前期基础

构建本地服务器

借助 php study 软件　(安装---启动 *Apache* 和 *Mysql* 服务)

设置服务器文件(*更改网站根目录的文件*)，找到自己的ip地址分享给别人即可被局域网内的用户访问

自己的访问自己：输入自己的ip地址 / 直接输入127.0.0.1(自己访问自己)

此时本计算机就成为本地web服务器

*注意：
1.软件安装目录中最好不要出现中文字符*
2.只有局域网内可访问
3.内网穿透可以接入互联网让所有人访问

端口问题：默认服务器软件端口为 80

端口是为软件分配资源，让不同软件可以正常运行，如果出现冲突，更改一下端口或关闭冲突的软件即可

例如：不同的音乐软件，同时播放音乐，若端口冲突则不能同时播放

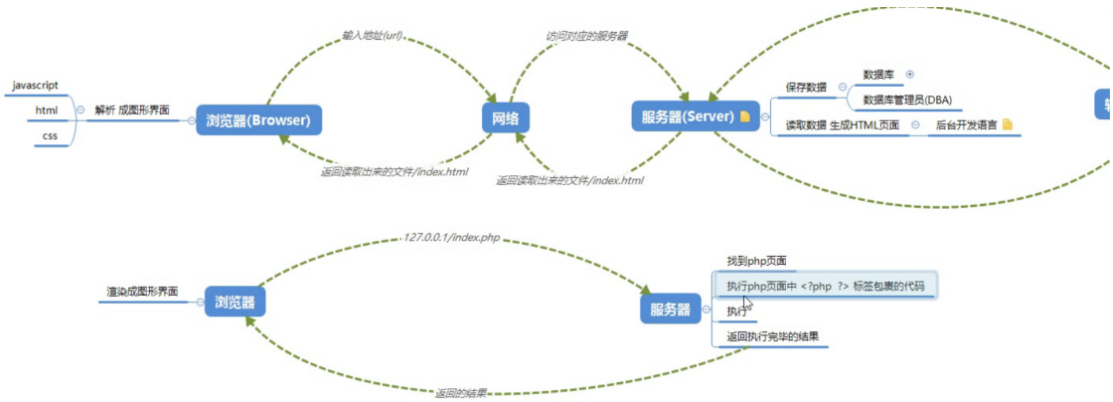
浏览器机制

输入url地址……>借助网络访问并获取服务器上的文件….>浏览器拿到文件后解析成图形界面

- url地址的组成

  http：//www.baidu.com/index.html

  http：服务器与客户端之间的通信协议　www.baidu.com：服务器名称　/index.html：资源在服务器上具体的存放位置



CS架构

浏览器 Browser　　服务器 Server　　客户端　Client

B & S 架构：浏览器 和 服务器；

- 常规的商业网站都属于 bs 架构
- 十分便捷，但性能受限
- B & S 架构 又称为特殊的 C & S 架构

C & S 架构： 客户端 和 服务器

- 必须安装对应的客户端
- .性能更好，画面更炫

4.数据库

服务器中用于***保存数据*** 的软件系统 叫做 数据库

数据库相关的维护岗位叫做： 运维 / 网管 / 数据库管理员(DBA)

功能： 提供很多保护数据安全的功能

5.后台开发语言

作用：读取数据 并 生成相应的界面

包含{ php java python go c ……} php语法和js很像

后台语言应放在服务器中

网络传输协议

1.常见协议

- http、https超文本传输协议
- fps 文件传输协议
- smtp 简单邮件传输协议

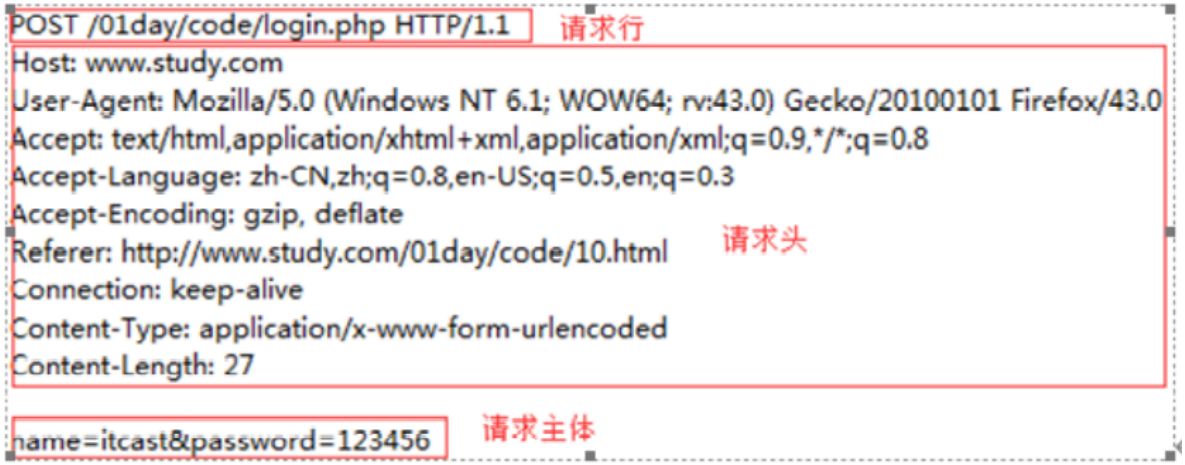
2.http 协议（超文本传输协议）

- 网站是基于http协议
- HTTP协议：超文本传输协议，协议规定了浏览器和万维网服务器之间的通信规则
- 对由客户机到服务器的请求（request）和从服务器到客户机的响应（response）进行了约束和规范

`常用请求方式：get post  put delete`

3.请求报文

`包含：请求行、请求头、请求主体`



请求行：

- 由请求方式、请求URL和协议版本构成
- 例：`POST /s?ie=utf-8 HTTP/1.1`

请求头

- Host：localhost请求的主机
- Cache-Control：max-age=0控制缓存
- Accept：/接受的文档MIME类型
- User-Agent：很重要
- Referer：从哪个URL跳转过来的
- Accept-Encoding：可接受的压缩格式

请求主体

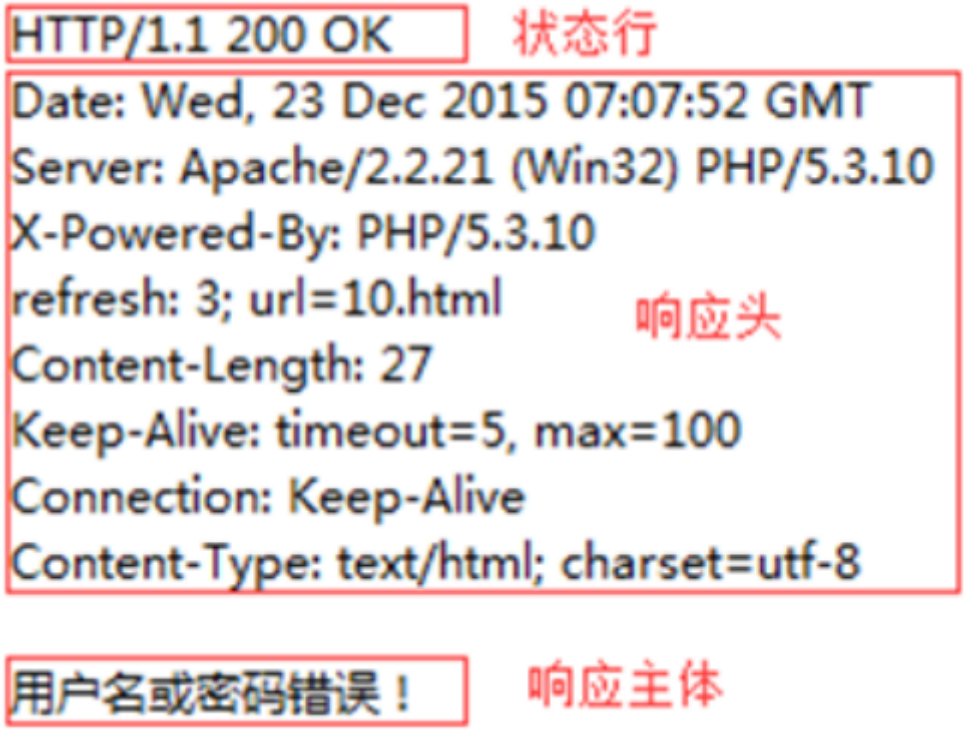
即传递给服务端的数据

注：当以post形式提交表单的时候，请求头里会设置`Content-Type: application/x-www-form-urlencoded`

以get 形式当不需要设置请求头

4.响应报文

`包含：状态行、响应头、响应主体`



状态行

- 由协议版本号、状态码和状态信息构成

响应头

- Date：响应时间
- Server：服务器信息
- Content-Length：响应主体长度
- Content-Type：响应资源的MIME类型

响应主体

即服务端返回给客户端的内容；

状态码	含义
100～199	表示成功接收请求，要求客户端继续提交下一次请求才能完成整个处理过程
200～299	表示成功接收请求并已完成整个处理过程
300～399	为完成请求，客户需进一步细化请求。例如，请求的资源已经移动一个新地址
400～499	客户端的请求有错误
500～599	服务器端出现错误

常见的有200代表成功、304文档未修改、403没有权限、404未找到、500服务器错误

其他基础

from表单提交数据

- `action`：指定提交的`url`
- `method`：指定提交的方法`post、get`，默认是`get`
- `name`：表单的名称，对提交的数据进行标记
- `target`：规定在何处打开页面`_blank  _self _parent _top`
- 注：每一个表单元素都要有`name`属性，它是作为提交后的key值

提交后的数据格式

- `get`提交 测试方便
  - 提交的数据拼接在`url`中 `xxx.php?key1=value1&key2=value2`
  - 问题：
    - 数据的安全性问题
    - `url`理论无限制长度，但部分浏览器会有限制
- `post`提交
  - 如果要上传文件必须使用`post`
  - 浏览器端没有提交数据大小的限制，服务器可能存在限制

了解XML

- XML：可扩展标记语言，类似HTML，其宗旨是用来传输数据，具有自我描述性（固定的格式的数据）
- XML中没有预定义标签，完全自定义，用来表示一些数据（而HTML使用的都是预定义标签）
- 已经被淘汰的数据传输格式，替代者：JSON

在很久之前使用 XML格式进行数据的前后端交互，再利用js对xml中的数据进行解析。

```
例：原数据：姓名：孙猴子，年龄108    用XML表示：
<student>
    <name>孙猴子</name>
    <age>108</age>
</student>
```

ajax编程

- ajax本质是在HTTP协议的基础上以异步的方式与服务器进行通信 (异步JS和XML)
- 异步：指某段程序执行时不会阻塞其它程序执行，其表现形式为程序的执行顺序不依赖程序本身的书写顺序，相反则为同步。
- 异步：各干各的   同步：必须执行完上一步才能下一步
- XMLHttpRequest：浏览器的内置对象，用于与服务器通信，而不刷新页面，从服务器上请求数据

ajax优点

- 可以无刷新于与服务器进行通信

- 允许根据用户事件更新部分页面内容

ajax缺点

- 没有浏览历史，无法回退页面
- 存在跨域问题 (同源)
- SEO优化不友好，无法爬取到ajax中的数据

使用步骤

步骤一：创建对象

```
var xhr = new XMLHttpRequest();
```

步骤二、设置请求行; 请求方式 及 请求的url地址

```
// get请求:数据追加在url地址后面  ? 分割 采用键值对追加数据
xhr.open('GET','index.php?name=lihua&agr=11');
```

步骤三、请求头 ()

- GET请求可以省略；post不发送数据时也可以省略
- 请求头的设置必须放在 open后面
- 请求头可以设置一些自定义请求头信息
  - 浏览器会有安全机制限制自定义的请求头,可以在后端进行配置即可解决
  - 后端配置：`response.setHeader("Access-Control-Allow-Headers","*)`表示接受任意的请求头

```
// Content-type 设置请求体内容的类型

//发送json格式数据:
xhr.setRequestHeader("Content-type","application/json;charset=UTF-8");
//发送表单数据
xhr.setRequestHeader("Content-type","application/x-www-form-
urlencoded;charset=UTF-8");
//发送纯文本（默认值）
xhr.setRequestHeader("Content-type","text/plain; charset=UTF-8");
//发送html文本
xhr.setRequestHeader("Content-type","text/html; charset=UTF-8");

//字符编码可带可不带
```

步骤四、注册回调函数两个方法

- 方法1.

```
 xhr.onload = function(){ console.log(xhr.responseText); } //打印服务器返回的数据
```

- 方法2（兼容性好）：

```
// 事件绑定，处理服务端返回的结果
// xhr.readyState,表示状态 4响应完成(得到全部结果)  3:接收到响应主体(部分结果) 2:send方法调用完毕   1:open方法执行完毕得到头信息 0:初始的数据值
// xhr.status 响应码 2xx表示成功  304文档未修改   403没有权限  404代表没有权限   500服务器错误
xhr.onreadystatechange = function(){
   if(xhr.readyState == 4 && xhr.status >= 200 && xhr.status < 300){
        console.log(xhr.responseText);
      // 处理结果 响应 行、头、空行、体
      // 响应行的状态码:xhr.status   响应状态字符串:xhr.statusText
      // 响应体:xhr.response
      console.log(xhr.status);
}}
```

步骤五、请求主体

- send中的数据必须是**字符串类型**
- 如果是对象或其他数据格式，需要使用js方法进行转化

```
// get请求为空，或者写null
// post请求发送的数据写在请求主体中，没有数据时为 空/null
xhr.send(null)
```

更多API

- `xhr.onreadystatechange = function () {}` 监听响应状态 当响应状态码发生改变时执行
- `xhr.responseText  或  xhr.responseXML  //响应主体`
- `xhr.getAllResponseHeaders()  //获取全部响应头信息`
- `xhr.getResponseHeader('key')  //获取指定头信息`
- `xhr.responseType = 'json'` 设置响应体数据的类型，自动转换，写在最外层
- get请求的效率更高，限制大小约4k； post则没有大小限制

GET和POST差异

1. GET没有请求主体，使用xhr.send(null)
2. GET可以通过在请求URL上添加请求参数
3. POST可以通过xhr.send('name=itcast&age=10')
4. POST需要设置请求头
5. GET效率更好（应用多）
6. GET大小限制约4K，POST则没有限制

代码示例

发送get请求

```
var xhr = new XMLHttpRequest();
xhr.open('GET','index.php?name=lihua&agr=11');
//get请求不需要设置请求头
//打印服务器返回的数据
xhr.onreadystatechange = function(){
    //判断 服务器返回了所有结果  并 响应成功
  if(xhr.readyState === 4 && xhr.status >= 200 && xhr.status < 300){
       console.log(xhr.response); //处理返回的结果
           //处理函数。。。
  }else{}
}
xhr.send(null)
```

发送post请求

```
// 创建对象 - 设置请求行 - 设置请求头 - 注册回调函数 - 发送请求
var xhr = new XMLHttpRequest();
xhr.open('POST','index.php');
xhr.setRequestHeader("Content-type","application/x-www-form-urlencoded");
xhr.onreadystatechange = function(){
  if(xhr.readyState == 4 && xhr.status == 200){
      console.log(xhr.response);//处理返回的结果
      //处理函数。。。
}}
xhr.send('name=lihua&age=11');
```

其他内容

- 超时设置
  - `xhr.timeout = xxx` xxx为数字，设置超时的毫秒数，不带单位
  - `xhr.ontimeout = function(){ }` 请求超时的回调函数
- 网络异常回调
  - `xhr.onerror = function(){ }` 网络异常时出发回调函数
- 手动取消发送的请求
  - `xhr.abort()` 当触发该方法时，就取消对应的ajax请求
  - 注：如果取消和创建请求不在一个函数体内，需要将new的接收者提前声明
- 请求重复发送问题
  - 请求较慢，用户重复多次发送同一个请求时，只保留最新的请求，能提高效率
  - ```
    //代码示例：可以设置一个状态 标识变量
    let isSending = false;
    btn.onclick = function(){
        //判断 状态标识，如果正在发送就取消上一个请求
        if(isSending) x.abort();
        xhr = new XMLHttpRequest();
        //新建请求 就修改状态标识
        isSending = true;
        xhr.open("GET",'http://www.baidu.com/');
    ```
    ```

```
    x.send();
    x.onreadystatechange = function(){
        if(x.readyState === 4){
            //响应完成就调整状态标识为 false，因为请求可能失败，所以不用判断相应码
            isSending = false;
        }
    }
}
```

- IE缓存问题
  - IE浏览器，在ajax相同请求内容时，会走缓存信息，影响最新ajax数据的接受
  - 解决办法：请求时携带参数，内容为当前时间戳

```
xhr.open("GET","http://127.0.0.1:8000/ie?t="+Date.now());
```

封装ajax请求

axios

> 基于Promise对xml的封装

- 可以在node.js / 浏览器中发送请求
- 支持promise 异步方案
  - axios请求的返回值是一个`promise对象`
- 响应状态码 头信息 字符串 响应体 都包含在响应结果中，且**经过处理**方便使用
- 安装：
  - yarn add axios 或 npm i axios
- **axios请求方法** 别名

方法名 (url、data、method 可以不在config中配置)	**描述**
axios.request(config)	
axios.get(url[,config]) 或 axios.post(url[,data[,config]])	
axios.delete(url[,config])	
axios.head(url[,config])	
axios.options(url[,config])	
axios.put(url[,data[,config]]) 或 axios.patch(url[,data[,config]])	

- **response 响应对象**

```
{
   data:{}, //服务器发回的响应数据
   status:200, //服务器响应的http状态码,可用于判断请求是否成功
   statusText:'OK',//服务器响应的HTTP状态描述
   headers:{}, // 服务器响应的消息报头
   config:{},// 为请求提供的配置信息
   request:{}, // 生成此响应的请求
}
```

- **axios请求配置**

```
{
   // 将自动加在请求的url之前，除非请求的url是一个绝对URL
   baseURL:'http://wudetian.top/',
   // transformRequest ......
}
```

- 更多内容查看axios相关课程和文档。。。

```
// 发送GET请求
axios.get("http://127.0.0.1:8000/index.html",{
   //参数:
   params: {
      id:100,
      name:"1223"
   },
   // 设置请求头信息
   headers: {
      'content-type':'application/x-www-from-urlencoded'
   }
}).then(value => {
   // 输出返回的结果,包含很多信息
   console.log(value);
})
```

```
axios({
   // 请求方法  默认值为get
   method : "POST",
   // 请求url
   url:"/front",
   // url参数,get请求/delete请求
   params:{
      vip:10,
      name:"wu"
   },
   // 头信息
   headers:{
      a:100,
      b:200
   },
   // 请求体参数,post请求/put请求...
   data:{
      username:"admin",
```

```
      password:"123456"
   }
}).then(response=>{
   // 处理请求的结果：响应状态码 头信息 字符串 响应体
   console.log(response);
}).catch(err=>{
   // 处理错误信息
   console.error(err);
})
```

```
// axios请求的返回值是一个 promise对象
// 例: 使用promise 的 then方法 解决回调地狱问题
let username;
const userpromsie = axios.get('http://api.github.com/users');
userpromsie.then(response=>{
   username = response.data[0].login;
   return axios.get(`http://api.github.com/users/${username}/repos`);
},(err)=>{
   console.log(err);
})
```

JQuery

```
$.ajax({
    url:'',   //请求的url
    data:{a:xxx,b:xxx},   //参数
    type:'',        //请求类型GET POST
    dataType:'json',      //响应体结果
    success:function(data){},   //成功的回调函数
    timeout:2000,       //超时时间，毫秒数
    error:function(){},      //失败的回调
    headers:{}      //头信息
})
```

fetch方法

- 自带原生方法
- 存在兼容性问题
- fetch发送请求默认是不发送cookie的，可以配置其`credentials`项
  - `omit`: 默认值，忽略cookie的发送
  - `same-origin`: 表示cookie只能同域发送，不能跨域发送
  - `include`: cookie既可以同域发送，也可以跨域发送

```
fetch("http://wudetian.top",{
    // 请求方法:
    method:'POST',
    // 请求头:
    headers:{
        //发送json格式数据:
        Content-type:"application/json;charset=UTF-8"
    },
    // 请求体
    body:"username=wzt&password=admin"
}).then(response => {
    // 处理响应信息
})
```

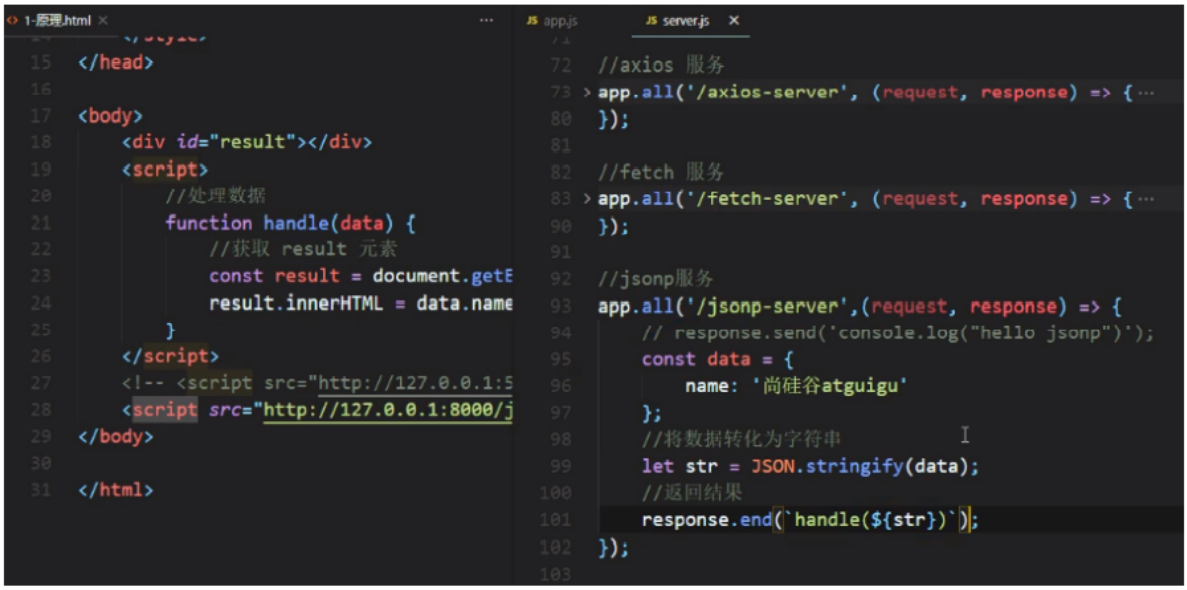
同源策略

- 同源：协议、域名、端口号，必须完全一致，违背同源策略就是跨域。
- ajax默认遵循同源策略

跨域问题

JSONP方案

- 原理：
    - 利用 `<script>` 标签可以跨域的特性，
    - 在服务器端返回一个函数（包含要返回的数据），因为 `<script>` 要传js代码
    - 在本地定义对应的的函数，将函数中的数据拿出来使用



CORS方案

- 跨域资源共享CORS： 官方的跨域解决方案，
- 特点:
  - 完全在服务器端设置，不需要在客户端进行操作
  - 通过设置响应头信息，告诉浏览器允许跨域，浏览器收到信息就会放行
- 更多内容参看相关文档。。。

```
// 常在 后端 设置的三个信息  * 表示对所有都支持
// 允许请求的url
response.setHeader("Access-Control-Allow-Origin","*");
response.setHeader("Access-Control-Allow-Origin","http://wudetian.top:8000");
// 允许携带的请求头信息
response.setHeader("Access-Control-Allow-Headers","*");
// 允许请求的方法，默认只允许get 和 post
response.setHeader("Access-Control-Allow-Methods","*");
```

代理服务器