# 基础概念介绍

## 1.1 服务器

提供服务的机器

## 1.2 浏览器

通过浏览器浏览网页内容

## 1.3 HTTP协议

HyperText Transfer Protocol 超文本传输协议

作用：规定浏览器如何发送请求，规定后端如何解析请求。规定后端如何返回数据，前端如何接收数据。

## 1.4 URL

统一资源定位符

前端浏览器用于定位服务器的字符串。

如下：

https://www.baidu.com/ 百度的主页URL

http://www.163.com/ 网易的主页URL

https://sports.163.com/nba/ 网易的URL

http://localhost:3000/flappybird/index.html

结构：一个完整的URL是由

demo: http://localhost:3000/flappybird/index.html?a=1&b=2&c=3#ccc

协议（protocol）：规定当前使用的协议

域名（domain）：指定要访问的域名（也叫主机）

端口（port）：端口 一共65535个

路径（path）: 具体的资源路径

查询串（query）：提供一些数据给服务器使用（静态资源是不需要这个的）

哈希（hash）：提供的一个给前端使用的值

## 1.5 域名

域名指向一个外网的机器的ip地址

域名的来历：

因为计算机之间互相识别身份依赖的是IP地址。浏览器发送请求的时候，要去服务器上获取内容，必须先找到这个服务器。怎么找？通过域名去找，域名与IP地址的映射关系会记录在域名服务器上。

## 1.6 缓存

缓存：暂时存储

浏览器的缓存机制：

当浏览器访问了一个页面之后，会将一部分信息暂时存储到本地。当下一次用户再次打开相同的页面时，会从本地调用这些信息。避免了带宽的浪费，以及提升了用户看到页面的速度。

缓存分两种：

强制缓存

一旦命中强制缓存，则直接使用本地的数据，不再向服务器发送请求。

协商缓存

一旦命中协商缓存，则依旧会向服务器发送请求，询问服务器“这个数据是不是已经更新了？”。如果服务器已经更新数据，则将更新后的内容获取回来，并重新设置缓存的有效期，替代原来的缓存。如果服务器没有更新数据，则不会将任何内容带回来，通知浏览器使用缓存中的数据。

## 1.7 当用户输入网址按下回车之后

第一步，浏览器解析用户输入的URL字符串

第二步，获取域名对应的IP地址

尝试从本地浏览器缓存中获取

尝试从本系统缓存中获取

尝试从路由器缓存中获取

尝试去域名服务器上获取

第三步，开始发送请求，尝试建立TCP连接

第四步，连接建立完毕，浏览器开始按照协议发送请求内容

第五步，服务器接收到，并按照协议开始处理，处理完毕，按照协议返回数据，并断开连接

第六步，浏览器接收到内容，按照协议开始解析。

第七步，浏览器解析第一个返回的HTML页面内容。

从上到下开始依次解析

DOM树、样式树

遇见img标签、link标签、script标签等各种标签都会发送请求

最终渲染完毕

用户看到页面

# HTTP请求

## 2.1 简介

HTTP协议，是规定前后端如何发送并接收信息的。

HTTP请求，指的是真的按照协议规定的方式发送的一次请求。

通常，我们将浏览器发送给服务器的一次，叫做请求。服务器返回给浏览器的一次叫响应。

## 2.2 组成

HTTP请求由四部分：请求首行、请求头、请求空行、请求正文

### 2.2.1 请求首行

首行指的是HTTP请求的第一行

|  |
| --- |
|  |

### 2.2.2 请求头

请求头中的内容，是相关的设定。

|  |
| --- |
|  |

### 2.2.3 请求空行

请求空行就是空白行，它的作用就是分割请求头和请求正文

### 2.2.4 请求正文

请求时所携带的数据都应当存放在请求正文中  
注：GET请求没有请求正文

## 2.3 请求类型

请求是从浏览器到达服务器。

根据目的划分：请求既可以从服务器上带内容回到浏览器。也可以从浏览器带内容到服务器（上传）。

GET：从服务器上带内容回到浏览器的请求

特点：

没有请求正文

将数据携带在URL的query上

因为数据在URL上，所以长度受限制，无法携带太多数据

不能够传递敏感信息

方便分享

触发方式多种多样

POST：从浏览器带内容到服务器的请求

特点：

有请求正文

将数据带在请求正文中

数据量没有上限

不方便分享

只有AJAX和表单可以触发POST请求

PUT、DELETE等

## 2.4 请求状态码

在请求结束之后，服务器会返回一个状态码，表示本次请求的请求结果。

200 成功

302 重定向

304 未修改

404 资源未找到

500 服务器内部错误

## 2.5 HTTP协议的无状态

HTTP协议是一个无状态的协议。

简单来说：通过HTTP协议规定建立的连接，在传递数据完毕之后会断开连接。

在HTTP协议1.0时，HTTP请求头中connection字段的值为close。

在HTTP协议1.1时，HTTP请求头中connection字段的值为close和keep-alive。

close：当服务器传递完毕之后，断开连接

keep-alive: 当服务器传递完毕之后，保持一段时间的连接，如果在这段时间内又有HTTP请求来了，就利用这个连接，避免了连接的建立和销毁的开销。

# NodeJS

NodeJS是一个JS的执行环境。

从今天开始，JS就可以脱离浏览器环境去执行。

## 3.1 简介

官网：<https://nodejs.org/en/>

下载：直接从官网首页选择对应的版本进行下载即可

|  |
| --- |
|  |

左边的是长期稳定版本：10.15.3

右侧的是最新特性版本：12.2.0

## 3.2 安装

略

## 3.3 运行JS文件

安装完毕之后，我们可以在命令行中通过node命令运行文件。

书写一个js文件：

|  |
| --- |
| 1. console.log("hello world"); |

输出：输出在命令行中

|  |
| --- |
|  |

## 3.4 特点

NodeJS什么都能做，我们的课程中，只是用来搭建服务器。

### 3.4.1 单线程

在执行代码时，只会有一个线程去执行。

### 3.4.2 非阻塞I/O

I/O: 输入/输出

从磁盘中向内存中输入内容，叫做Input。

从内存中向磁盘中输出内容，叫做Output。

非阻塞介绍：

JS线程执行任务时，如果遇见IO任务，只是开启该任务，并去执行线程中下一个任务。

阻塞介绍：

JS线程执行任务时，如果遇见IO任务，开启该任务并等待，直到任务结束再去执行线程中下一个任务。

### 3.4.3 事件驱动

因为非阻塞IO的原因，JS线程开启任务之后就离开，如果任务执行完毕了，还有后续任务需要执行，可是线程已经走了。此时，会触发一个事件，该事件会将后续任务重新放入JS执行队列（NodeJS的事件循环队列）

### 3.4.4 适用场景

因为非阻塞IO的原因，NodeJS搭建的服务器特别合适异步多，并发量大。

特别不合适计算。因为计算是同步的。

### 3.4.5 诊所案例

诊所：NodeJS搭建的服务器

医生：单线程

医生要做的事情：执行队列

护士：内存

库房：硬盘

患者：前端请求

清早起床，开门。（服务器启动）

十个患者开始排队发送请求。（前端请求来了）

医生将它们放到“要做的计划中”（执行队列中）

***阻塞IO场景：***

给第一个患者诊断（医生要做）、抓药（安排一个护士去库房中抓药）。如果此时医生等待护士回来再继续后续任务（下医嘱）。那么就叫做阻塞I/O.

***非阻塞IO场景：***

给第一个患者诊断（医生要做）、抓药（安排一个护士去库房中抓药）。医生不等护士回来，就给第二个患者诊断。并继续如此做。直到10个患者都诊断完毕。就没有事情做了。

如果护士在患者还没有结束的情况下回来了（触发事件），就将后续任务给医生添加到执行队列中

如果护士在患者已经全部被诊断完毕的情况下回来（触发事件），就将后续任务给医生添加到执行队列中

### 3.4.6 医院案例

医院：其它多线程后台服务器

当来了患者以后，不论来多少个患者，每一个患者安排一个医生，专职照顾。

医生诊断完毕，通知护士抓药，医生等着。

## 3.5 模块化

NodeJS是模块化开发。

Seajs是模块化工具、RequireJS也是模块化工具。

Seajs遵循的规范：CMD规范

定义模块：define定义 一个JS文件中只能有一个模块

暴露接口：

module.exports打点或者赋值去暴露内容

exports打点暴露内容

引入：

require引入

RequireJS遵循的规范：AMD规范

定义模块：define定义 一个JS文件中只能有一个模块

暴露接口：

通过return暴露

引入：

依赖集合引入

require引入

NodeJS遵循CommonJS规范：这是一个同步规范

定义模块：一个JS文件就是一个模块

暴露接口：

通过module.exports和exports暴露

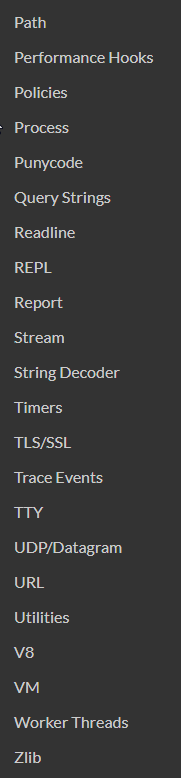
引入：

require引入

## 3.6 模块的分类

NodeJS中有两种模块，一种是内置模块也叫做核心模块。另一种是第三方模块。

核心模块：



其中，与服务器搭建有关系的：http、https、url、querystring、path、fs

第三方模块：只要不是核心模块就是第三方模块。也就是我们定义的每一个JS文件都是第三方模块。

## 3.7 演示第三方模块

定义模块：a.js b.js

a.js:

|  |
| --- |
| 1. var b = require("./b.js"); 2. console.log(b); |

b.js

|  |
| --- |
| 1. // 定义内容 并暴露 2. var a = 11; 3. module.exports.a = a; |

执行：

|  |
| --- |
|  |

如果将a.js修改为require("b.js") 或者是require("b");

|  |
| --- |
|  |

注：引入第三方模块时 我们必须以./开头

## 3.8 node\_modules

这是一个专门存放node第三方模块的文件夹。该文件夹只有一个特征就是名字必须是node\_modules

因为只有当名字是它时，node才认识。

特点：

当我们将第三方模块放入该文件夹时，就可以像引入核心模块一样引入第三方模块了

不论该文件夹存在于当前层级还是上一层级或者更上层级，只要在运行模块的直接祖先级层级中存在就能找得到

demo:

当前模块的目录：

C:\Users\Administrator\Desktop\2019年5月13日AJAX第一天\案例笔记\03 Node第三方模块文件夹\

|  |
| --- |
| 1. var b = require("b"); 2. console.log(b); |

只要被引入的b.js文件存在于

C:\Users\Administrator\Desktop\2019年5月13日AJAX第一天\案例笔记\03 Node第三方模块文件夹\node\_modules

C:\Users\Administrator\Desktop\2019年5月13日AJAX第一天\案例笔记\node\_modules

C:\Users\Administrator\Desktop\2019年5月13日AJAX第一天\node\_modules

C:\Users\Administrator\Desktop\node\_modules

C:\Users\Administrator\\node\_modules

C:\Users\node\_modules

C:\node\_modules

在这些里面有任何一个拥有，就能够成功执行

# HTTP模块

HTTP模块负责搭建HTTP服务器

demo:

|  |
| --- |
| 1. // 引入核心模块http 2. var http = require("http"); 3. // 调用创建服务器的方法 4. var server = http.createServer(function() { 5. }); 6. // 监听 7. server.listen(3000); |

以上代码中：

引入

http.createServer方法负责创建一个http服务器对象

该方法接收一个参数 是一个函数 该函数表示当有请求到达服务器时 如何响应

该函数有两个参数：req、res

req: request 本次HTTP请求对象

url: 本次HTTP请求的URL的path往后部分，没有hash部分

method:本次HTTP请求的方式、类型

headers: 本次HTTP的请求头对象

res: response 本次HTTP响应对象

end: 该方法负责向前端返回数据并断开连接

write: 该方法负责向前端返回数据

setHeader: 该方法负责设置响应头

server.listen方法负责决定监听在某一个端口

最终，我们可以这样访问：http://localhost:3000/xxxx

不论我们如何访问该服务器，都得到一样的结果：没响应。

# FS模块

## 5.1 介绍

FS指的是File System 文件系统

它可以操作文件和文件夹

## 5.2 创建文件

fs.appendFile(filePath, content, callback);

filePath: 合法的路径

content: 要追加的内容

callback: 回调函数

回调函数中有一个参数，表示异常

如果创建过程失败，则该参数是一个错误对象

如果创建过程成功，则该参数为null

demo:

|  |
| --- |
| 1. // 引入FS模块 2. var fs = require("fs"); 3. // fs.appendFile方法用于创建文件、追加文件内容 4. fs.appendFile("a.txt", "今天第一天上课，看见很多帅哥美女，很开心", function(err) { 5. console.log(err); 6. if (err) { 7. console.log("创建失败"); 8. return; 9. } 10. console.log("创建成功"); 11. }); 12. console.log("证明是异步"); |

执行结果：

|  |
| --- |
|  |

执行结果：

|  |
| --- |
|  |

## 5.3 删除文件

fs.unlink(filePath, callback);

filePath: 合法的路径

callback: 回调函数

回调函数中有一个参数，表示异常

如果删除过程失败，则该参数是一个错误对象

如果删除过程成功，则该参数为null

demo:

|  |
| --- |
| 1. // 把a.txt删除 2. var fs = require("fs"); 3. // 调用对应的删除方法 4. fs.unlink("./a.txt", function(err) { 5. if (err) { 6. console.log("删除失败"); 7. return; 8. } 9. console.log("删除成功"); 10. }); |

执行删除之前：

|  |
| --- |
|  |

删除：

|  |
| --- |
|  |

删除之后：

|  |
| --- |
|  |

## 5.4 修改文件名

fs.rename(oldPath, newPath, callback);

oldPath: 原名称

newPath: 新名称

callback: 回调函数

## 5.5 创建文件夹

fs.mkdir(dirPath, callback);

dirPath: 目录名称

callback: 回调函数

## 5.6 删除文件夹

fs.rmdir(dirPath, callback);

dirPath: 目录名称

callback: 回调函数

注：该方法只能够删除空文件夹

如果文件夹不是空的：

|  |
| --- |
|  |

## 5.7 判定目标状态

fs.stat(targetPath, callback);

targetPath: 目标路径

callback: 回调函数

err: 异常对象

stat: 目标状态对象

stat.isDirectory() 判定目标是否是文件夹 返回值为布尔值

## 5.8 读取文件

fs.readFile(filePath, callback);

filePath: 被读取的文件路径

callback: 回调函数

err: 异常对象

data: 读取到的文件内容

## 5.9 读取文件夹

fs.readdir(dirPath, callback);

dirPath: 被读取的文件夹路径

callback: 回调函数

err: 异常对象

arr: 文件夹下的所有文件和文件夹名称组成的数组（只有一层）

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |