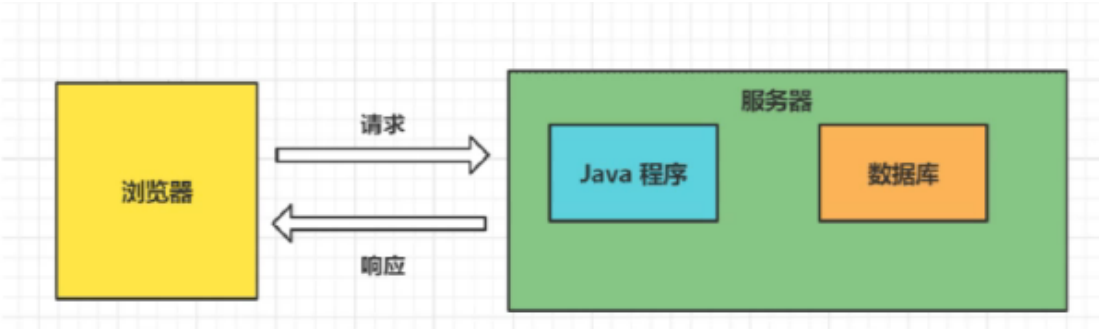
Java Web

Web项目的是实质?

两个用例：客户端和服务器

Web项目就是客户端和服务器的交互

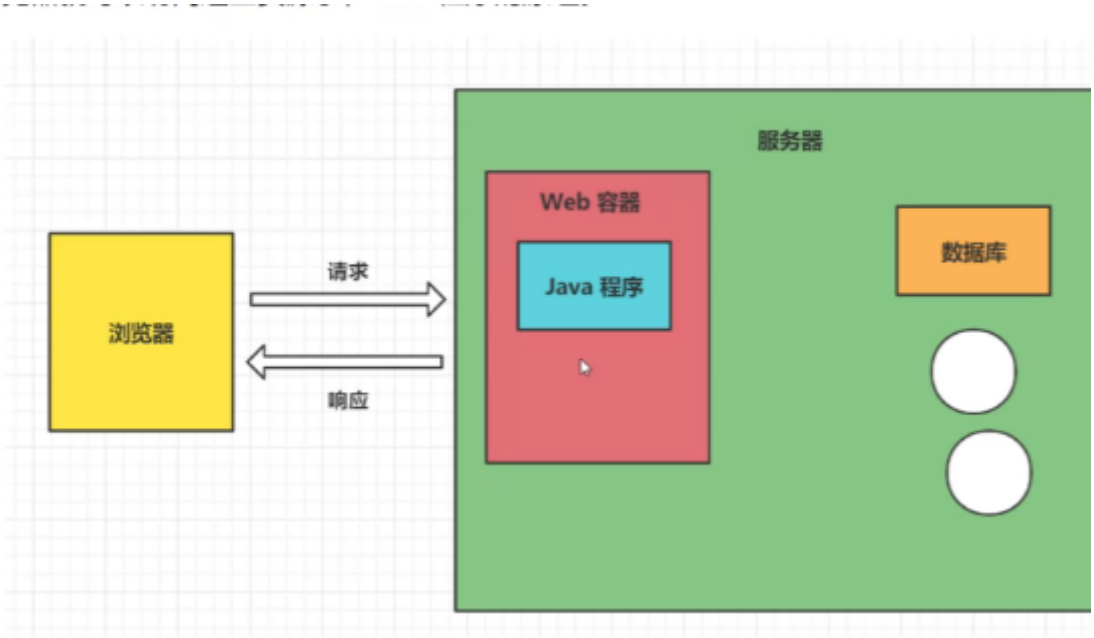
代码中，客户端就是浏览器，服务器就是java程序+数据库



浏览器不能直接访问服务器中的资源，出于安全性考虑，但服务器上的程序是允许被访问的，所以服务器可以将这些允许的资源开放出来，供客户端访问

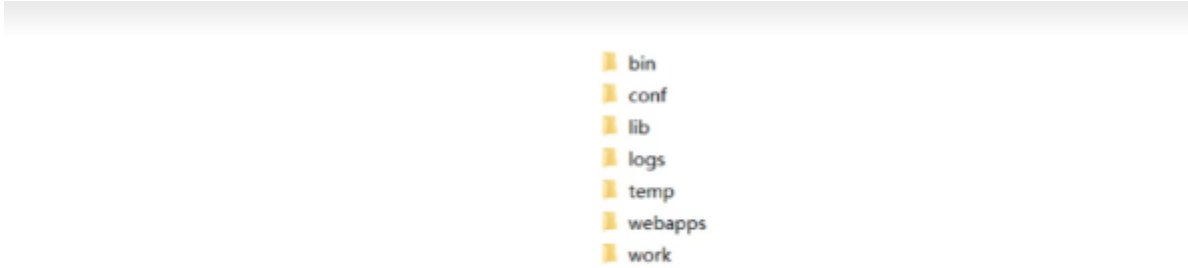
如何实现?

通过web容器来解决，它是一个产品，，安装在服务器计算机上的一个服务产品，可以在Web容器上放置允许客户端访问的资源，然后启动web容器，那么客户端就可以访问这些资源了，web程序的原理



首先要在服务器上安装web容器，然后将写好的java程序部署到web容器中，启动web容器，客户端浏览器就可以通过访问web容器，实现对程序的请求

Web容器： Tomcat 、jetty 、WebLogic、jboss、IS、Nginx



bin：存放各个平台下启动和停止 操作 Tomcat 服务的脚本文件

conf：存放各种 Tomcat 服务器配置文件

lib：存放 Tomcat 服务器需要用到的 jar 文件（打包之后的 Java 程序）

logs：存放 Tomcat 启动运行日志

temp：存放 Tomcat 临时文件

webapps：存放运行客户端访问的资源

work：Tomcat 将 JSP 转成 Servlet 之后的文件，存放在这里

要求写一个WEB应用服务，不能使用Servlet接口，用Socket实现，可以响应GET请求，打印请求信息，并判断请求资源，是否存在，不存在则返回404，若存在，返回该资源，并且可以返回默认的静态页面

思路：主线程启动Socket服务，循环接受客户端请求，接收到请求后，将数据流中的数据取出来拼接成字符串，在控制台打印

响应时需要判断该资源是否存在，如果存在，将资源通过输出流响应给客户端，如果不存在，将404信息通过输出流响应给客户端，同时指定一个静态页面作为默认返回

创建类：

MyHttpServer：定义Socket，循环接收请求

MyHttpRequest:自定义请求对象，解析请求

MyHttpResponse：自定义响应对象，响应请求

Test：测试类，启动server

MyHttpServer

```
public class MyHttpServer {

//    定义路径 user.dir是显示工程的绝对路径 File.separator是分割符
    public  static String WebContent = System.getProperty("user.dir")+
File.separator+"WebContent";
//  定义端口
    private int port =8081;
//    定义域值，程序是否关闭
    private  boolean isShutdown = false;
    public  void receiving(){
        ServerSocket serverSocket = null;
//        InetAddress将IP转换为一个对象
        try {
            serverSocket = new ServerSocket(port,1,
InetAddress.getByName("127.0.0.1"));
```

```
        } catch (IOException e) {
            e.printStackTrace();
        }
//          接受请求
        while(!isShutdown){
            Socket socket = null;
            InputStream inputStream = null;
            OutputStream outputStream = null;

//              获取连接
            try {
                socket = serverSocket.accept();
                inputStream = socket.getInputStream();
                outputStream =socket.getOutputStream();

//                  解析请求,拿到一个request对象
                MyHttpRequest request = new MyHttpRequest(inputStream);
                request.parse();
//                  响应
                MyHttpResponse response = new MyHttpResponse(outputStream);
                response.sendStaticResource(request);
            } catch (IOException e) {
                e.printStackTrace();
            }finally {
                try {
                    socket.close();
                    inputStream.close();
                    outputStream.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }

            }

        }
    }

}
```

MyHttpRequest

```
package com.southwind.Servlet;

import java.io.IOException;
import java.io.InputStream;

public class MyHttpRequest {
    private InputStream inputStream;
    private  String uri;

    public MyHttpRequest(InputStream inputStream) {
        this.inputStream = inputStream;

    }


    public void parse(){
```

```
        StringBuffer stringBuffer = new StringBuffer(1024);
        int i=0;
        byte[] buffer = new byte[1024];
        try {
            i=inputStream.read(buffer);
        } catch (IOException e) {
            e.printStackTrace();
        }

        for (int j = 0; j < i; j++) {
            stringBuffer.append((char)buffer[j]);

        }
       uri = parseUri(stringBuffer.toString());

    }

    public String parseUri(String requestStr){
//        取出需要返回的资源页面
        int index1,index2;
        index1 = requestStr.indexOf(' ');
        if (index1 != -1) {
            //            获取下标位置
            index2 = requestStr.indexOf(' ',index1+1);
            if (index2>index1){
                return  requestStr.substring(index1+1,index2);
            }
        }
        return null;

        }
        public  String getUri(){
            return uri;
        }



}
```

MyHttpResponse

```
package com.southwind.Servlet;

import java.io.*;

public class MyHttpResponse {
    private OutputStream outputStream;

    public MyHttpResponse(OutputStream outputStream) {
        this.outputStream = outputStream;
    }

    public void sendStaticResource(MyHttpRequest request){
        byte[] bytes = new byte[1024];
        FileInputStream fileInputStream= null;
//        将请求中包含的资源进行返回
//        1.获取请求中包含的资源
```

```
        String filePath= request.getUri();
        //2.资源不存在，返回404 资源存在 返回资源 指定默认资源
        if(filePath.equals("/")) {
            filePath = "/index.html";
        }
//          响应

            try {
                String response = null;
                File file = new File(MyHttpServer.WebContent,filePath);
                byte[] fileByte = new byte[(int) file.length()];
                if (file.exists()){
                fileInputStream = new FileInputStream(file);
                fileInputStream.read(fileByte);
                response = new String(fileByte);
               response= warpMessage("200",response);

                }else {
//                    返回404
                   response = warpMessage("404","404notfound");
                }
                this.outputStream.write(response.getBytes());

            } catch (Exception e) {
                e.printStackTrace();
            }


    }
    public  String warpMessage(String status,String response){
        return "HTTP/1.1 "+status+"\r\n"+"Content-type:text/html\r\n"+
                "Content-Length: "+response.length()+
                "\r\n"+"\r\n"+response;
    }
}
```

Test

```
public class Test {
    public static void main(String[] args) {
        System.out.println("Server startup successfully");
        MyHttpServer server = new MyHttpServer();
        server.receiving();
    }
}
```

什么是servlet?

servlet是java web的基石，与平台无关的服务器组件，运行在web容器（Tomcat）中，负责于客户端进行通信。

Servlet可以完成一下功能：

1.创建并返回基于客户端请求的动态html页面

2.与数据库进行通信

```
public interface Servlet {
    void init(ServletConfig var1) throws ServletException;

    ServletConfig getServletConfig();

    void service(ServletRequest var1, ServletResponse var2) throws
ServletException, IOException;

    String getServletInfo();

    void destroy();
}
```

servlet是一个接口，有5个抽象方法