE平台的重要组成部分。

JAX-WS简化了使用Java技术开发Web服务的工作。

JNDI

JNDI（Java Naming and Directory Interface，Java命名和目录接口）是一组在Java应用中访问命名和目录服务的API，命名服务把对象和名称联系在一起，并且可以通过名称找到相应的对象。

### HTTP（HyperText Transfer Protocol，超文本传输协议）协议

HTTP是一个无状态协议，它基于客户端/服务器模型。HTTP客户端（如浏览器）需要与服务器建立一个连接，并将一则请求消息通过连接发送到HTTP服务器，以请求相应的资源。然后服务器返回带有请求资源的响应消息。当响应结束后，就断开连接，彼此并不保存连接信息。所以HTTP是无状态协议。

HTTP使用端口发送和接收消息。端口是协议发送和接收数据的信道或机制。80端口是为HTTP开发的，主要用于在WWW（World Wide Web，万维网）服务上传输信息。浏览网页服务默认的端口号是80。

### 使用MyEclipse提供的数据监控工具TCP/IP Monitor来进行数据监控。

TCP/IP Monitor（TCP/IP 监控器）的配置步骤：

Window（窗口）--->Preferences（首选项）--->Run Debug（运行/调式）--->TCP/IP Monitor（TCP/IP 监视器）

注意：如果没有找到，则按以下路径查找：

Window（窗口）--->Preferences（首选项）--->MyEclipse ---> Internet Tools（因特网工具）---> TCP/IP Monitor（TCP/IP 监视器）

然后选中Show the TCP/IP Monitor view there is activity（显示TCP/IP监视器视图，用来查看什么时候有请求和响应）。

单击“Add”按钮，弹出New Monitor窗口。

Local monitoring Port：填写8088。本机监控端口，也就是Monitor实际监听的端口。

Host name：填写localhost或127.0.0.1。欲监控的主机名称，监听对象的设置。

Port：填写8080。欲监控的端口号，你想要监听的端口号，你的请求发送到Local monitoring Port后会转到这个端口。

Type：选择HTTP。要监控的类型。

Timeout：0 。

单击确定按钮，然后单击Start（启动）按钮启动监视器。

然后启动tomcat，在地址栏中输入：<http://localhost:8088/>项目名（或上下文路径）。在MyEclipse中会出现“TCP/IP Monitor”窗口，可以看到监听的数据了。

可以单击窗口右上角的ValidateWS-I Message Log File图标（图标上有个勾）添加记录文件。

下面对某一请求示例进行分析：

客户端发送的消息为文本流，由以下内容组成：

请求行：包括方法、统一资源标识符（URL）和HTTP协议版本。

头信息：典型的请求消息类似于以下内容。

              GET /monitor/test.jsp?username=xinxin&userpassword=1234 HTTP/1.1

              Host: localhost:8080

              User-Agent: Mozilla/5.0(Windows;U;Window NT 5.1;zh-CN;)

              Accept: text/xml,application/xml, text/html; text/plain;

第一行使用GET方法提交请求数据，并指定正在使用的是HTTP协议1.1版本。后续行包括头信息，User-Agent头指示客户端使用的浏览器类型是Mozilla，Accept头信息提供有关客户端可以接受的MIME类型列表。在头信息之后，客户端发送一个空白行，指示请求消息结束。请求消息由服务器进一步处理，并且生成相应的响应消息。

       下面是响应示例：

       HTTP/1.1 200 OK

       Server: JavaWebServer

       Last-modified: Tuesday, 21-Aug-07 11:12:10 GMT

       Content-length: 100

       Content-type: text/html

第一行为状态行。状态行中的状态码200指示已成功处理请求，因此描述为“OK”，404是找不到被请求的网页时遇到的状态码。其他行是头信息，Server头指定服务器上使用的软件，Last-modified头指示被请求文件的最后修改日期，Content-length头指定文件的大小（以字节表示），Content-type头指定文档的MIME类型。例如，text/plain指示未格式化的文本文档。text/html指示格式化的Web文档。

### get和post发送方式

post：post方式会将表单的内容通过http发送，在地址栏中看不到表单的提交信息，比较安全，而且使用post方式发送信息没有字符长度的限制。

get方式：表单内容经过编码之后，通过URL发送，可以在地址栏中看到表单信息，不安全。使用get方式发送信息时有255个字符的限制。get请求的页面可以设置为书签或使用邮件发送，而post不可以。

## Java注解

注解：Java代码提供元数据，描述数据的数据(标签)；

2、基本内置注解：

(1)@Override:对覆盖超类中方法的方法进行标记，如果被标记的方法并没有实际覆盖超类中的方法，则编译器会发出错误警告。

(2)@Deprecated:过时代码注解，若使用，则警告；

(3)@SuppressWarnings:去处警告,可传参；

3、元注解：注解到注解上的注解,主要是应用在注解声明；

(1)@Retention:保留期，注解存活时间；

- RetentionPolicy.SOURCE 注解只在源码阶段保留，在编译器进行编译时它将被丢弃忽视。

- RetentionPolicy.CLASS 注解只被保留到编译进行的时候，它并不会被加载到 JVM 中。

- RetentionPolicy.RUNTIME 注解可以保留到程序运行的时候，它会被加载进入到 JVM 中，所以在程序运行时可以获取到它们。

(2)@Documented 将注解中的元素包含到 Javadoc 中去;

(3)@Target 指定应用场景

参数为枚举ElementType。若应用错误，编译报错；

(4)@Inherited 继承 ，若超类的注解声明包含，则子类默认继承超类的的该注解；

(5)@Repeatable 可重复注解。jdk1.8开始使用 其余1.5后

4、注解的属性：

成员变量。只有变量，没有方法。通过实例来说明。

5、java预置注解

(1)@Deprecated 过时代码(方法、类)；

(2)@Override

(3)@SuppressWarnings

(4)@SafeVarargs

(5)@FunctionalInterface 函数式接口，jdk1.8后加入；

6、lambda表达式

通过表达式来代替功能接口。和方法一样,它提供了一个正常的参数列表和一个使用这些参数的主体(body,可以是一个表达式或一个代码块。

语法：

// 1. 不需要参数,返回值为 5

() -> 5

// 2. 接收一个参数(数字类型),返回其2倍的值

x -> 2 \* x

// 3. 接受2个参数(数字),并返回他们的差值

(x, y) -> x – y

// 4. 接收2个int型整数,返回他们的和

(int x, int y) -> x + y

// 5. 接受一个 string 对象,并在控制台打印,不返回任何值(看起来像是返回void)

(String s) -> System.out.print(s)

具体例子，详看：LambdasTest.java

7、函数式接口

Java 8对一类特殊类型的接口的称呼。 这类接口只定义了唯一的抽象方法的接口（除了隐含的Object对象的公共方法）。

用例详见FunctionInterfaceTest.java

java.lang.Runnable,

java.awt.event.ActionListener,

java.util.Comparator,

java.util.concurrent.Callable

## 集合(Collection)

集合类和接口层次结构，如图所示:



集合的定义: 一组[类](https://en.wikipedia.org/wiki/Class_(computer_science))和[接口](https://en.wikipedia.org/wiki/Interface_(java))实现普遍可重复使用的收集[的数据结构](https://en.wikipedia.org/wiki/Data_structure)。

# Web应用

在不使用IDE情况下构建简单的web应用，首先需要了解web应用的文件结构

# JAVA远程通讯

### 远程方法调用(RMI，英文：Remote Method Invocation)

什么是RMI？

维基百科：一种用于实现[远程过程调用](https://zh.wikipedia.org/wiki/%E8%BF%9C%E7%A8%8B%E8%BF%87%E7%A8%8B%E8%B0%83%E7%94%A8)的[应用程序编程接口](https://zh.wikipedia.org/wiki/%E5%BA%94%E7%94%A8%E7%A8%8B%E5%BA%8F%E7%BC%96%E7%A8%8B%E6%8E%A5%E5%8F%A3)。它使客户机上运行的程序可以调用远程服务器上的对象。

(1)序列化：把对象转换为字节序列的过程称为对象的序列化。

(2)反序列化：把字节序列恢复为对象的过程称为对象的反序列化。

对象序列化用途:

1） 把对象的字节序列永久地保存到硬盘上，通常存放在一个文件中；

　　2） 在网络上传送对象的字节序列。

RMI远程调用步骤：

1)，客户对象调用客户端辅助对象（stub对象）上的方法；

2)，客户端辅助对象打包调用信息（变量，方法名），通过网络发送给服务端辅助对象（skeleton对象）；

3)，服务端辅助对象将客户端辅助对象发送来的信息解包，找出真正被调用的方法以及该方法所在对象；

4)，调用真正服务对象上的真正方法，并将结果返回给服务端辅助对象；

5)，服务端辅助对象将结果打包，网络发送给客户端辅助对象；

6)，客户端辅助对象将返回值解包，返回给客户对象；

7)，客户对象获得返回值；

共同部分：

Person.java:

**import** java.io.Serializable;  
**public class** Person **implements** Serializable {  
 */\*\*  
 \* 序列化ID  
 \*/* **private static final long *serialVersionUID*** = -3245478690496182643L;  
 **public** String **name**;  
 **public int age**;  
 **public** Person(String name, **int** age) {  
 **this**.**name** = name;  
 **this**.**age** = age;  
 }  
 **public** String getName() {  
 **return name**;  
 }  
 **public void** setName(String name) {  
 **this**.**name** = name;  
 }  
 **public int** getAge() {  
 **return age**;  
 }  
 **public void** setAge(**int** age) {  
 **this**.**age** = age;  
 }  
}

ImyRemote.java  
**import** java.rmi.Remote;  
**import** java.rmi.RemoteException;**public interface** IMyRemote **extends** Remote {  
 String sayHello() **throws** RemoteException;  
 Person getPerson() **throws** RemoteException;  
}

Server实现：

Server.java:

**import** java.net.MalformedURLException;  
**import** java.rmi.Naming;  
**import** java.rmi.RemoteException;  
  
**public class** Server {  
 **public static void** main(String[] args) {  
 **try**{  
*// LocateRegistry.createRegistry(1099);* IMyRemote iRemote = **new** MyRemoteImpl();  
 Naming.*rebind*(**"rmi://198.181.47.219:1099/remoteServer"**, iRemote);  
 System.***out***.println(**"绑定成功"**);  
 }**catch** (RemoteException e){  
 System.***out***.println(**"RemoteException"**);  
 e.printStackTrace();  
 } **catch** (MalformedURLException e) {  
 System.***out***.println(**"MalformedURLException"**);  
 e.printStackTrace();  
 }  
 }  
}

MyRemoteImpl.java  
**import** java.rmi.RemoteException;  
**import** java.rmi.server.UnicastRemoteObject;  
  
**public class** MyRemoteImpl **extends** UnicastRemoteObject **implements** IMyRemote {  
 **public** MyRemoteImpl() **throws** RemoteException {  
 }  
 @Override  
 **public** String sayHello() {  
 **return "server say,hello world"**;  
 }  
  
 @Override  
 **public** Person getPerson() **throws** RemoteException {  
 **return new** Person(**"张三"**,18);  
 }  
  
}

Cliet实现：

**import** java.net.MalformedURLException;  
**import** java.rmi.Naming;  
**import** java.rmi.NotBoundException;  
**import** java.rmi.RemoteException;  
  
**public class** Client {  
 **public static void** main(String[] args) **throws** RemoteException, NotBoundException, MalformedURLException {  
 IMyRemote client = (IMyRemote) Naming.*lookup*(**"rmi://198.181.47.219:1099/remoteServer"**);  
 System.***out***.println(client.sayHello());  
 Person p = client.getPerson();  
 System.***out***.println(**"name:"** + p.getName() + **"age:"** + p.getAge());  
 }  
}

注：

1. 启动远程服务前，需要先启动rmiregistry(Naming.rebind注册服务，rmiregistry必须运行)；
2. 传输的对象的类型必须成为可序列化类型(编译器无法发现)；

### 2.远程过程调用(RPC，英文：Remote Procedure Call)

RPC支持多语言，而RMI只支持JAVA写的应用程序。RMI运行方法返回JAVA对象及基本数据类型。而RPC不支持对象概念，传送到RPC服务的消息由外部数据表示语言标志(XDR)。

注：RMI和RPC没有可比性，RMI是Java分布式框架(基于TCP/IP)。RPC是网络协议

例子略

### 3.Java消息服务(JMS，英文：Java Message Service)

(1)JMS概念

[Java](https://zh.wikipedia.org/wiki/Java)平台中关于[面向消息中间件](https://zh.wikipedia.org/w/index.php?title=%E9%9D%A2%E5%90%91%E6%B6%88%E6%81%AF%E4%B8%AD%E9%97%B4%E4%BB%B6&action=edit&redlink=1)（MOM）的[API](https://zh.wikipedia.org/wiki/API)，用于在两个应用程序之间，或分布式系统中发送消息，进行异步通信。

JMS包含两种消息模式，点对点和发布者/订阅者。

JMS支持面向事件的方法接收消息(事件驱动的程序设计)。

1. JMS元素

JMS提供者：连接面向消息中间件的，JMS接口的一个实现。提供者可以是Java平台的JMS实现，也可以是非Java平台的面向消息中间件的适配器。

JMS客户：生产或消费消息的基于Java的应用程序或对象。

JMS生产者：创建并发送消息的JMS客户。

JMS消费者：接收消息的JMS客户。

JMS消息：包括可以在JMS客户之间传递的数据的对象

JMS队列：一个容纳那些被发送的等待阅读的消息的区域。队列暗示，这些消息将按照顺序发送。一旦一个消息被阅读，该消息将被从队列中移走。

JMS主题：一种支持发送消息给多个订阅者的机制。

# 数据结构

## 排序

冒泡排序

选择排序

插入排序

归并排序

快速排序

随机快速排序(random quick sort)

计数排序

基数排序

# 前端相关

## JSP

全称JavaServer Pages，中文名叫java服务器页面。JSP技术是以Java语言作为脚本语言的，JSP网页为整个服务器端的Java库单元提供了一个接口来服务于HTTP的应用程序。

JSP本质是Servlet,当用户向指定Servlet发送请求时，Servlet利用输出流动态生成HTML页面，包括每一个静态的HTML标签和所有在HTML页面中出现的内容。

当服务器接收一个客户端请求时，需要做一下四件事情。

(1) 加载和实例化

如果Servlet容器还没实例化一个Servlet对象，此时容器装载和实例化一个 Servlet。创建出该 Servlet 类的一个实例。如果已经存在一个Servlet对象，此时不再创建新实例。

(2) 初始化

在产生 Servlet 实例后，容器负责调用该 Servlet 实例的 init() 方法，在处理用户请求之前，来做一些额外的初始化工作。  
(3) 处理请求

当 Servlet 容器接收到一个 Servlet 请求时，便运行与之对应的 Servlet 实例的 service() 方法，service() 方法根据用户的请求调用相对应的doGet或doPost 方法来处理用户请求。然后再进入对应的方法中调用逻辑层的方法，实现对客户的响应。  
(4) 销毁

当 Servlet 容器决定将一个 Servlet 从服务器中移除时 ( 如 Servlet 文件被更新 )，便调用该 Servlet 实例的 destroy() 方法，在销毁该 Servlet 实例之前，来做一些其他的工作。

其中，(1)(2)(4) 在 Servlet 的整个生命周期中只会被执行一次。

Servlet工作原理流程图：



## HTML

HTML（Hypertext Markup Language）文本标记语言，它是静态页面。

与JSP的区别：定义上HTML页面是静态页面可以直接运行，JSP页面是动态页它运行时需要转换成servlet。

### Ajax

Asynchronous JavaScript and XML(异步JavaScript和XML)

# JavaScript

## Worker

# 服务器技术

## WebSphere

## Tomcat

## 知识点

服务器集群

## 负载均衡

将负载（工作任务，访问请求）进行平衡、分摊到多个操作单元（服务器，组件）上进行执行。是解决高性能，单点故障（高可用），扩展性（水平伸缩）的终极解决方案。

负载平衡服务通常是由专用软件和硬件来完成.

(1)软件(Nginx反向代理)：

异步框架的 Web服务器，也可以用作反向代理，负载平衡器 和 HTTP缓存。

Nginx集中策略：

1)轮询，根据Nginx配置文件中的顺序，依次把客户端的Web请求分发到不同的后端服务端。

2)最少连接，当前谁连接最少，分发给谁。

3)IP地址哈希：确定相同IP请求可以转发给同一个后端节点处理，以方便session保持；

4)基于权重的负载均衡：配置Nginx把请求更多地分发到高配置的后端服务器上，把相对较少的请求分发到低配服务器。

Eg:

(2)硬件(F5):

## 代理服务器

客户机在发送请求时，不会直接发送给目的主机，而是先发送给代理服务器，代理服务接受客户机请求之后，再向主机发出，并接收目的主机返回的数据，存放在代理服务器的硬盘中，再发送给客户机。

使用代理服务器的用处:

(1)提高访问速度；

(2)防火墙作用；

(3)通过代理服务器访问不能访问的目标站点(翻墙)；

(一)正向代理

架设在客户机与目标主机之间，只用于代理内部网络对Internet的连接请求，客户机必须指定代理服务器,并将本来要直接发送到Web服务器上的http请求发送到代理服务器中。

(二)反向代理

服务器架设在服务器端，通过缓冲经常被请求的页面来缓解服务器的工作量，将客户机请求转发给内部网络上的目标服务器；并将从服务器上得到的结果返回给Internet上请求连接的客户端，此时代理服务器与目标主机一起对外表现为一个服务器。

可以防止外网对内网服务器的恶性攻击、缓存以减少服务器的压力和访问安全控制之外，还可以进行负载均衡，将用户请求分配给多个服务器。

# Linux知识

## Linux标准输入、标准输出、标准错误与重定向介绍

当Linux执行一个程序的时候，会自动打开三个流，标准输入(standard input)，标准输出(standard output)，标准错误(standard error)。比如说你打开命令行的时候，默认情况下，命令行的标准输入连接到键盘，标准输出和标准错误都连接到屏幕。

步骤一：屏幕敲入 $ ls

步骤二：键盘敲击的文本流("ls\n"，\n是回车时输入的字符，表示换行)命令行 (命令行实际上也是一个程序)。命令行随后调用/bin/ls得到结果("a.txt")，最后这个输出的文本流("a.txt")流到屏幕，显示出来

步骤三：假设说我们不想让文本流流到屏幕，而是流到另一个文件，我们可以采用重新定向(redirect)的机制。

(1)命令：$ls > a.txt

注：新建文件a.txt并将ls的命令行的标准输出指向这个文件。



(2)命令：$ls >> a.txt

注：若文件a.txt不存在，则创建并将ls的命令行的标准输出指向这个文件；若文件a.txt存在，则直接将ls的命令行的标准输出追加至该文件内容后面。



(3) 命令：$cat < a.txt (类似cat.txt)

注：将cat标准输入指向a.txt，文本会从文件流到cat，然后再输出到屏幕上。



(4)命令：$cat < a.txt > b.txt

注：a.txt的内容就复制到了b.txt

## 常用命令

### rm

1.rm -rf \* 删除[当前目录](https://www.baidu.com/s?wd=%E5%BD%93%E5%89%8D%E7%9B%AE%E5%BD%95&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YvnjDvujDvnWR1PjDkPHuW0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnHT1PWDkn1cY)下的所有文件,这个命令很危险，应避免使用。

所删除的文件，一般都不能恢复！

2.rm -f 其中的，f参数 （f --force ） 忽略不存在的文件，不显示任何信息

不会提示确认信息。

3.rm 命令删除一个目录中的一个或者多个文件或者目录，只用rm命令不会

删除目录，通常文件通常可以恢。

4.其他参数命令：

    -f, --force    忽略不存在的文件，从不给出提示。

    -i, --interactive 进行交互式删除

    -r, -R, --recursive   指示rm将参数中列出的全部目录和子目录均递归地删除。

    -v, --verbose    详细显示进行的步骤

       --help     显示此帮助信息并退出

       --version  输出版本信息并退出

友情提示：一般的删除操作用rm 即可 。

### mkdir和touch

新建目录，mkdir 路径加目录名称。

新建文件，touch 文件名

### tar

压缩解压缩命令

-c: 建立压缩档案  
-x：解压  
-t：查看内容  
-r：向压缩归档文件末尾追加文件  
-u：更新原压缩包中的文件

这五个是独立的命令，压缩解压都要用到其中一个，可以和别的命令连用但只能用其中一个。下面的参数是根据需要在压缩或解压档案时可选的。  
  
-z：有gzip属性的  
-j：有bz2属性的  
-Z：有compress属性的  
-v：显示所有过程  
-O：将文件解开到标准输出

下面的参数-f是必须的

-f: 使用档案名字，切记，这个参数是最后一个参数，后面只能接档案名。

**解压**

tar -xvf file.tar //解压 tar包

tar -xzvf file.tar.gz //解压tar.gz

tar -xjvf file.tar.bz2   //解压 tar.bz2

tar -xZvf file.tar.Z   //解压tar.Z

unrar e file.rar //解压rar

unzip file.zip //解压zip

**总结**

1、\*.tar 用 tar -xvf 解压

2、\*.gz 用 gzip -d或者gunzip 解压

3、\*.tar.gz和\*.tgz 用 tar -xzf 解压

4、\*.bz2 用 bzip2 -d或者用bunzip2 解压

5、\*.tar.bz2用tar -xjf 解压

6、\*.Z 用 uncompress 解压

7、\*.tar.Z 用tar -xZf 解压

8、\*.rar 用 unrar e解压

9、\*.zip 用 unzip 解压

### vim

vim分为三种模式，命令模式(首先进来)、编辑模式(可输入)、底线命令模式(命令模式下输入:即可进入)

1快速定位文件头及文件尾

底线命令模式 :1 定位至文件首 :$ 定位至文件尾

命令模式 ctrl+F 向下翻页 ctrl+B 向上翻页

2保存文件或退出vi

底线命令模式下操作:

|  |  |
| --- | --- |
| 命令 | 简单说明 |
| :w | 保存编辑后的文件内容，但不退出vim编辑器。这个命令的作用是把内存缓冲区中的数据写到启动vim时指定的文件中。 |
| :w! | 强制写文件，即强制覆盖原有文件。如果原有文件的访问权限不允许写入文件，例如，原有的文件为只读文件，则可使用这个命令强制写入。但是，这种命令用法仅当用户是文件的属主时才适用，而超级用户则不受此限制。 |
| :wq | 保存文件内容后退出vim编辑器。这个命令的作用是把内存缓冲区中的数据写到启动vim时指定的文件中，然后退出vim编辑器。另外一种替代的方法是用ZZ命令。 |
| :wq! | 强制保存文件内容后退出vim编辑器。这个命令的作用是把内存缓冲区中的数据强制写到启动vim时指定的文件中，然后退出vim编辑器。 |
| ZZ | 使用ZZ命令时，如果文件已经做过编辑处理，则把内存缓冲区中的数据写到启动vim时指定的文件中，然后退出vim编辑器。否则只是退出vim而已。注意，ZZ命令前面无需加冒号“：”，也无需按Enter键。 |
| :q | 在未做任何编辑处理而准备退出vim时，可以使用此命令。如果已做过编辑处理，则vim不允许用户使用“:q”命令退出，同时还会输出下列警告信息：  No write since last change (:quit! overrides) |
| :q! | 强制退出vim编辑器，放弃编辑处理的结果。如果确实不需要保存修改后的文件内容，可输入“:q!”命令，强行退出vim编辑器。 |
| :w filename | 把编辑处理后的结果写到指定的文件中保存 |
| :w! filename | 把编辑处理后的结果强制保存到指定的文件中，如果文件已经存在，则覆盖现有的文件。 |
| :wq! filename | 把编辑处理后的结果强制保存到指定的文件中，如果文件已经存在，则覆盖现有文件，并退出vim编辑器。 |

3查找内容

底线命令模式 : /string查找 注:找到后继续查找：n 向下继续搜索 N 反方向继续搜索

### mv、cp和rm

命令行下的移动、复制与删除：mv ,cp,rm,

(1)mv

语法：mv(选项)(参数)

选项：

--backup=<备份模式>：若需覆盖文件，则覆盖前先行备份；

-b：当文件存在时，覆盖前，为其创建一个备份；

-f：若目标文件或目录与现有的文件或目录重复，则直接覆盖现有的文件或目录；

-i：交互式操作，覆盖前先行询问用户，如果源文件与目标文件或目标目录中的文件同名，则询问用户是否覆盖目标文件。用户输入”y”，表示将覆盖目标文件；输入”n”，表示取消对源文件的移动。这样可以避免误将文件覆盖。

--strip-trailing-slashes：删除源文件中的斜杠“/”；

-S<后缀>：为备份文件指定后缀，而不使用默认的后缀；

--target-directory=<目录>：指定源文件要移动到目标目录；

-u：当源文件比目标文件新或者目标文件不存在时，才执行移动操作。

参数：

源文件：源文件列表。

目标文件：如果“目标文件”是文件名则在移动文件的同时，将其改名为“目标文件”；如果“目标文件”是目录名则将源文件移动到“目标文件”下。

eg:

mv -fi download test

(2)cp

语法：cp(选项)(参数)

选项：

-a:相当于-pdr的意思，请参考后文  
         -d:若源文件为链接文件的属性，则复制链接文件属性而非文件本身  
         -f:强制的意思（force）,也就是说覆盖掉已存在的文件

参数：

源文件：源文件列表。

目标文件：如果“目标文件”是文件名则在移动文件的同时，将其改名为“目标文件”；如果“目标文件”是目录名则将源文件移动到“目标文件”下。

eg:

(3)rm

语法： rm (选项) (参数)

选项：

   -r:递归操作  
         -f:强制(force)  
         -i:删除时给出提示信息

参数：

文件或者目录

eg:

### sudo

### find和grep

查找命令, 两者的区别：

(1)find命令是根据文件的属性进行查找如文件名，文件大小，所有者，所属组，是否为空，访问时间，修改时间等

(2)grep是根据文件的内容进行查找，会对文件的每一行按照给定的模式(patter)进行匹配查找。

一find命令

格式 find  (path) (expression)

1按照文件名查找

(1)find / -name httpd.conf　#在根目录下查找文件httpd.conf，表示在整个硬盘查找  
　　(2)find /etc -name httpd.conf　#在/etc目录下文件httpd.conf  
　　(3)find /etc -name '\*srm\*'　#使用通配符\*(0或者任意多个)。表示在/etc目录下查找文件名中含有字符串‘srm’的文件  
　　(4)find . -name 'srm\*' #表示当前目录下查找文件名开头是字符串‘srm’的文件

2按照文件特征查找

(1)find / -amin -10 　　# 查找在系统中最后10分钟访问的文件(access time)  
　　　(2)find / -atime -2　　 # 查找在系统中最后48小时访问的文件  
　　　(3)find / -empty 　　# 查找在系统中为空的文件或者文件夹  
　　　(4)find / -group cat 　　# 查找在系统中属于 group为cat的文件  
　　　(5)find / -mmin -5 　　# 查找在系统中最后5分钟里修改过的文件(modify time)  
　　　(6)find / -mtime -1 　　#查找在系统中最后24小时里修改过的文件  
　　　(7)find / -user fred 　　#查找在系统中属于fred这个用户的文件  
　　　(8)find / -size +10000c　　#查找出大于10000000字节的文件(c:字节，w:双字，k:KB，M:MB，G:GB)  
　　　(9)find / -size -1000k 　　#查找出小于1000KB的文件

二 grep命令

格式：grep expression

(1)grep 'test' d\*　　#显示所有以d开头的文件中包含 test的行  
(2)grep ‘test’ aa bb cc 　　 #显示在aa，bb，cc文件中包含test的行

(3)grep ‘[a-z]\{5\}’ aa 　　#显示所有包含每行字符串至少有5个连续小写字符的字符串的行

(4)grep magic /usr/src　　#显示/usr/src目录下的文件(不含子目录)包含magic的行

(5)grep -r magic /usr/src　　#显示/usr/src目录下的文件(包含子目录)包含magic的行

(6)grep -w pattern files ：只匹配整个单词，而不是字符串的一部分(如匹配’magic’，而不是’magical’)，

### ls

显示目录列表。格式：ls (选项) (参数)

选项:

-a：显示所有档案及目录（ls内定将档案名或目录名称为“.”的视为影藏，不会列出）；

-A：显示除影藏文件“.”和“..”以外的所有文件列表；

-C：多列显示输出结果。这是默认选项；

-l：与“-C”选项功能相反，所有输出信息用单列格式输出，不输出为多列；

-F：在每个输出项后追加文件的类型标识符，具体含义：“\*”表示具有可执行权限的普通文件，“/”表示目录，“@”表示符号链接，“|”表示命令管道FIFO，“=”表示sockets套接字。当文件为普通文件时，不输出任何标识符；

-b：将文件中的不可输出的字符以反斜线“”加字符编码的方式输出；

-c：与“-lt”选项连用时，按照文件状态时间排序输出目录内容，排序的依据是文件的索引节点中的ctime字段。与“-l”选项连用时，则排序的一句是文件的状态改变时间；

-d：仅显示目录名，而不显示目录下的内容列表。显示符号链接文件本身，而不显示其所指向的目录列表；

-f：此参数的效果和同时指定“aU”参数相同，并关闭“lst”参数的效果；

-i：显示文件索引节点号（inode）。一个索引节点代表一个文件；

参数：

目录：指定要显示列表的目录，也可以是具体的文件。

eg：

### stat

查看文件状态,包含文件信息 访问时间，修改时间。

eg：

stat debug.log 查看debug.log的文件状态。

### free和top

查看内存使用情况。

### ps -ef |grep和kill

ps -ef |grep redis 查找redis的进程；

kill -9 PID 杀死进程；

### iptables

### cat

文本输出命令，通常是用于观看某个文件的内容的.

cat主要有三大功能：

命令格式：

cat [-AbeEnstTuv] [--help] [--version] fileName

常用命令：

(1)一次显示整个文件;

$ cat   filename

(2)从键盘创建一个文件;

$ cat  >  filename

$ cat >> filename

(3)将几个文件合并为一个文件。

$cat   file1   file2  > file



(4) 以EOF输入字符为标准输入结束：

$cat<<EOF



(5) 以EOF输入字符为标准输入,并追加至filename文件后面

$cat>filename<<EOF



cat << EOF > test.sh



### lsof

(1)lsof –i 显示符合条件的进程情况



(2)lsof –i:端口号 查询该端口占用情况



### **netstat**

打印Linux中网络系统的状态信息。

netstat -a #列出所有端口

netstat -[at](http://man.linuxde.net/at) #列出所有tcp端口

netstat -au #列出所有udp端口

netstat -l #只显示监听端口

netstat -lt #只列出所有监听 tcp 端口

netstat -lu #只列出所有监听 udp 端口

netstat -lx #只列出所有监听 UNIX 端口

# 分布式知识

负载均衡

反向代理

静态内容缓存

访问控制

# 网络编程(计算机网络)

## TCP/IP协议

Transmission Control Protocol/Internet Protocol的简写，中译名为传输控制协议/[因特网](https://baike.baidu.com/item/%E5%9B%A0%E7%89%B9%E7%BD%91)互联协议，又名网络[通讯协议](https://baike.baidu.com/item/%E9%80%9A%E8%AE%AF%E5%8D%8F%E8%AE%AE)，是Internet最基本的协议、Internet国际[互联网](https://baike.baidu.com/item/%E4%BA%92%E8%81%94%E7%BD%91)络的基础，由[网络层](https://baike.baidu.com/item/%E7%BD%91%E7%BB%9C%E5%B1%82)的IP协议和[传输层](https://baike.baidu.com/item/%E4%BC%A0%E8%BE%93%E5%B1%82)的TCP协议组成。TCP/IP 定义了电子设备如何连入[因特网](https://baike.baidu.com/item/%E5%9B%A0%E7%89%B9%E7%BD%91)，以及数据如何在它们之间传输的标准。协议采用了4层的层级结构，每一层都呼叫它的下一层所提供的协议来完成自己的需求。

TCP/IP参考模型分为四个层次：应用层、传输层、网络互连层和主机到网络层。





## OSI模型

开放系统互连参考模型 (Open System Interconnect 简称OSI）是国际标准化组织(ISO)和国际电报电话咨询委员会(CCITT)联合制定的开放系统互连参考模型，为开放式互连信息系统提供了一种功能结构的框架。它从低到高分别是：[物理层](https://baike.baidu.com/item/%E7%89%A9%E7%90%86%E5%B1%82)、[数据链路层](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E9%93%BE%E8%B7%AF%E5%B1%82)、[网络层](https://baike.baidu.com/item/%E7%BD%91%E7%BB%9C%E5%B1%82)、[传输层](https://baike.baidu.com/item/%E4%BC%A0%E8%BE%93%E5%B1%82)、会话层、[表示层](https://baike.baidu.com/item/%E8%A1%A8%E7%A4%BA%E5%B1%82)和[应用层](https://baike.baidu.com/item/%E5%BA%94%E7%94%A8%E5%B1%82)。

## TCP/IP与OSI比较



## socket编程

套接字（socket）是通信的基石，是支持TCP/IP协议的网络通信的基本操作单元。

### Socket编程模型

# 前沿技术

## 区块链

### 比特币

## 人工智能

# 压力测试

# 设计模式

# 数据库技术

## 存储过程

存储过程 (Stored Procedure) 是在大型数据库系统中 , 一组为了完成特定功能的 SQL 语句集 , 存储在数据库中 , 经过第一次编译后再次调用不需要再次编译 , 用户通过指定存储过程的名字并给出参数 (如果该存储过程带有参数) 来执行它 , 存储过程是数据库中的一个重要对象 ; 存储过程中可以包含 逻辑控制语句 和 数据操纵语句 , 它可以接受参数 , 输出参数 , 返回单个或多个结果集以及返回值 ;

基本语法：

1.CREATE OR REPLACE PROCEDURE XX(存储过程名)

2.IS

3.BEGIN

4.NULL;

5.END;

注：行1是一个SQL语句通知数据库创建一个叫做XX的存储过程，若存在则覆盖；

行2 IS关键词表明后面将跟随一个PL/SQL体；

行3 BEGIN关键词表明PL/SQL体的开始；

行4 NULL PL/SQL语句表明什么事都不做，这句不能删去，因为PL/SQL体中至少需要有一句;

行5 END关键词表明PL/SQL体的结束；

例如创建过程：

create or replace procedure 存储过程名（param1 in type，param2 out type）   
as   
变量1 类型（值范围）; --vs\_msg VARCHAR2(4000);   
变量2 类型（值范围）;

Begin

If(判断条件) then

Select count(\*) into 变量2 from 表A where 列名= param1;

Dbms\_output.Put\_line(‘打印信息’);

else if(判断条件) then

Dbms\_output.Put\_line(‘打印信息’);

else

Raise 异常名(NO\_DATA\_FOUND);

End if;

Exception

When others then

Rollback;

End;

注意事项：   
1， 存储过程参数不带取值范围，in表示传入，out表示输出   
类型可以使用任意Oracle中的合法类型。   
2， 变量带取值范围，后面接分号   
3， 在判断语句前最好先用count（\*）函数判断是否存在该条操作记录   
4， 用select 。。。into。。。给变量赋值   
5， 在代码中抛异常用 raise+异常名

# 设计模式

### 创建模式-单例模式

(1)普通单例类创建：

public class Singleton

{

private static Singleton instance = new Singleton();

private Singleton(){

…

}

public static Singleton getInstance(){

return instance;

}

}

缺点：实例已经创建，但是消耗资源。

(2)惰性加载创建：

public class Singleton{

private static Singleton instance = null;

private Singleton(){

…

}

public static Singleton getInstance(){

if (instance == null)

instance = new Singleton();

return instance;

}

}

缺点：多线程避免不了同时new两个实例；

(3)Class锁机制

public class Singleton{

private static Singleton instance = null;

private Singleton(){

…

}

public static synchronized Singleton getInstance(){

if (instance == null)

instance = new Singleton();

return instance;

}

}

缺点:影响性能每次调用getInstance都必须获得Singleton的锁，但

(4)double-checked locking

public class Singleton{

private static Singleton instance = null;

private Singleton(){

…

}

public static Singleton getInstance(){

if (instance == null)

synchronized(instance){

if(instance == null)

instance = new Singleton();

}

return instance;

}

}

工作原理：

首先当一个线程发出请求后，会先检查instance是否为null，如果不是则直接返回其内容。

当并发访问时，必会有一线程先执行创建，后续线程非空不会创建。

缺点：

A、B线程先后执行，JVM给A分配空间但未new，直接返回。B线程进入会出错；

(5)完美解决：

public class Singleton{

private Singleton(){

…

}

private static class SingletonContainer{

private static Singleton instance = new Singleton();

}

public static Singleton getInstance(){

return SingletonContainer.instance;

}

}

原理：(内部类只在需要的时候才会被类加载器加载，实现了懒加载)JVM内部的机制能够保证当一个类被加载的时候，这个类的加载过程是线程互斥的。这样当我们第一次调用getInstance的时候，JVM能够帮我们保证instance只被创建一次，并且会保证把赋值给instance的内存初始化完毕，这样我们就不用担心3.2中的问题。此外该方法也只会在第一次调用的时候使用互斥机制。最后instance是在第一次加载SingletonContainer类时被创建的，而SingletonContainer类则在调用getInstance方法的时候才会被加载，因此也实现了惰性加载。

### 代理模式

远程代理实现原理(RMI，具体见RMI)。