# Adv Web Dev Architecture
# Lecture 4

Amir Amintabar, PhD

# 高级网页开发架构
# 第4讲

阿米尔·阿明塔巴尔，博士

# Outline

- Review
- 1- Server and server scripting
- 2- Node js
- 3- JS is asynchronous, single threaded, restaurant analogy
- 4- Node.js installation
  - Node.js debugging in VSC ( Visual Studio Code)
  - Node.js examples
    - Hello World!
- 5- Module (put your functions in module, just like including js external files), examples
- 6- using built in modules, examples
  - http
  - url
  - fs
- 7- Hosting Node js in shared hosting services, example



# 大纲

- 复习
- 1- 服务器与服务器端脚本
- 2- Node.js
- 3- JavaScript 是异步、单线程的，以餐厅类比说明
- 4- Node.js 安装
  - 在 Visual Studio Code（VSC）中调试 Node.js
  - Node.js 示例
    - Hello World!
- 5- 模块（将您的函数放入模块中，类似于引入外部 JavaScript 文件），附示例
- 6- 使用内置模块，附示例
  - http
  - url
  - fs
- 7 - 在共享主机服务中托管 Node.js，示例

# Review

- Hoisting
- Functions with no return statement, return undefined !
- JS is asynchronous, single threaded, non-blocking!
- Difference between GET and POST
- In JS a function can return a function



# 复习

- 变量提升（Hoisting）
- 不带 return 语句的函数，其返回值为 undefined!
- JavaScript 是异步、单线程、非阻塞的!
- GET 与 POST 的区别
- 在 JavaScript 中，函数可以返回另一个函数

- every team member should have a comprehensive understanding of the entire project, including the work done by their fellow teammates,



- 每位团队成员都应全面了解整个项目，包括其团队成员所完成的工作，

# Code of conduct

- Every team member should have a comprehensive understanding of the entire project, including the work done by their fellow teammates.



# 行为准则

- 每位团队成员都应全面了解整个项目，包括其团队成员所完成的工作。

1

# Server and server scripting



1

# 服务器与服务器端脚本

# What is a server?

- A server is a computer ( or computers) running 24/7 and accessible via the internet
- When a client such as a browser or mobile app sends a requests via protocol such as http, ftp etc, the server looks for the application responsible to deal with that request, and returns the response using those protocols

- In this course, the request from clint to server are carried using http methods such as GET, POST etc.



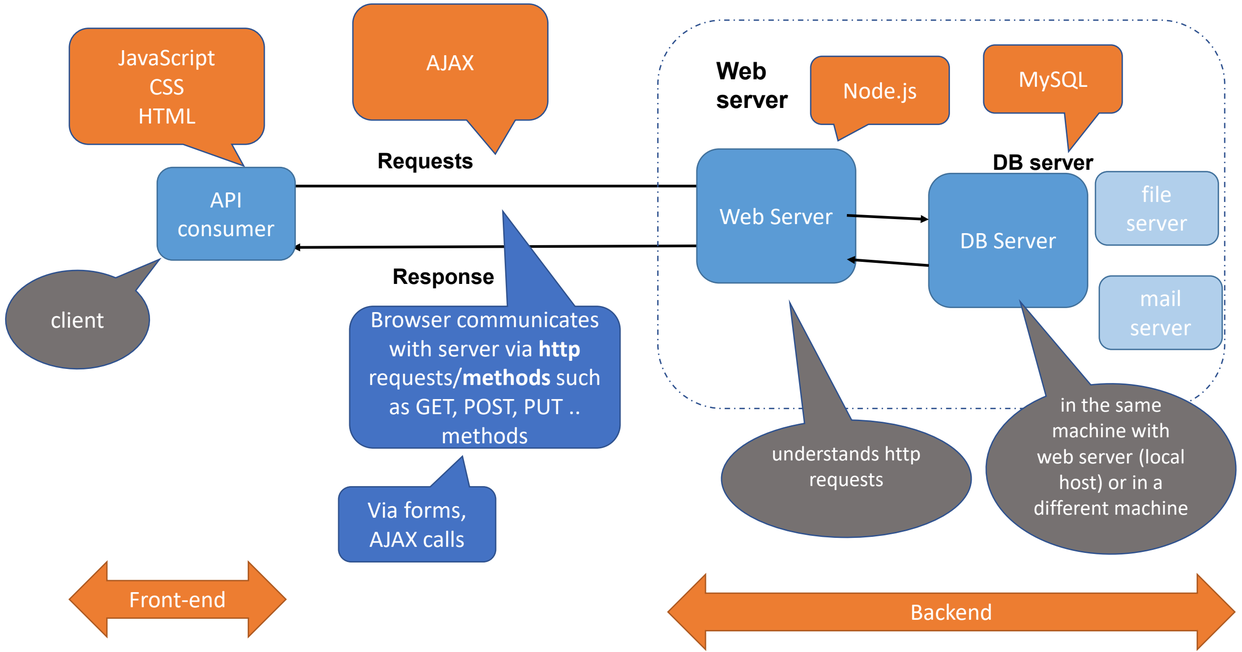
# 什么是服务器?

- 服务器是一台（或若干台）全天候运行且可通过互联网访问的计算机。

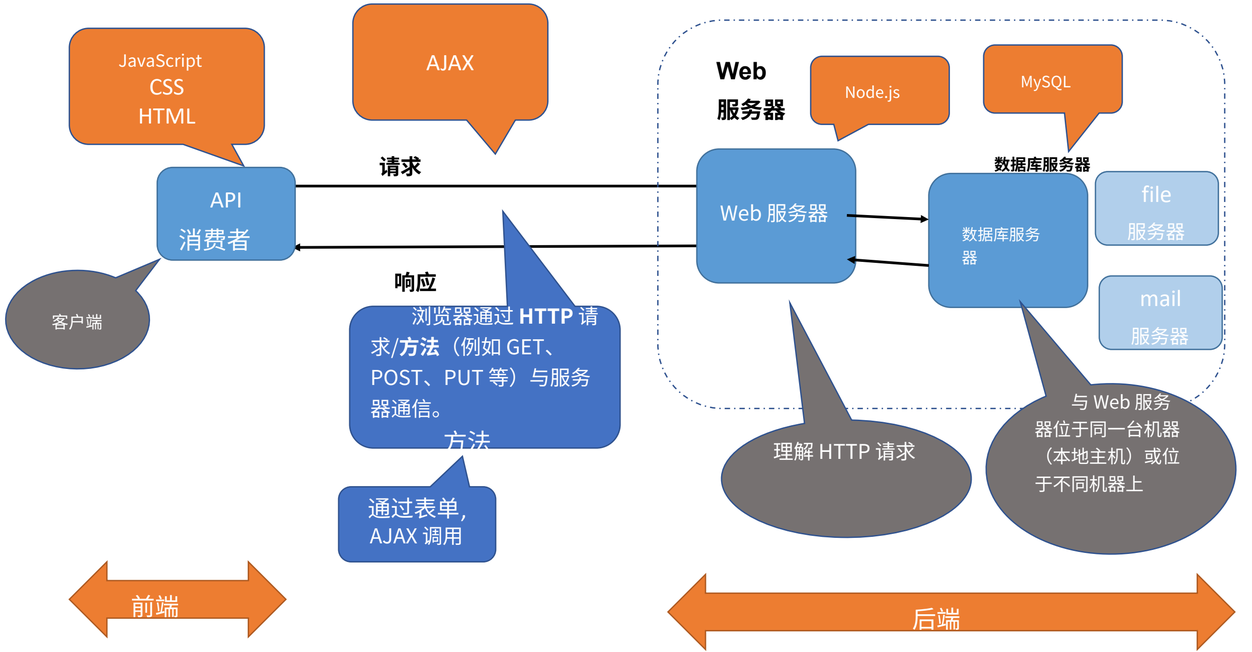
- 当浏览器或移动应用等客户端通过 HTTP、FTP 等协议发送请求时，服务器会查找负责处理该请求的应用程序，并通过相应协议返回响应。

- 在本课程中，客户端向服务器发起的请求将使用 HTTP 方法（例如 GET、POST 等）进行传输。

# What is a server?




# 什么是服务器?



# What a server side scripting language can do?

- It can create a webserver to
- generate dynamic page content
- E.g. opening wordpress.com/index.php at borwser
  - sends a request to the server,
  - The server looks for an app to run php files
  - If the php server is installed it runs that file and returns the result in the form of html
- Server side script can create, open, read, write, delete, and close **files** on the server
- Server side script can collect form data
- Server side script can add, delete, modify data in your **database**



# 服务器端脚本语言能做什么?

- 它可以创建一个 Web 服务器，以
- 生成动态网页内容
- 例如，在浏览器中打开 wordpress.com/index.php
  - 向服务器发送请求，
  - 服务器查找用于运行 PHP 文件的应用程序
  - 如果已安装 PHP 服务器，则会运行该文件，并以 HTML 格式返回结果
- 服务器端脚本可在服务器上创建、打开、读取、写入、删除和关闭**文件**
- 服务器端脚本可收集表单数据
- 服务器端脚本可在您的**数据库**中添加、删除和修改数据

In this course, we are going to use

2

node.js

for
server side (backend) scripting



在本课程中，我们将学习使用

2

节点.js

for
服务器端（后端）脚本编程

# What is Node.js?

- Node.js is an open source, free, server environment that uses JavaScript on the server side

- Node.js is an **open-source**, **cross-platform**, back-end JavaScript **runtime environment** that runs on the
- **Chrome V8** engine
- and executes JavaScript code **outside a web browser**

https://en.wikipedia.org/wiki/Node.js



# 什么是 Node.js?

- Node.js 是一个开源、免费的服务器运行环境，它在服务器端使用 JavaScript

- Node.js 是一个 **开源**、**跨平台** 的后端 JavaScript **运行环境**，基于
- **Chrome V8** 引擎
- 运行，并可在 **浏览器之外** 执行 JavaScript 代码

https://zh.wikipedia.org/wiki/Node.js

# Why nodeJS?

- **Node.js is non blocking!**
- Here is how PHP or ASP handles a file request:
  1. Sends the task to the computer's file system.
  2. Waits while the file system opens and reads the file.
  3. Returns the content to the client.
  4. Ready to handle the next request.
- Here is how Node.js handles a file request:
  1. Sends the task to the computer's file system.
  2. Ready to handle the next request.
  3. When the file system has opened and read the file, the server returns the content to the client.
- Node.js eliminates the waiting, and simply continues with the next request.
- Node.js runs single-threaded, non-blocking, asynchronously programming, which is very memory efficient.



# 为何选择 Node.js?

- **Node.js 采用非阻塞式设计！**
- PHP 或 ASP 处理文件请求的方式如下：1. 将任务发送至计算机的文件系统。
  2. 等待文件系统打开并读取文件。
  3. 将文件内容返回给客户端。
  4. 准备处理下一个请求。
- Node.js 处理文件请求的方式如下：
  1. 将任务发送至计算机的文件系统。
  2. 立即准备处理下一个请求。
  3. 当文件系统完成文件的打开与读取后，服务器将内容返回给客户端。
- Node.js 消除了等待时间，直接继续处理下一个请求。
- Node.js 采用单线程、非阻塞、异步编程模式，内存占用非常高效。

JS is
asynchronous
single threaded
nonblocking



JavaScript 是一种异步、单线程、非阻塞的编程语言。

# The restaurant analogy!

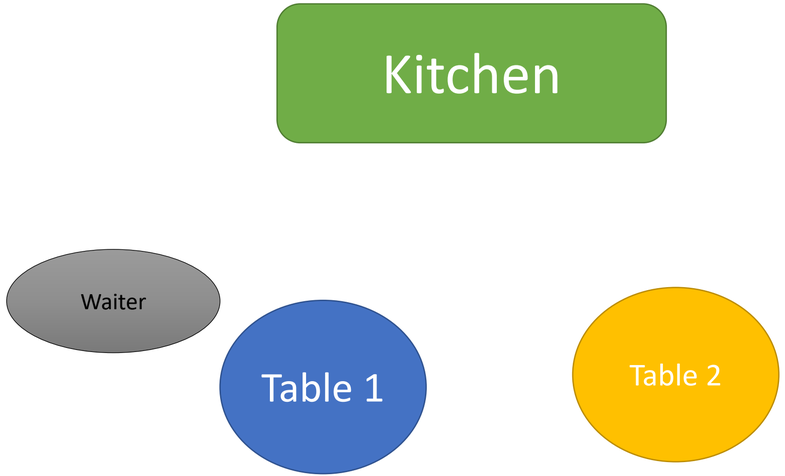


# 餐厅类比！

14

# Example 1

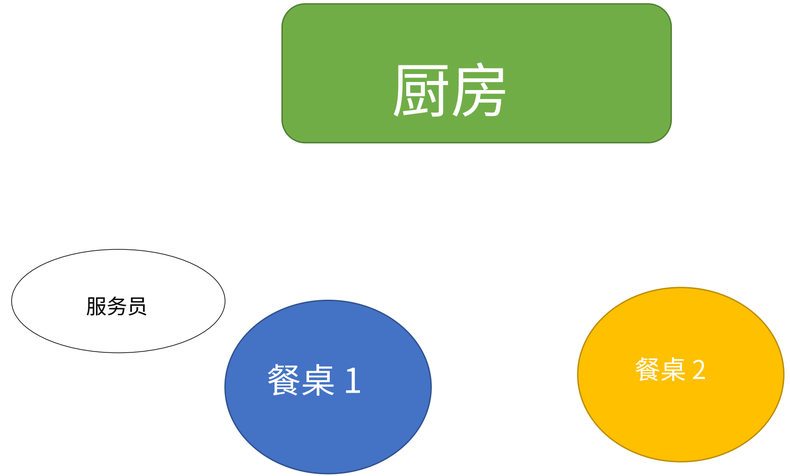
- Same waiter can serve multiple tables






# 示例 1

- 同一名服务员可同时服务多张餐桌



# Example 2

- Same waiter can serve multiple tables, but every time waits for the chief to prepare the meal then (and only then) takes another order.





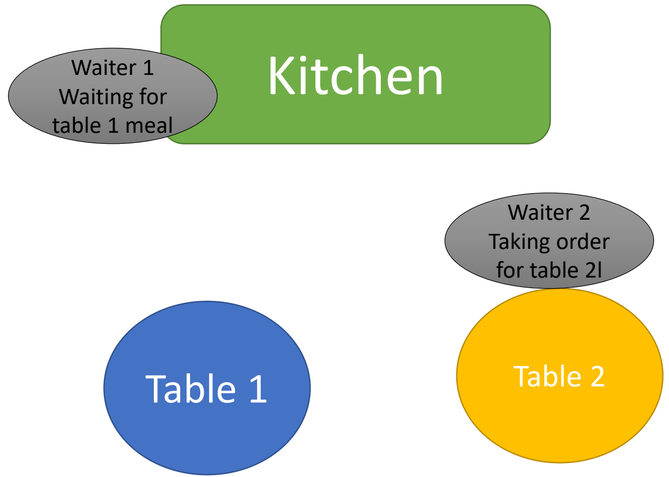
# 示例 2

- 同一名服务员可为多张餐桌提供服务

但每次均需等待厨师备好餐食后（且仅在此之后）才可接收下一份订单。


# Example 3

- Multiple waiters
- Each dedicated to a single table only!

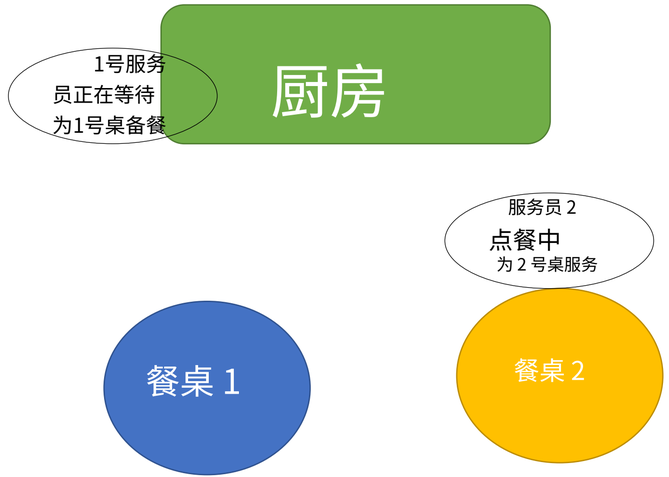





# 示例 3

- 多位服务员
- 每位均专为一张餐桌服务

仅限一张餐桌！



- Example 1: single threaded non-blocking ( asynchronous)
- Example 2: single threaded blocking (synchronous)
- Example 3: multi-threaded



- 示例 1：单线程、非阻塞（异步）
- 示例 2：单线程、阻塞（同步）
- 示例 3：多线程

18

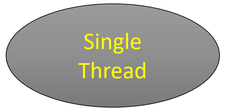
JavaScript(nodejs) is
Single threaded
Asynchronous(non blocking) just like the restaurant in Example 1 ( one waiter serves everyone)

JavaScript（Node.js）是单线程、异步（非阻塞）的，就像示例 1 中的餐厅（一名服务员为所有顾客服务）


19

# Non-blocking

- Same waiter can server multiple tables (don't not get blocked by one table)



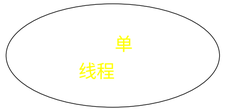


JavaScript(nodejs) is single threaded Asynchronous(non blocking) architecture
Example 1



# 非阻塞式

- 可由同一位服务员处理
  可同时服务多张餐桌（不会因某一张餐桌而被阻塞）

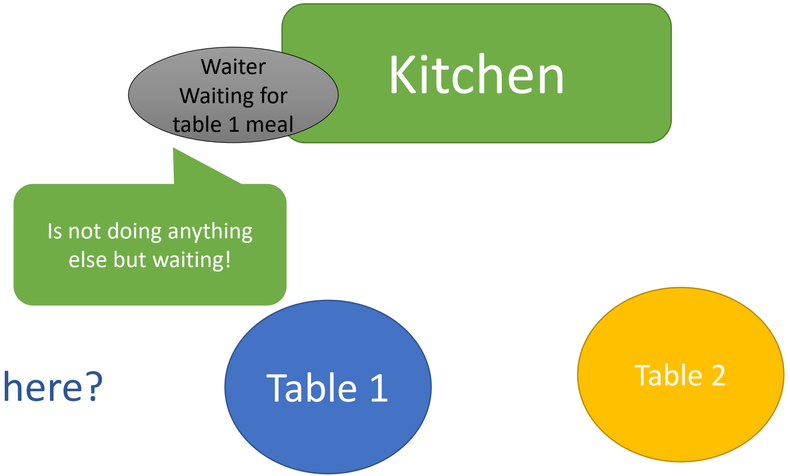




JavaScript（Node.js）采用单线程、异步（非阻塞式）架构。示例 1

# Blocking or synchronous (ASP. Net by nature )

- Same waiter can server multiple tables, but every time waits for chief to prepare the meal then takes another order.



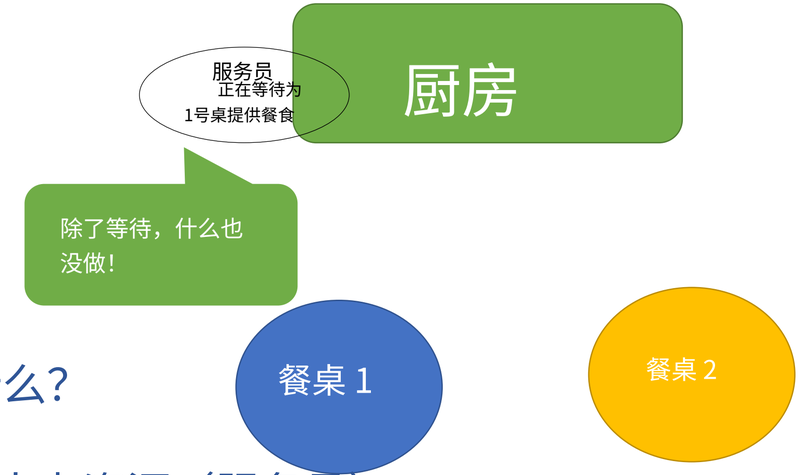

**Q:**What's the problem here?

We are not utilizing our resources (the waiter) efficiently



# 阻塞式或同步式（ASP.NET 本质上即如此）

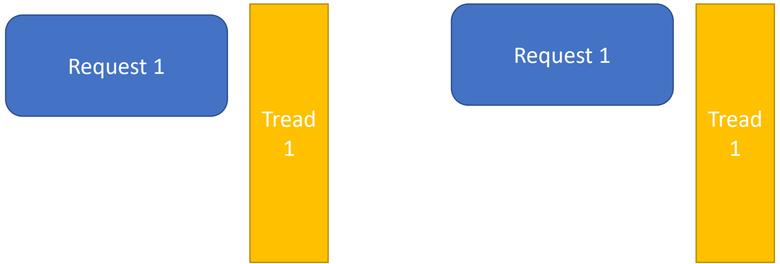
- 同一名服务员可服务多张餐桌，但每次均需等待厨师备餐完毕后，再接收下一份订单。




**问：**此处的问题是什么？

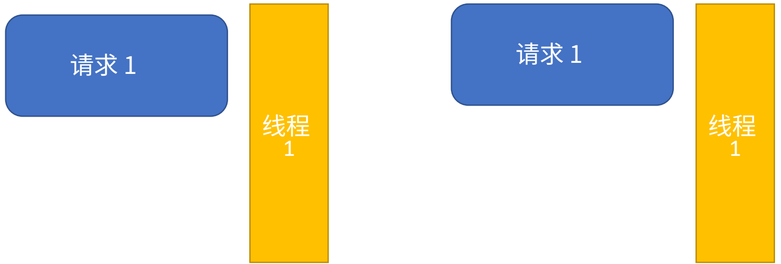
我们未能高效利用人力资源（服务员）

# What if there are multiple requests for  synchronous systems?






# 如果同步系统收到多个请求，情况会如何?



# Imaging 100 concurrent requests for a synchronous system.

Then the system has to create 100 threads! ( 100 waiters!!)


Check this good reference: https://youtu.be/6oP-c30s7co?t=6m8s



# 模拟同步系统处理 100 个并发请求。

此时，系统必须创建 100 个线程！（即 100 个等待线程！！）


请参考以下优质资料：https://youtu.be/6oP-c30s7co?t=6m8s

Lets get started with

4

# node.js

Installation
and
Example code



让我们开始学习

4

# Node.js

安装与
示例代码

# Getting started

- Follow the instructions at …
- …**NodeJs-Install-and-Debug.pdf**
- To install nodejs on your machine locally and to use Visual Studio Code to debug
- Using Nodejs you can easily develop a web server at your pc
- Then you send requests from your browser or the html files stored in your machine to the nodejs server you have developed



# 入门指南

- 请按照……处的说明操作
- ……**NodeJs-安装与调试.pdf**
- 在本地计算机上安装 Node.js，并使用 Visual Studio Code 进行调试

- 借助 Node.js，您可轻松在个人电脑上开发 Web 服务器
- 随后，您可通过浏览器或本地存储的 HTML 文件，向您开发的 Node.js 服务器发送请求

# Example 1 : Hello world!

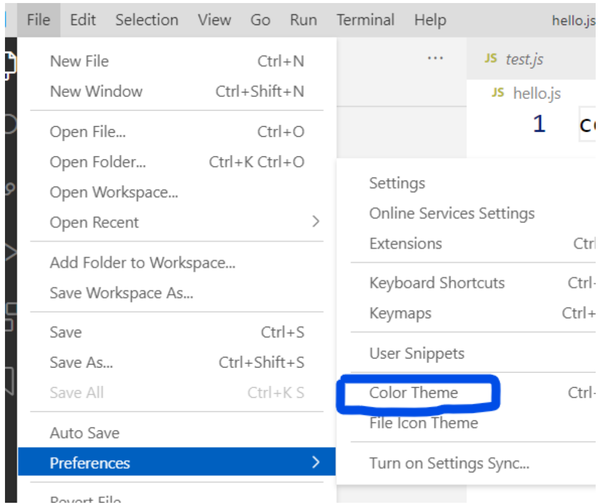
- Just like any desktop programing language. Here console.log acts like System.out.println() in Java




# 示例 1：你好，世界！

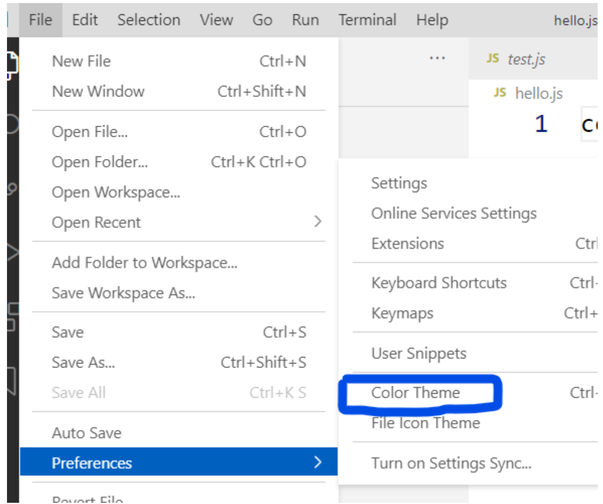
- 与任何桌面编程语言一样。此处的 console.log 功能类似于 Java 中的 System.out.println()

# You can change the look and feel of your VSC (Visual Studio Code)



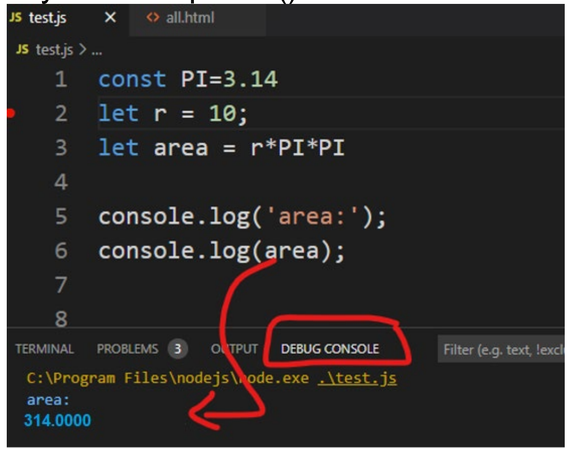


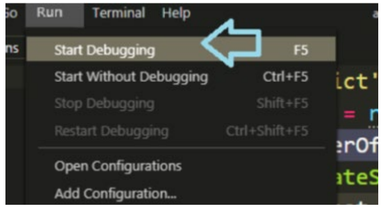

# 您可以自定义 Visual Studio Code（VSC）的外观和体验



# Example 2 : area calculation

- Just like any desktop programing language. Here console.log acts like System.out.println() in Java



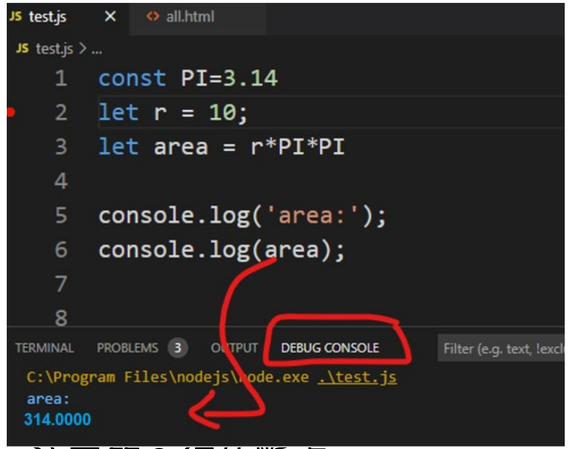

Follow the steps on "...-NodeJs-Install-and-Debug.pdf" to learn how to debug your server side nodejs code in VSC

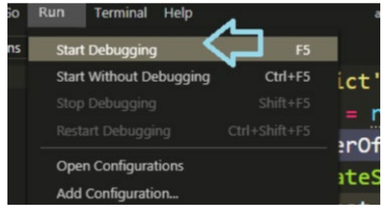
- Note the break point on the line 2
- Bad function identifier/name ! Do we really calculate the area of circle like this? Or is it just circumference



# 示例 2：面积计算

- 与任何桌面编程语言一样，此处的 console.log 起到类似作用
  Java 中的 System.out.println() 方法




请参阅“...-NodeJs-安装与调试.pdf”中的步骤，学习如何在 Visual Studio Code 中调试服务器端 Node.js 代码

- 注意第 2 行的断点
- 函数标识符/名称错误！ 我们真的这样计算圆的面积吗？还是仅仅在计算周长？

5

# Put your functions in **Modules**



5

# 将您的函数放入**模块中**

# What is a Module in Node.js?

- Consider modules to be the same as JavaScript libraries.
- A set of functions you want to include in your application.

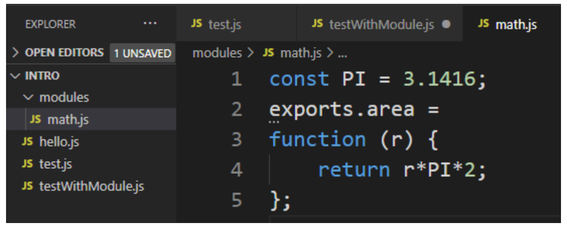


# Node.js 中的模块是什么?

- 可将模块视为与 JavaScript 库相同的概念。
- 一组您希望在应用程序中引入的函数。

# Example 3, area calculation, using module

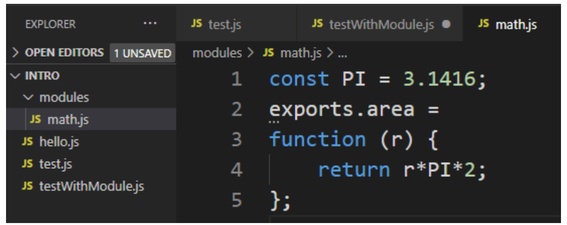
- 1. Our module math.js is located at modules/math.js
- Notice that we use ./ to locate the folder of our module
- That means that the folder is located in the same folder as the main Node.js file.
- 2. Notice we don't need to include the .js when we import the module (its optional)




# 示例 3：使用模块进行面积计算

- 1。我们的模块 math.js 位于 modules/math.js 目录下。
- 请注意，我们使用 ./ 来定位模块所在的文件夹。
- 这意味着该文件夹与主 Node.js 文件位于同一目录下。
  与主 Node.js 文件位于同一目录下。
- 2。注意：导入模块时无需包含 .js 扩展名（此扩展名为可选项）。

6

# More examples using nodejs
# **built-in modules**

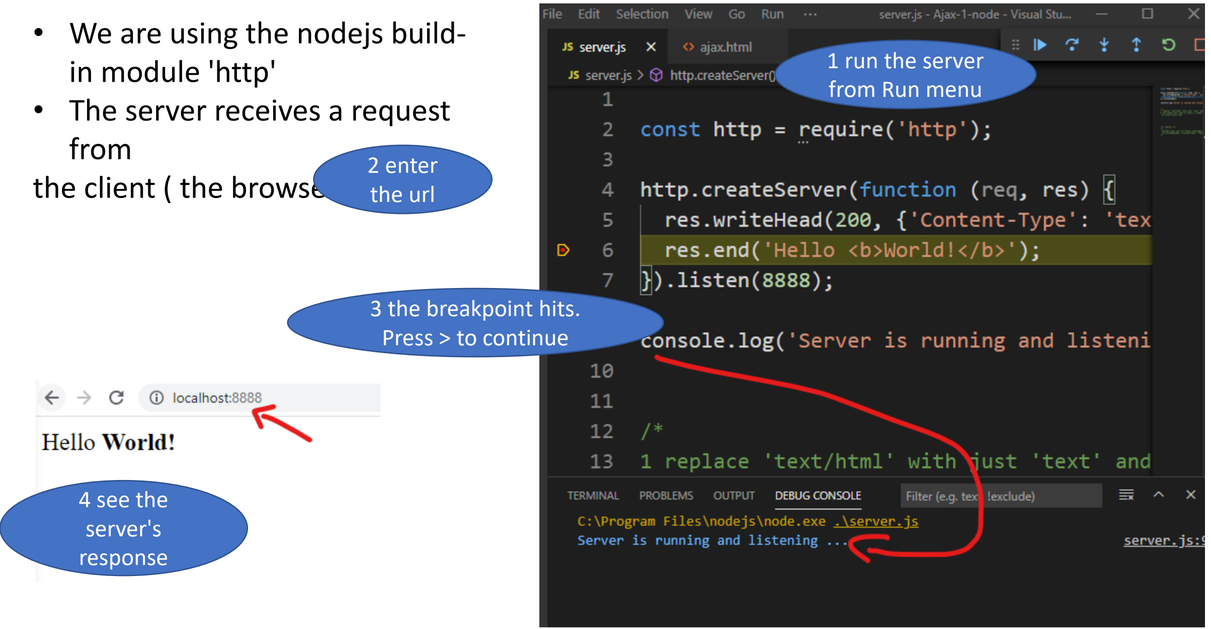


6

# 更多使用示例
# Node.js
# **内置模块**

# Example 4 : a simple http server

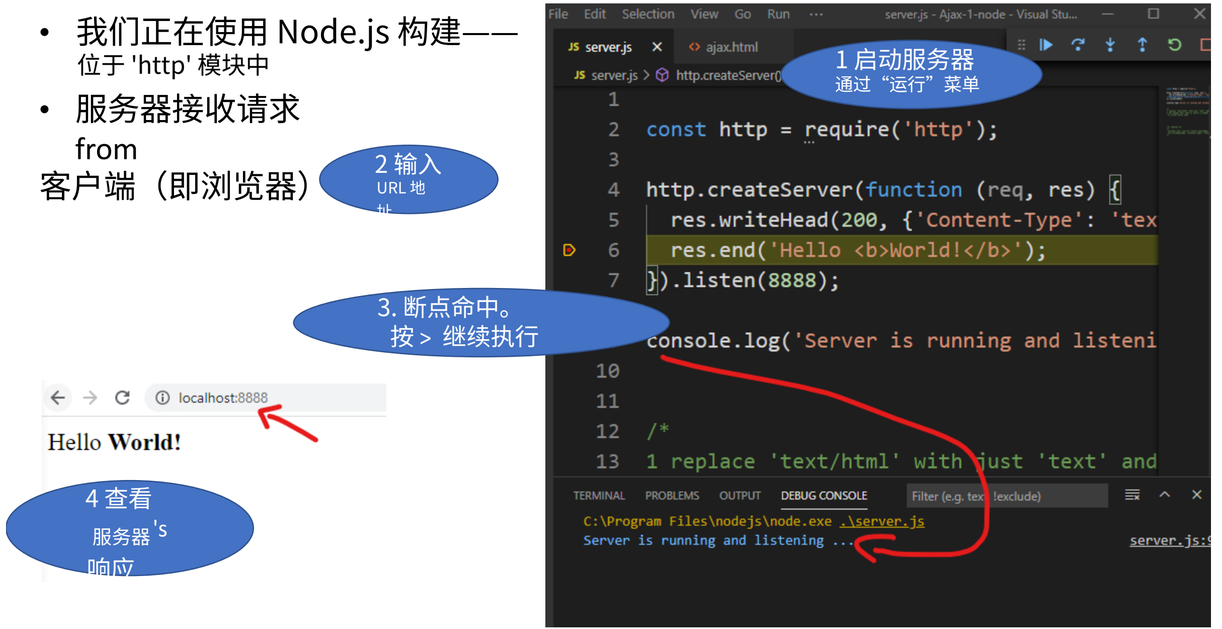
- We are using the nodejs build-in module 'http'
- The server receives a request from
the client ( the browse






# 示例 4：一个简单的 HTTP 服务器

- 我们正在使用 Node.js 构建——
位于 'http' 模块中
- 服务器接收请求
from
客户端（即浏览器）



# Example 4 : a simple http server …

```
const http = require('http');

http.createServer(function (req, res) {
  console.log("The server recived a request ");
  res.writeHead(200, { "Content-Type": "text/html", "Access-Control-Allow-Origin": "*" });
  res.end('Hello <b>World!</b>');
}).listen(8888);
```

- 1 at your local browser type http://localhost:8888/ and observe the result
- 2 replace 'text/html' with just 'text' and see what happens
- 3 now change this code to return a random number between 0 to 10
- to clinet every time
- 4 what does "Access-Control-Allow-Origin" do? do we need it now?



# 示例 4：一个简单的 HTTP 服务器……

```
const http = require('http');

http.createServer(function (req, res) {
  console.log("The server recived a request ");
  res.writeHead(200, { "Content-Type": "text/html", "Access-Control-Allow-Origin": "*" });
  res.end('Hello <b>World!</b>');
}).listen(8888);
```

- 1 在本地浏览器中输入 http://localhost:8888/，并观察结果
- 2 将 'text/html' 替换为仅 'text'，查看会发生什么
- 3 现在修改此代码，使其每次返回一个 0 到 10 之间的随机数
- 每次均发送给客户端
- 4. “Access-Control-Allow-Origin” 的作用是什么？我们现在需要它吗?

# Example 4, returning server's current date (put functions in module)

myModule.js

```
exports.area =
function () {
    return Date();
};
```

compare

```
let http = require('http');

let dt = require('./myModule');

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write("Now: " + dt.myDateTime());
  res.end();
}).listen(8888);
```



# 示例 4：返回值 server's current date (put functions in module)

myModule.js

```
exports.area =
函数 () {
    返回 Date();
};
```

compare

```
let http = require('http');

声明 dt = 调用 require('./myModule');

http.createServer(函数 (req, res) {
  res.writeHead(200, {'内容类型': 'text/html'});
  res.write("当前时间: " + dt.myDateTime());
  res.end();
}).监听(8888);
```

# In-class activity

- Replace the 'text/html' with just 'text' and observe the response that the client receives

server.js × ajax.html

server.js > ...

```
1
2  const http = require('http');
3
4  http.createServer(function (req, res) {
5    console.log("Hi");
6    res.writeHead(200, {'Content-Type': 'text/html'});
7    res.end('Hello <b>World!</b>');
8  }).listen(8888);
9
10 console.log('Server is running and listening ...');
11
12
```



# 课堂活动

- 将 'text/html' 替换为仅 'text'，并观察客户端接收到的响应。

server.js × ajax.html

server.js > ...

```
1
2  const http = require('http');
3
4  http.createServer(function (req, res) {
5    console.log("Hi");
6    res.writeHead(200, {'Content-Type': 'text/html'});
7    res.end('Hello <b>World!</b>');
8  }).listen(8888);
9
10 console.log('Server is running and listening ...');
11
12
```

Example 5, using 'url' built-in module to extract the query strings in the url



示例 5：使用 'url' 内置模块从 URL 中提取查询字符串。

37

## Using url built-in module to pars parameters in a string

```
let url = require('url');
let adr =
'http://localhost:8888/default.htm?name=John&age=23';
let q = url.parse(adr, true);

console.log(q.host); //returns 'localhost:8888'
console.log(q.pathname); //returns '/default.htm'
console.log(q.search); //returns '?name=John&age=23'

let qdata = q.query; //returns an object: { name: John,
age: 23 }
console.log(qdata.name); //returns 'John'
```

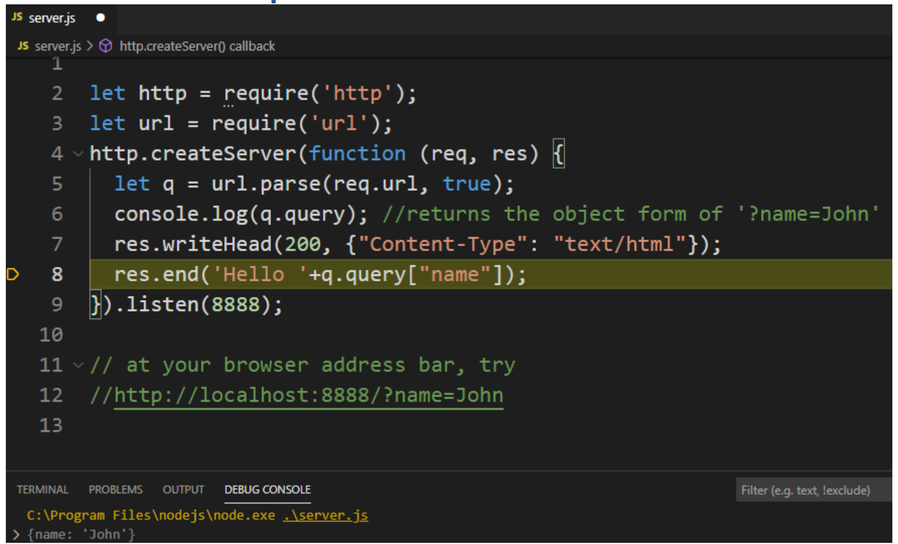

## Using url built-in module to pars parameters in a string

```
let url = require('url');
let adr =
'http://localhost:8888/default.htm?name=John&age=23';
let q = url.parse(adr, true);

console.log(q.host); //returns 'localhost:8888'
console.log(q.pathname); //returns '/default.htm'
console.log(q.search); //returns '?name=John&age=23'

let qdata = q.query; //returns an object: { name: John,
age: 23 }
console.log(qdata.name); //returns 'John'
```

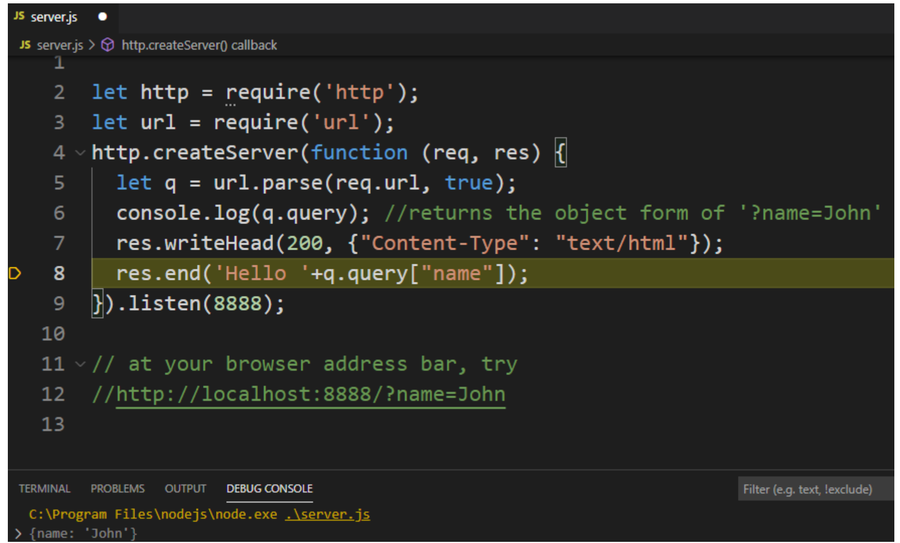
# Example 5: http server with parameters




- Note! I did not need `"Access-Control-Allow-Origin": "*"`
- Why?



# 示例 5：带参数的 HTTP 服务器




- 注意！我不需要 `"Access-Control-Allow-Origin": "*"`
- 为什么?

# 'fs', the file system built-in module

- We are going to be using another built-in module, 'fs', file system
- allows us to work with the file system on our computer or the server.
-



# 'fs'，即文件系统内置模块

- 我们将使用另一个内置模块——'fs'（文件系统模块）
- 该模块使我们能够在计算机或服务器上操作文件系统。
-

# Example 6: Creating a file server in node js

- Create a Node.js file that opens the requested file and returns the content of the file to the client. If the file does not exists, throw a 404 error!



# 示例 6：在 Node.js 中创建一个文件服务器

- 创建一个 Node.js 文件，用于打开客户端请求的文件，并将该文件的内容返回给客户端。如果文件不存在，则抛出 404 错误！

# Example 6: Creating a file server in node js

```
const http = require('http');
const url = require('url');
const fs = require('fs');
http.createServer(function (req, res) {
    const q = url.parse(req.url, true);
    const filename = "." + q.pathname;
    fs.readFile(filename, function (err, data) {
        if (err) {
            res.writeHead(404, { 'Content-Type': 'text/html' });
            return res.end(q.pathname + " 404 Not Found!");
        }
        res.writeHead(200, { 'Content-Type': 'text/html' });
        res.write(data);
        return res.end();
    });
}).listen(8888);
// at your browser address bar, try
//http://localhost:8888/file.txt
```

4xx are status message codes meaning client error.
That means client request was not valid, either due to authentication or the client asked for a file which did not exist
404: not found

HTTP Status Messages: https://www.w3schools.com/tags/ref_httpmessages.asp



# 示例 6：在 Node.js 中创建一个文件服务器

```
const http = require('http');
const url = require('url');
const fs = require('fs');
http.createServer(function (req, res) {
    const q = url.parse(req.url, true);
    const filename = "." + q.pathname;
    fs.readFile(filename, function (err, data) {
        if (err) {
            res.writeHead(404, { 'Content-Type': 'text/html' });
            return res.end(q.pathname + " 404 Not Found!");
        }
        res.writeHead(200, { 'Content-Type': 'text/html' });
        res.write(data);
        return res.end();
    });
}).listen(8888);
// at your browser address bar, try
//http://localhost:8888/file.txt
```

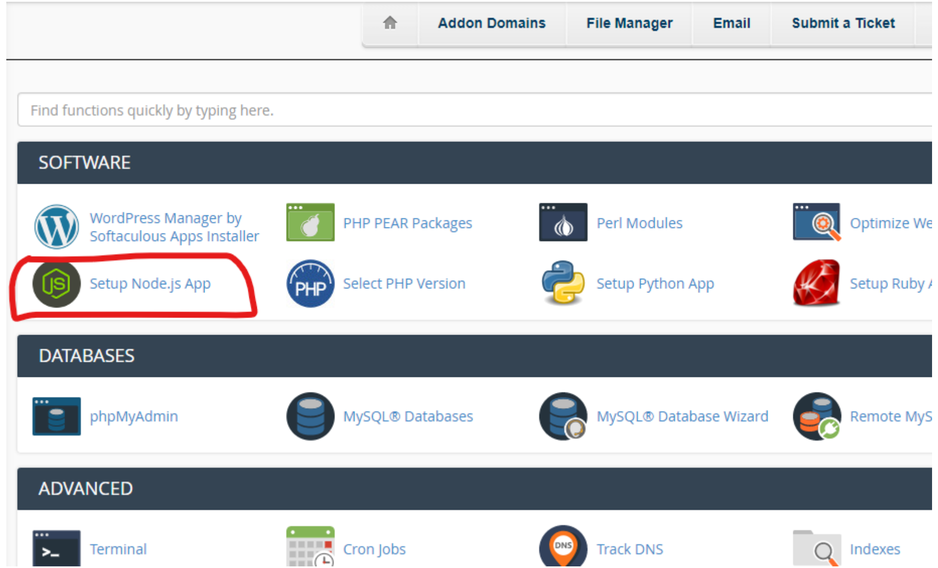
4xx 是表示客户端错误的状态码。这意味着客户端请求无效，原因可能是身份验证失败，也可能是客户端请求了一个不存在的文件。404：未找到

HTTP 状态消息：

https://www.w3schools.com/tags/ref_httpmessages.asp

# Setting up node js app on shared hosting with cPanel

- Refer to the lecture videos

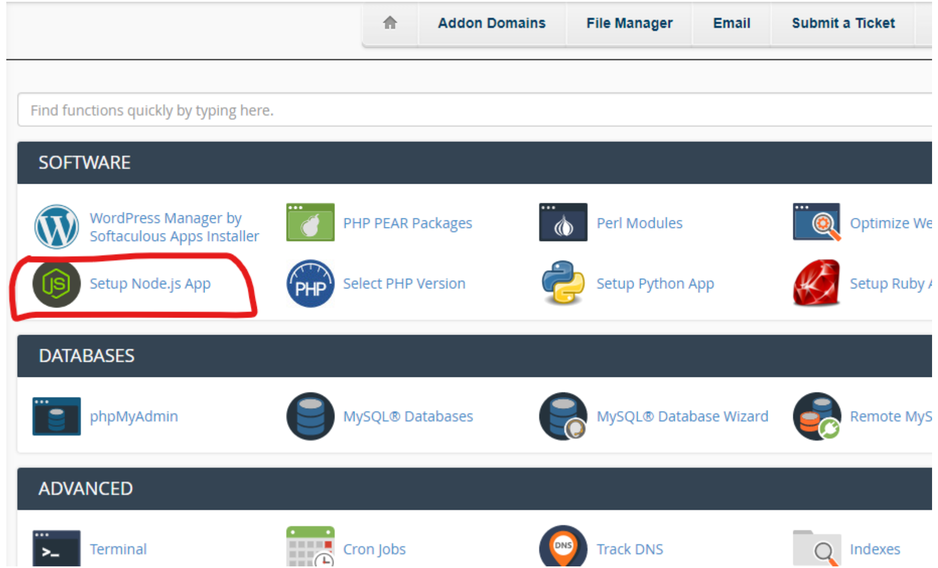



7



# 在支持 cPanel 的共享主机上部署 Node.js 应用

- 请参考教学视频




7

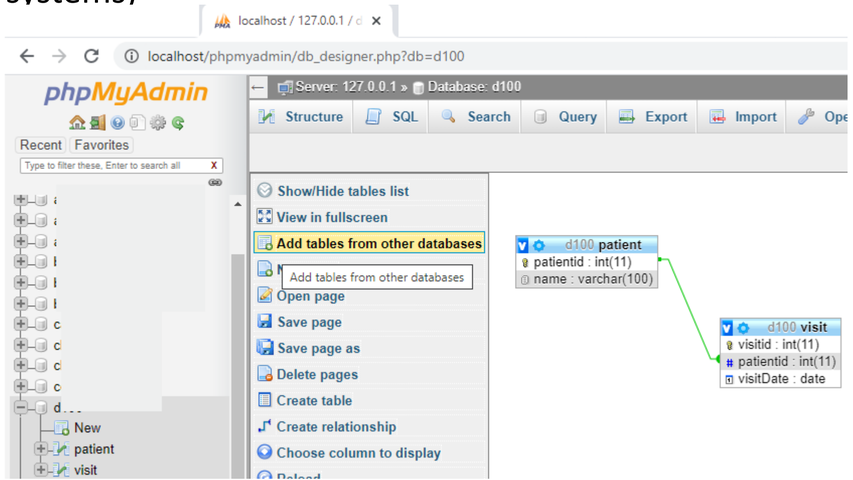
# Tips for future

8

# 未来小贴士

# Database server we use in this course

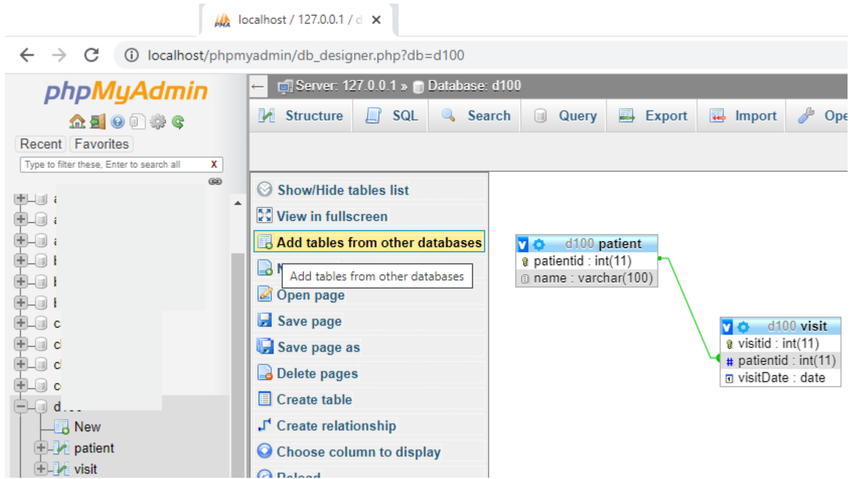
- We will use the free and open source relational DB administration tool
- Called phpMyAdmin which is browser based ( thus working on all operating systems)



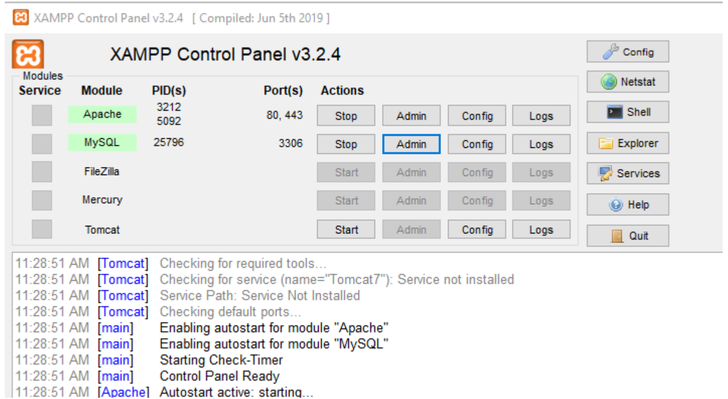



# 本课程所使用的数据库服务器

- 我们将使用一款免费且开源的关系型数据库管理工具
- 该工具名为 phpMyAdmin，基于浏览器运行（因此可在所有操作系统上使用）

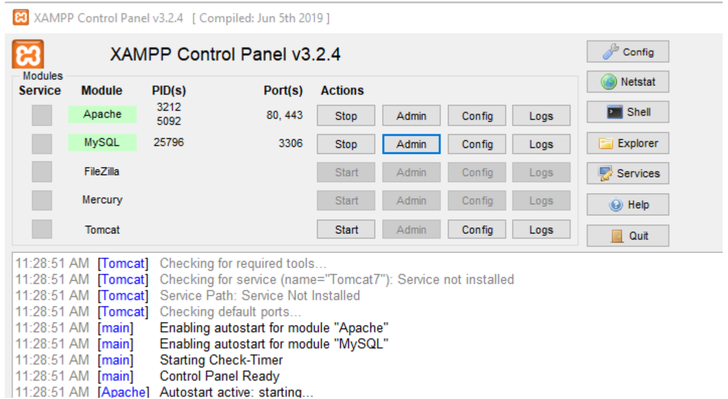


- You will learn how to use XAMPP to install phpMyAdmin on your machine
- ( in two weeks)






- 您将学习如何使用 XAMPP 在您的计算机上安装 phpMyAdmin
- （两周内）



# ESC6 tips of the week/ Arrow functions to replace old ones

- regular functions

```
setTimeout(function() {

}, timeout);
```

- arrow functions

```
setTimeout(() => {

}, timeout);
```

```
date = function() {
  return new Date();
}
```

```
date = () => {
  return new Date();
}
```

```
// one parameter
hello = (val) => "Hello " + val;

hello = val => "Hello " + val;

// two or more parameters
sum = (a,b) => a+b;
```

# ES6 每周小贴士：使用箭头函数替代传统函数

- 常规函数

```
setTimeout(function() {

}, timeout);
```

- 箭头函数

```
setTimeout(() => {

}, timeout);
```

```
date = function() {
  return new Date();
}
```

```
date = () => {
  return new Date();
}
```

```
// one parameter
hello = (val) => "Hello " + val;

hello = val => "Hello " + val;

// two or more parameters
sum = (a,b) => a+b;
```

## Resources

- https://www.w3schools.com/nodejs/nodejs_intro.asp
- https://en.wikipedia.org/wiki/Node.js
- On JS being asynchronous: https://youtu.be/6oP-c30s7co?t=6m8s

## 资源

- https://www.w3schools.com/nodejs/nodejs_intro.asp
- https://zh.wikipedia.org/wiki/Node.js
- 关于 JavaScript 的异步特性：https://youtu.be/6oP-c30s7co?t=6m8s