Web 的发展历程

一、Socket 编程

1、回顾

回顾 Socket 编程，客户端请求服务器通信，直观了解 Web 编程。将所有的客户端都统一为浏览器，则由原来的 C/S 架构编程了解 B/S 架构。

Server.java

```
/**
 * 服务器端，接收客户端请求并给出简单的响应
 * @author Administrator
 */
publicclass Server {
	publicstaticvoid main(String[] args) throws IOException {
		//1、创建服务器，指定端口ServerSocket(int port)
		ServerSocket socket=new ServerSocket(8888);
		//2、接收客户端连接
		Socket client=socket.accept();
		System.out.println("********************");
		//获取数据的输入流
		InputStream is=client.getInputStream();
		//使用字符缓存流
		BufferedReader br=new BufferedReader(new InputStreamReader(is));
		String msg="";
		while((msg=br.readLine())!=null){
			System.out.println(msg);
		}
		br.close();
	}
}
```

Client.java

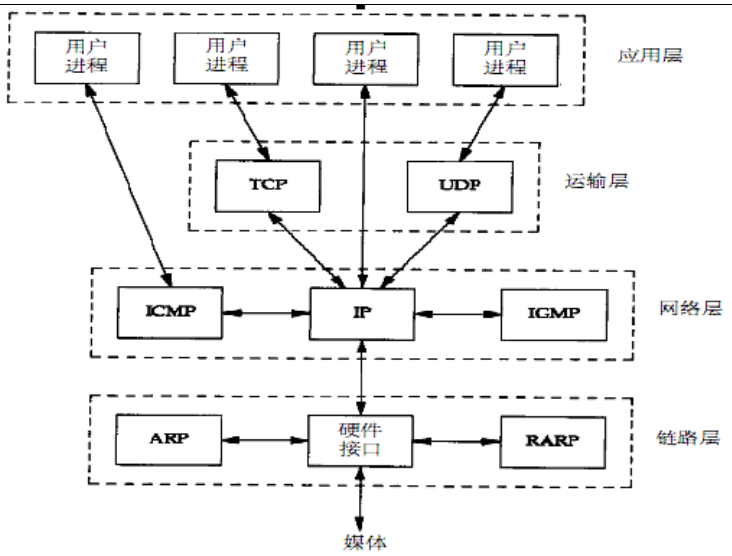
```
/**
 * 客户端：向服务器发送请求，并发送简单的消息
 * @author Administrator
```

```
 */
publicclass Client {
    publicstaticvoid main(String[] args) throws UnknownHostException,
IOException {
        //创建客户端 必须指定服务器+端口
        Socket client=new Socket("localhost",8888);
        //发送消息    请求资源
        //获取发送流
        OutputStream os=client.getOutputStream();
        BufferedWriter br=new BufferedWriter(new OutputStreamWriter(os));
        //写出消息，发送内容
        String msg="hello I need some source";
        br.write(msg);
        br.close();
    }
}
```

从上面的例子总结通信条件如下信息：
1、 需要有服务器端(server) ：等待被请求，需要暴露 ip 和 port
2、 需要有客户端(client)：主动发起请求 ，知晓服务端的 ip 和 port
3、 通信规则（协议）：TCP/IP 协议
解释：
ip 用于定位计算机
端口号(定位程序)
用于标识进程的逻辑地址,不同进程的标志
有效端口:0~65535,其中 0~1024 由系统使用或者保留端口,开发中建议使用 1024 以上的端口。

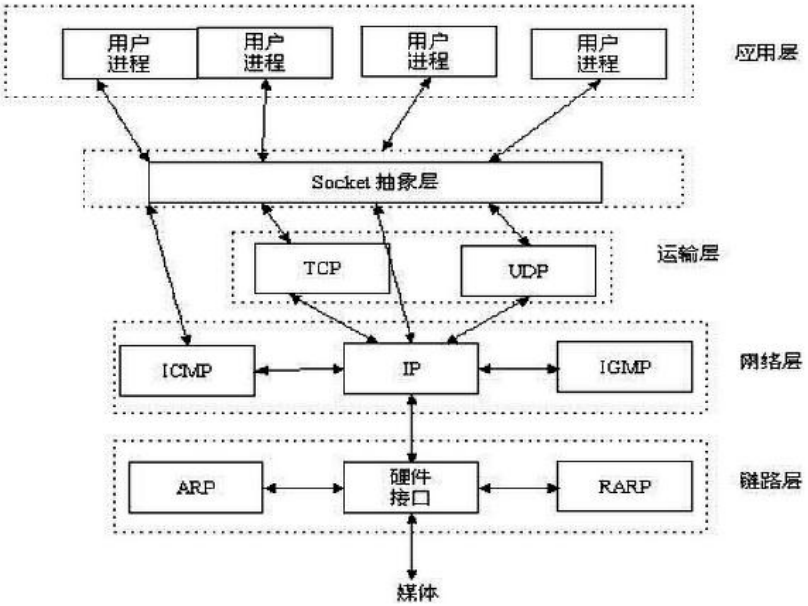
2、再说 Socket

使用 TCP/IP 协议的应用程序通常采用应用编程接口来实现网络程序之间的通信，就目前而言，几乎所有的应用程序都是采用 socket。
在计算机网络中我们就学过了 tcp/ip 协议族，其实使用 tcp/ip 协议族就能达到我们想要的通信效果。如下图

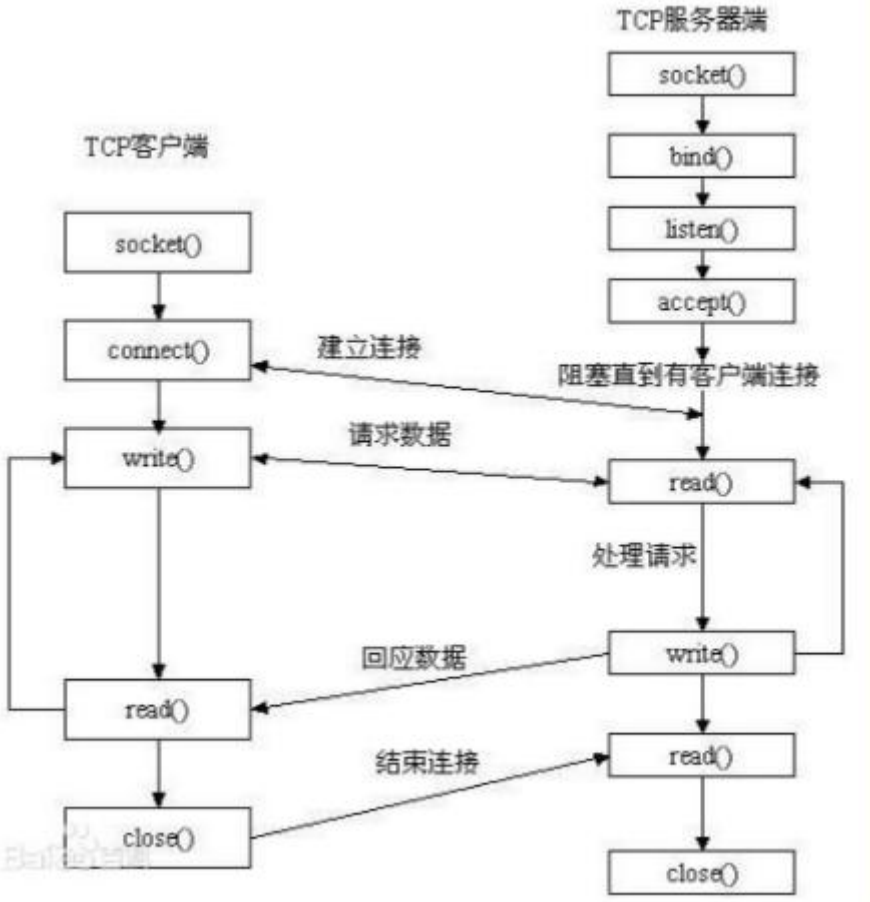


图一 各协议所处层次

当然，这样做固然是可以的，但是，当我们使用不同的协议进行通信时就得使用不同的接口，还得处理不同协议的各种细节，这就增加了开发的难度，软件也不易于扩展。于是 UNIX BSD 就发明了 socket 这种东西，socket 屏蔽了各个协议的通信细节，使得程序员无需关注协议本身，直接使用 socket 提供的接口来进行互联的不同主机间的进程的通信。这就好比操作系统给我们提供了使用底层硬件功能的系统调用，通过系统调用我们可以方便的使用磁盘（文件操作），使用内存，而无需自己去进行磁盘读写，内存管理。socket 其实也是一样的东西，就是提供了 tcp/ip 协议的抽象，对外提供了一套接口，同过这个接口就可以统一、方便的使用 tcp/ip 协议的功能了。



Socket(套接字)：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。它是网络通信过程中端点的抽象表示，包含进行网络通信必须的五种信息：**连接使用的协议**，本地主机的 IP 地址，本地进程的协议端口，远地主机的 IP 地址，远地进程的协议端口。



3、客户端不同的需求

Client.java

```
package com.shsxt.socket;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.InetSocketAddress;
import java.net.Socket;
import java.net.SocketAddress;

/**
 *
 * @author Lj
 *
 * 客户端
 */
public class Client {
	public static void main(String[] args) throws IOException {
```

```
            //  通过系统默认类型的 SocketImpl 创建未连接套接字
            Socket socket = new Socket();
            //此类实现 IP 套接字地址（IP 地址 + 端口号）。它还可以是一个对（主机名 + 端口号），在此情况下，将尝试解析主机名
            SocketAddress address = new InetSocketAddress("localhost",8898);
            // 将此套接字连接到服务器，并指定一个超时值。  或者不指定超时时间
            socket.connect(address, 1000);

            OutputStream os = socket.getOutputStream();
            os.write("time".getBytes());
            os.flush();
            socket.close();
      }
}
```

Server.java

```
package com.shsxt.socket;

import java.io.BufferedInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.net.ServerSocket;
import java.net.Socket;

/**
 *
 * @author Lj
 * 服务端
 * public class ServerSocketextends Object：此类实现服务器套接字。服务器套接字等待请求通过网络传入。它基于该请求执行某些操作，然后可能向请求者返回结果。
 */
public class Server {
      public static void main(String[] args) throws IOException {
            //创建绑定到特定端口的服务器套接字。
            ServerSocket server = new ServerSocket(8898);

            // Socket accept()   侦听并接受到此套接字的连接。
            Socket client = server.accept();
            System.out.println("接收到连接");

            InputStream is = client.getInputStream();
            BufferedInputStream bis = new BufferedInputStream(is);
            byte[] req = new byte[1024];
```

```
            // 接收客户端请求
            int len = bis.read(req);
            String reqStr = new String(req,0,len);
            System.out.println(reqStr);
            if(reqStr.equals("money")){
                System.out.println("here's the money");
            }else if(reqStr.equals("time")){
                System.out.println("you have so much time");
            }
            client.close();
            server.close();
        }
}
```

4、动态资源的请求

Client.java

```
package com.shsxt.socket;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.InetSocketAddress;
import java.net.Socket;
import java.net.SocketAddress;

/**
 * @authorLj
 * 客户端
 */
publicclass Client {
    publicstaticvoid main(String[] args) throws IOException {
        //  通过系统默认类型的 SocketImpl 创建未连接套接字
        Socket socket = new Socket();
        //此类实现 IP 套接字地址（IP 地址 + 端口号）。它还可以是一个对（主机名 + 端口号），在此情况下，
将尝试解析主机名
        SocketAddress address = new InetSocketAddress("localhost",8898);
        // 将此套接字连接到服务器，并指定一个超时值。或者不指定超时时间
        socket.connect(address, 1000);

        OutputStream os = socket.getOutputStream();
        os.write("money".getBytes());
```

```
            os.flush();
            // 接收响应，显示结果
            InputStream is = socket.getInputStream();
            byte[] result = newbyte[1024];
            intlen = is.read(result);
            String resultStr    = new String(result,0,len);
            System.out.println(resultStr);

            socket.close();

        }
}
```

Server.java

```
package com.shsxt.socket;

import java.io.BufferedInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.ServerSocket;
import java.net.Socket;

/**
 *
 * @author Lj  服务端  public class ServerSocketextends
 *              Object：此类实现服务器套接字。服务器套接字等待请求通过网络传入。它基于该请求执行某些操作，然后可能向请求者返回结果。
 */
public class Server {
        public static void main(String[] args) throws IOException {
            // 创建绑定到特定端口的服务器套接字。
            ServerSocket server = new ServerSocket(8898);

            // Socket accept()  侦听并接受到此套接字的连接。
            Socket client = server.accept();
            System.out.println("接收到连接");

            InputStream is = client.getInputStream();
            BufferedInputStream bis = new BufferedInputStream(is);
            byte[] req = new byte[1024];
            // 接收客户端请求
            int len = bis.read(req);
```

```
            String reqStr = new String(req, 0, len);
            System.out.println(reqStr);
            // 将接收到的请求封装成对象，传送给请求的类
            MyRequest request = new MyRequest();
            MyResponse response = new MyResponse();

            OutputStream os = client.getOutputStream();
            if (reqStr.equals("money")) {
                // 根据请求的信息，构造处理的类
                MyServlet s1 = new ServletMoney();
                s1.service(request, response);
                // 通过client的响应，将结果响应回客户端
                os.write("here's the money".getBytes());
                os.flush();

            } else if (reqStr.equals("time")) {
                // 根据请求的信息，构造处理的类
                MyServlet s2 = new ServletTime();
                s2.service(request, response);
                // 通过client的响应，将结果响应回客户端
                os.write("you have somuch time".getBytes());
                os.flush();
            }
            client.close();
            server.close();
        }
}

/*
 * 我是一个有要求的人，你请求的这个资源必须是满足我要求格式的类作用：防止混乱，方便调用
 * 这是我的标准
 */
interface MyServlet {
        void service(MyRequest req, MyResponse resp);
}

class ServletMoney implements MyServlet {

        /*
         * @see com.shsxt.socket.MyServlet#service(com.shsxt.socket.MyRequest)
         */
        @Override
        public void service(MyRequest req, MyResponse resp) {
            // 做出力所能及的处理
```

```
        }
}

class ServletTime implements MyServlet {

        /*
         * @see com.shsxt.socket.MyServlet#service(com.shsxt.socket.MyRequest)
         */
        @Override
        public void service(MyRequest req, MyResponse resp) {
                // 做出力所能及的处理
        }

}

/*
 * 请求信息都按规律封装在该对象
 *
 */
class MyRequest {

}

class MyResponse {

}
```

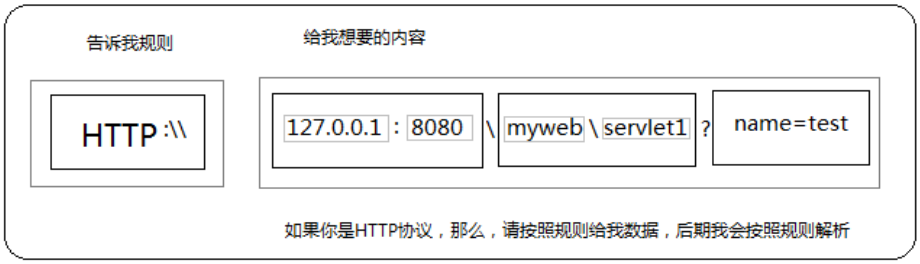
5、服务器的出现

当客户端请求的资源越来越丰富，需求越来越复杂，程序的核心就因该放在解决业务和计算响应数据上， 于是出现了服务器统一接收客户端处理并进行分发到不同的资源，由各个资源进行处理，最后结果交由服务器响应。

统一服务器端（总开关|总入口） -> Tomcat
统一客户端（所有请求不需要再单独写一套客户端） -> 浏览器

服务器和客户端遵循统一规则发送消息（请求和相应），方便获取数据和信息。 此规则即 HTTP 协议

分析规则，根据规则填充内容（书写规则，由浏览器规定）



浏览器是 HTTP 客户，向服务器发送请求，当中浏览器地址栏输入 URL 后回车或点击了一个超级链接时，浏览器就向服务器发送了 HTTP 请求，此请求被送往由 URL 指定的 IP 地址。

至此，服务器诞生了， HTTP 协议诞生了。

服务器：解决固定的获取数据和响应请求的功能

协议：同一规范请求格式和响应格式

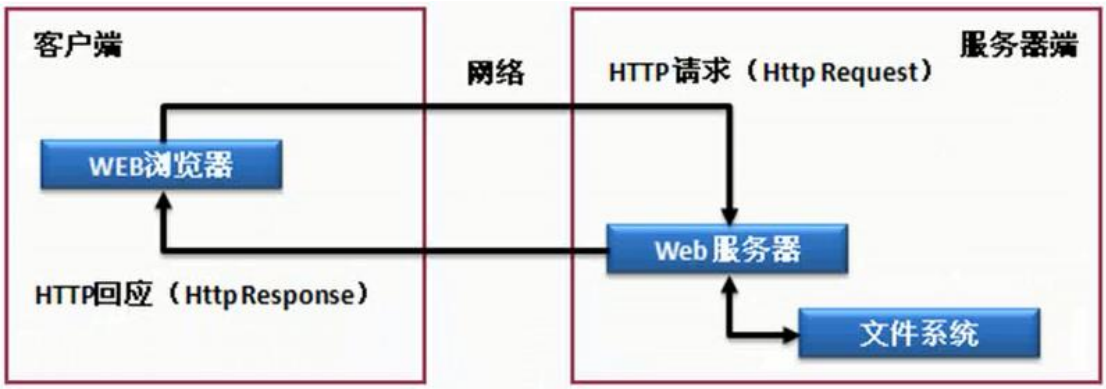
二、WEB 的发展历程

1、WEB 的发展历程

在最早，人们为了方便开展科学研究，设计出了 Internet 用于连接美国的少数几个顶尖研究机构，之后随着进一步的发展，人们开始用 HTTP 协议（**Hypertext Transfer Protocol，超文本传输协议**）进行超文本（hypertext）和超媒体（hypermedia）数据的传输，从而将一个个的网页展示在每个用户的浏览器上，今天的 WEB 已经从最早的静态 WEB 发展到了动态 WEB 阶段，随之而来的像网上银行、网络购物的兴起，更是将 WEB 带入了人们的生活和工作之中。

HTTP 协议是超文本传输协议，也是以后在开发中主要用到的协议

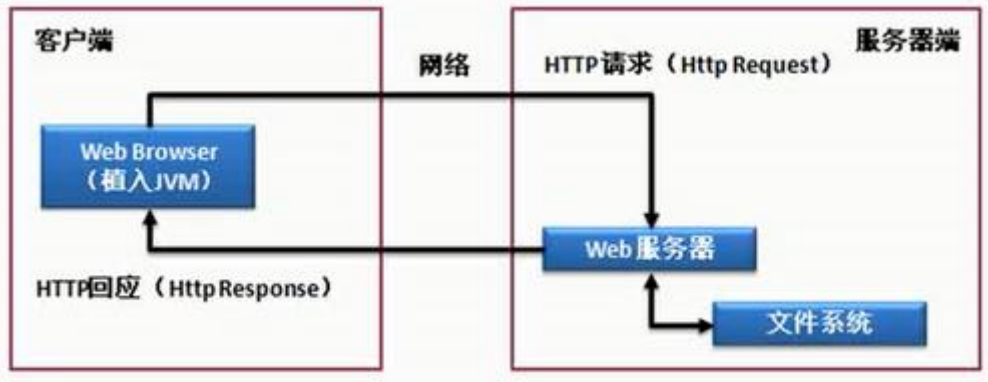
早期的静态 WEB



在 WEB 中分为两端：

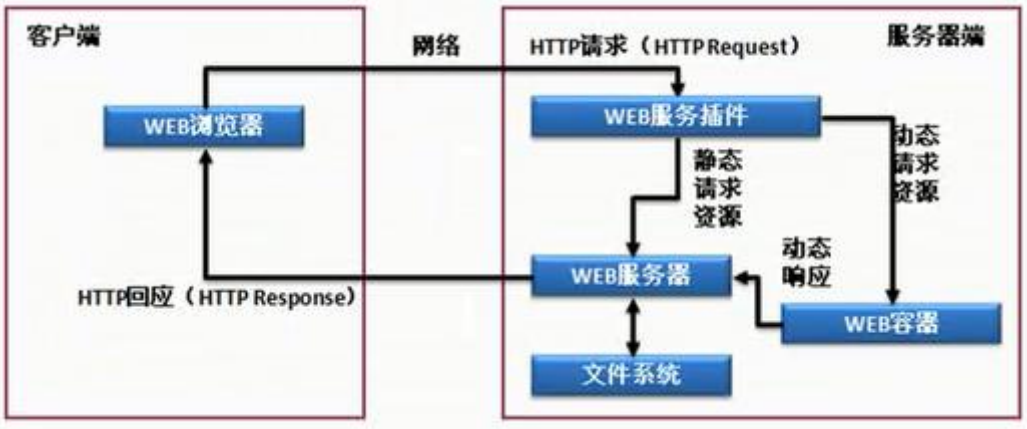
客户端：使用浏览器，向 WEB 服务器发出请求

服务器端：从 WEB 服务器端，根据用户的请求内容，找到文件，并通过 HTTP 协议发回给客户端，客户端再通过浏览器解释，这一切都是依靠 HTTP 协议完成的。早期的 WEB 不具备交互的能力，用户需要什么，服务器给什么；1995 年，出现了 Applet 程序



Applet 实际上是作为最早的客户端实现的 WEB 技术手段出现的此时也是静态 WEB，而静态 WEB 不能访问资源，例如：数据库、文件资源等等
这时候也只是简单的在客户端上发出了变化，但其他的服务器端并没有任何改变，所以这个时候相当于所有的动态效果，基本都是由客户端实现的。除了 Applet 之外，也可以通过 JavaScript 进行客户端动态效果的实现

动态 WEB



动态 WEB 最大的特点就是具备了**交互性**，而且在整个动态 WEB 之中，客户端现在所做的工作非常简单，就是一个普通的浏览器。

但是在服务器端却有了很大的改变，例如：在服务器端不再直接使用 WEB 服务器进行接收了，而是先通过了一个 WEB 服务插件，用于区分是动态请求还是静态请求。

若是静态请求：直接将内容交给 WEB 服务器，并且调用文件系统，和静态 WEB 一样
若是动态请求 ：则将进入到一个 WEB 容器，之后将开始进行代码的拼凑工作，动态 WEB 本身是没有固定代码的。但是不管是固定的代码还是拼凑的代码，现在基本上都会通过 WEB 服务器返回，返回给客户端，进行内容显示
动态 WEB 和静态 WEB 的最大区别：动态 WEB 可以进行数据库连接，而静态 WEB 无法连接数据库，现在的大部分应用程序都是围绕数据库进行的。

静态 WEB 基本上就是靠 HTML（网页）实现的。

动态 WEB 的实现方式很多：

CGI(Common Gateway Interface，公共网关接口)

PHP(Hypertext Preprocessor ，超文本预处理)

ASP(Active Server Page，动态服务页)

JSP(Java Server Page，Java 服务页)/Servlet(服务器端小程序)

资源简介(静态资源动态资源)

- 静态资源：浏览器可以直接打开的 如：html、css、javascript、图片、影音文件.... (浏览器直接可以读懂展示)
- 动态资源：一些资源浏览器不能够直接打开，但是经过(服务器)翻译之后浏览器能够打开的资源称之动态资源。如： jsp/servlet、php、ASP 等运行在服务器端的程序

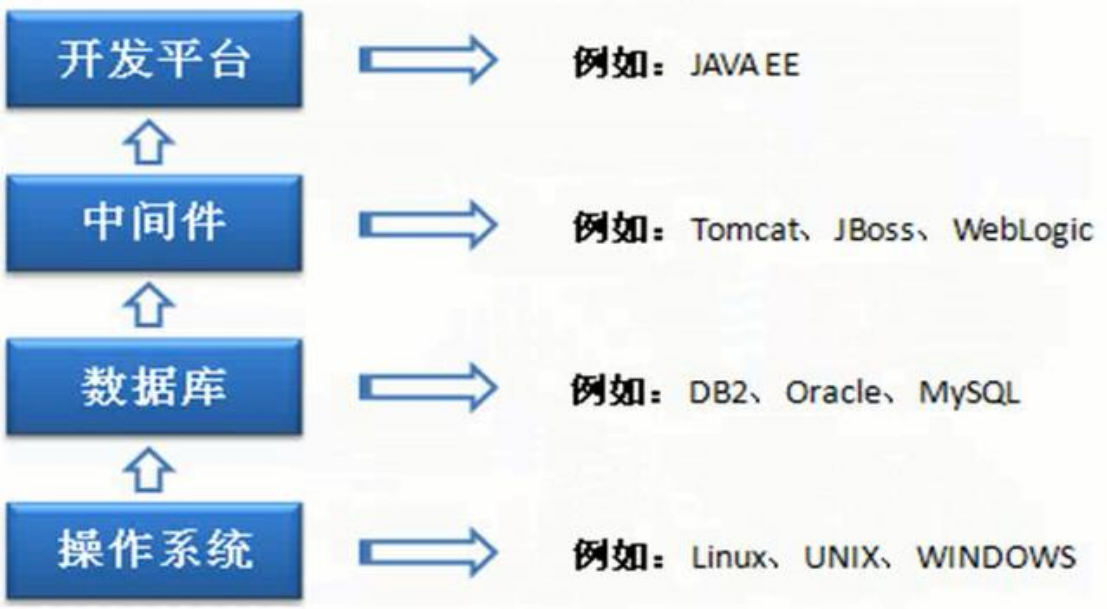
Web 资源的访问方式

- 浏览器

  格式：
  协议名：//域名(ip 地址):端口/url
  例如：http://www.baidu.com
  端口号：浏览器默认端口 80
  tomcat：默认端口 8080

2、企业开发简介

了解清楚企业开发中各个操作的形式：



JAVA 有三个分支：JAVA SE、JAVA ME、JAVA EE，其中 JAVA EE 是企业开发，主要是在 JAVA SE 基础之上建立起来的。JAVA SE 有的东西，JAVA EE 基本都有，但是 JAVA EE 有的东西，JAVA SE 可能就没有。

三、服务器

1、服务器

服务器是提供计算服务的设备。由于服务器需要响应服务请求，并进行处理，因此一般来说服务器应具备承担服务并且保障服务的能力。简单来说，服务器是提供某些服务的设备。

随着 Internet 的发展壮大，“主机/终端”或“客户机/服务器”的传统的应用系统模式已经不能适应新的环境,于是就产生了新的分布式应用系统,相应地,新的开发模式也应运而生，即所谓的“浏览器/服务器”结构、“瘦客户机”模式。应用服务器便是一种实现这种模式核心技术。

Web 应用程序驻留在应用服务器(Application Server)上。应用服务器为 Web 应用程序提供一种简单的和可管理的对系统资源的访问机制。它也提供低级的服务，如 HTTP 协议的实现和数据库连接管理。Servlet 容器仅仅是应用服务器的一部分。除了 Servlet 容器外，应用服务器还可能提供其他的 Java EE(Enterprise Edition)组件，如 EJB 容器，JNDI 服务器以及 JMS 服务器等。

市场上可以得到多种应用服务器，其中包括 Apache 的 Tomcat、IBM 的 websphere、Caucho Technology 的 Resin、Macromedia 的 JRun、NEC WebOTX Application Server、JBoss Application Server、BEA 的 WebLogic 等。其中有些如 NEC WebOTX Application Server、WebLogic、WebSphere 不仅仅是 Servlet 容器，它们也提供对 EJB(Enterprise JavaBeans)、JMS(Java Message Service)以及其他 Java EE 技术的支持。每种类型的应用服务器都有自己的优点、局限性和适用性。

2、Tomcat

Tomcat 是 Apache 软件基金会（Apache Software Foundation）的 Jakarta 项目中的一个核心项目，由 Apache、Sun 和其他一些公司及个人共同开发而成。由于有了 Sun 的参与和支持，最新的 Servlet 和 JSP 规范总是能在 Tomcat 中得到体现。因为 Tomcat 技术先进、性能稳定,而且免费,因而深受 Java 爱好者的喜爱并得到了部分软件开发商的认可，成为目前比较流行的**Web 应用服务器**。

Tomcat 服务器是一个免费的开放源代码的 Web 应用服务器，属于轻量级应用服务器，在中小型系统和并发访问用户不是很多的场合下被普遍使用，是开发和调试 JSP 程序的首选。对于一个初学者来说，可以这样认为，当在一台机器上配置好 Apache 服务器，可利用它响应 HTML（标准通用标记语言下的一个应用）页面的访问请求。实际上 Tomcat 部分是 Apache 服务器的扩展，但它是独立运行的，所以当你运行 tomcat 时，它实际上作为一个与 Apache 独立的进程单独运行的。

当配置正确时，Apache 为 HTML 页面服务，而 Tomcat 实际上是在运行 JSP 页面和 Servlet。另外，Tomcat 和 IIS 等 Web 服务器一样，具有处理 HTML 页面的功能，另外它还是一个 Servlet 和 JSP 容器，独立的 Servlet 容器是 Tomcat 的默认模式。不过，Tomcat 处理静态 HTML 的能力不如 Apache 服务器。目前 Tomcat 最新版本为**9.0**。