

前端全栈 后台开发

Node.js

Unit01

内容

上午	09:00 ~ 09:30	Node.js 概述
	09:30 ~ 10:20	
	10:30 ~ 11:20	Node.js 全局对象
	11:30 ~ 12:00	
下午	14:00 ~ 14:50	模块
	15:00 ~ 15:50	
	16:00 ~ 16:50	
	17:00 ~ 17:30	总结和答疑



Node.js 概述

Node.js 概述

Node.js 概述

什么是 Node.js

Node.js 与 JavaScript

安装 Node.js

运行 Node.js

使用 Node.js

Node.js 体系结构

Node.js 语法概览

数据类型

第一个 Node.js 应用

Node.js 的特点

Node.js 概述

什么是 Node.js

- Node.js 是一个基于Chrome JavaScript 运行时建立的平台，用于方便地搭建响应速度快、易于扩展的 WEB 应用
 - 轻量、高效
 - 非常适合数据密集型的实时应用
- Node.js是基于 ECMAScript 语言开发的服务器端语言
 - 可以编写独立的服务器端应用
 - 可以向客户端提供Web内容，但无需借助于任何Web服务器



Node.js 与 JavaScript

知识讲解

- 语法类似，功能不同
- Javascript由ECMAScript、DOM、BOM组成（Mozilla称为Core Javascript 和 Client Javascript）
 - 运行于浏览器中的语言，编写前端代码
- Node.js是使用C++编写的基于V8引擎的JS运行时环境，同时提供了很多基于ECMAScript/Core Javascript的扩展对象
 - 运行于服务器端的语言，编写服务器端应用
 - 需要下载并安装 Node.js 的解释器



安装 Node.js

知识讲解

- 官网：<https://nodejs.org>，下载对应版本
 - LTS: Long Term Support
 - Current
- 手册网址：<https://nodejs.org/api/>

Download for Windows (x64)

v4.6.0 LTS

Recommended For Most Users

v6.7.0 Current

Latest Features

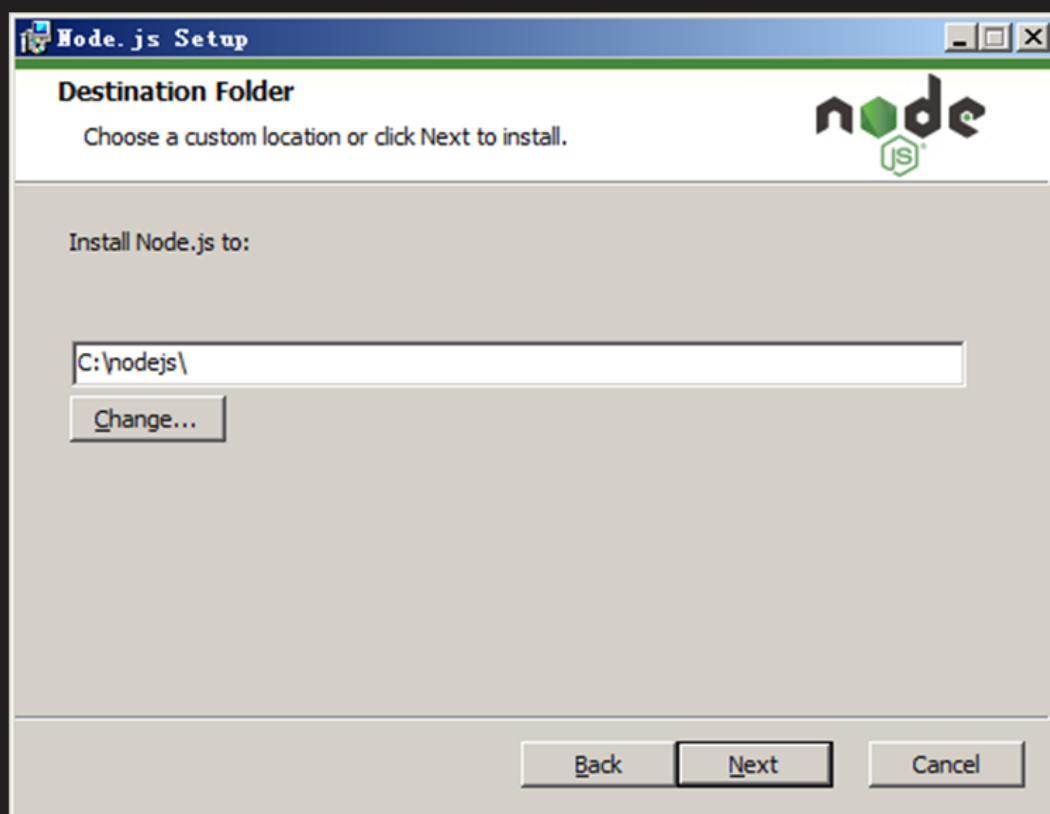
[Other Downloads](#) | [Changelog](#) | [API Docs](#)

[Other Downloads](#) | [Changelog](#) | [API Docs](#)



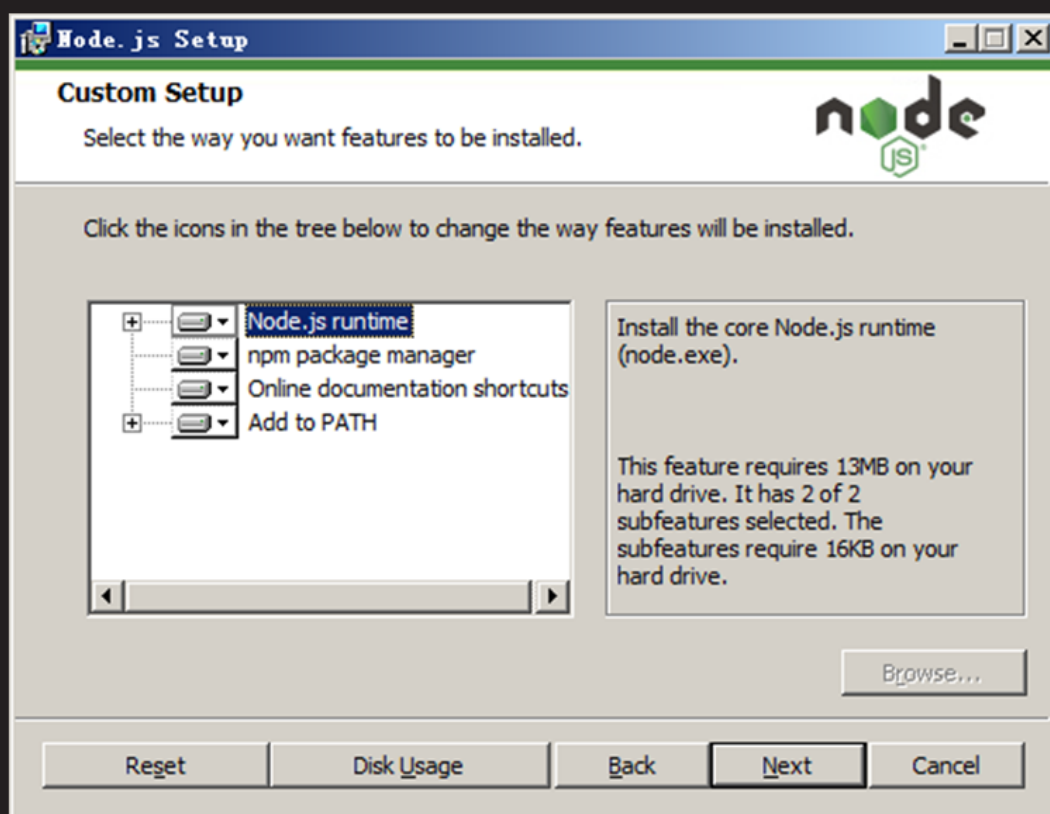
安装 Node.js (续1)

知识讲解



安装 Node.js (续2)

知识讲解



安装 Node.js (续3)

- 安装后测试

知识讲解



```
管理员: C:\Windows\system32\cmd.exe
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\Administrator>node -v
v4.6.0

C:\Users\Administrator>npm -v
2.15.9

C:\Users\Administrator>node
> console.log(100+99);
199
undefined
>
<To exit, press ^C again or type .exit>
>

C:\Users\Administrator>
```



运行 Node.js

- 交互模式 (REPL 模式)
 - Read-Evaluate-Print-Loop
 - 读取用户输入，执行运算，输出执行结果，继续下一次循环
 - 交互模式下，Node.js自带的模块无需引入
- 脚本模式
 - 将所有语句编写在独立的脚本文件中，一次性执行
 - 脚本模式下，除了全局对象及其相关成员外，所有其它模块中声明的对象和方法必须使用 require() 引入

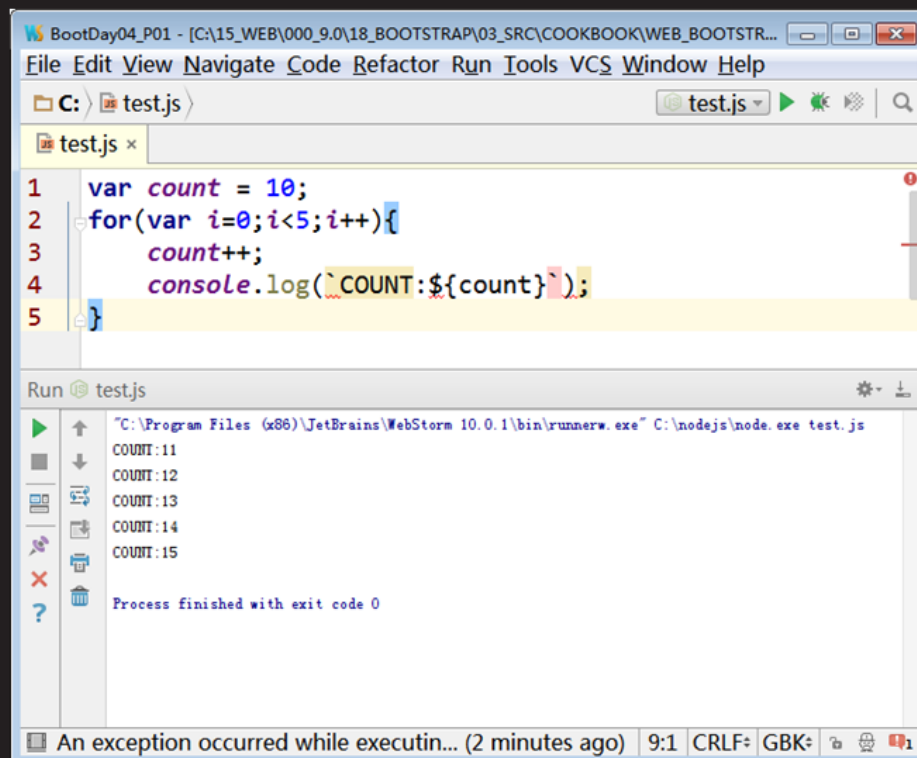
知识讲解



运行 Node.js (续1)

- WebStorm IDE

知识讲解



The screenshot shows the WebStorm IDE interface. The top toolbar includes menus like File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, and Help. Below the toolbar is a file explorer showing the current file is `C:\test.js`. The main editor displays the following JavaScript code:

```
1 var count = 10;
2 for(var i=0;i<5;i++){
3     count++;
4     console.log(`COUNT:${count}`);
5 }
```

Below the editor is a 'Run' window for `test.js`. It shows the command executed: `"C:\Program Files (x86)\JetBrains\WebStorm 10.0.1\bin\runnerw.exe" C:\nodejs\node.exe test.js`. The output of the program is displayed as:

```
COUNT:11
COUNT:12
COUNT:13
COUNT:14
COUNT:15
```

At the bottom of the Run window, it states: `Process finished with exit code 0`. The status bar at the very bottom indicates: `An exception occurred while executin... (2 minutes ago) 9:1 CRLF GBK`.



运行 Node.js

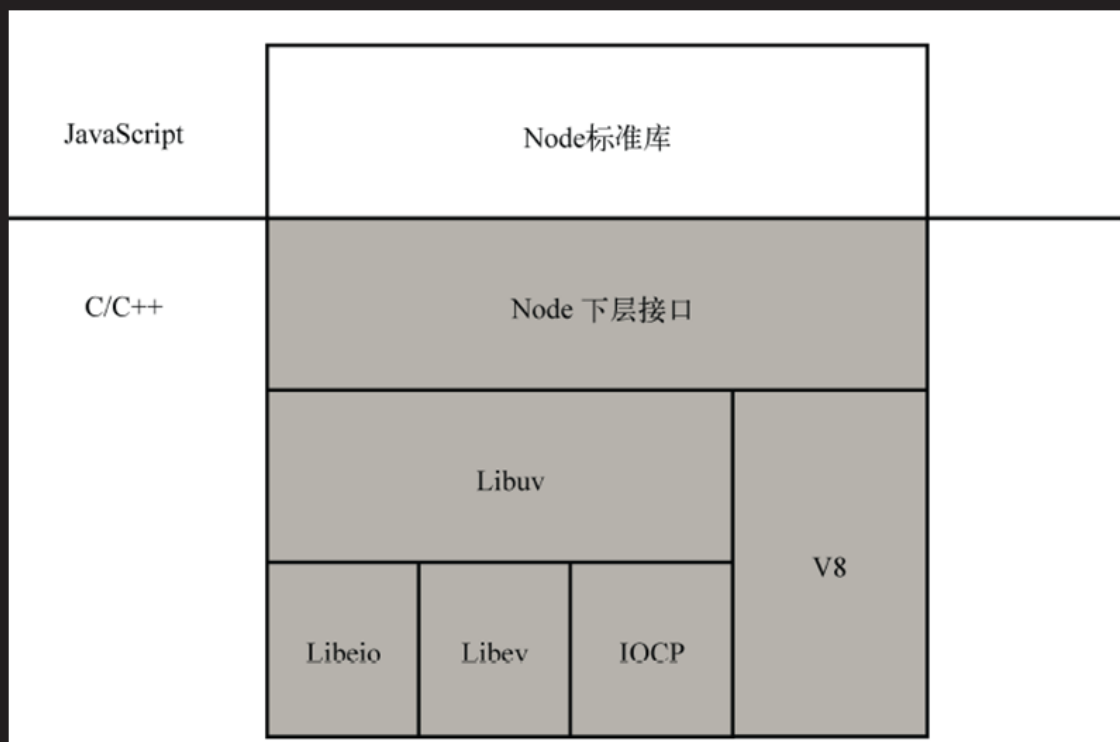
- 安装 Node.js
- 命令行方式下，两种方式运行 Node.js
 - 测试简单语句
 - 测试对象
- WebStorm IDE 方式运行
 - 运行
 - 调试

课堂练习



使用 Node.js

Node.js 体系结构



Node.js 体系结构 (续1)

知识讲解

- ES 对象
 - Global、String、Number、Boolean、Date、Math、Array、Error、Function、Object、RegExp
- 扩展对象
 - 以模块划分
 - File、HTTP。。。。
- 支持 ES6 新特性



Node.js 语法概览

知识讲解

- Node.js 从4.0开始，支持ES6的大多数新特性，例如：
classes、collections、arrow functions、block scoping、template strings 等。
 - 数据类型
 - 声明变量和常量
 - 运算符
 - 逻辑结构
 - 函数作用域和闭包
 - 对象和原型
 - 对象分类



数据类型

知识讲解

- 原始类型 (Primitive Type)
 - string、number、boolean、null、undefined
- 引用类型 (Reference Type)
 - ES核心对象：String、Boolean、Number、Date、Math、RegExp、Error ...
 - Node.js对象：Buffer、ReadStream、ClientRequest...
 - 自定义对象

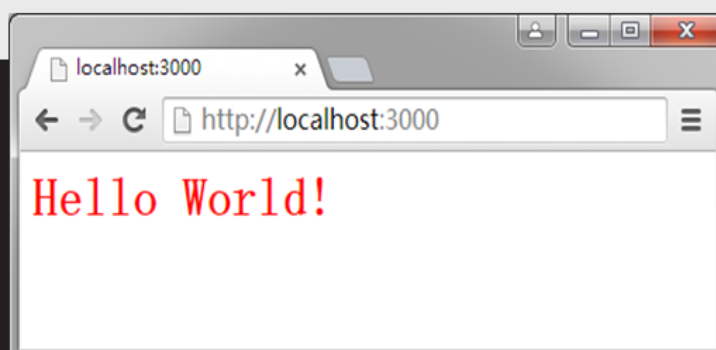


第一个 Node.js 应用

- 引入模块，并使用模块所提供的方法

```
var http = require('http');
var serv = http.createServer(
  function(req,res){
    res.writeHead(200,{ 'content-type': 'text/html' });
    res.end("<h1 style='color:red'>Hello World!</h1>");
  });
serv.listen("3000");
```

知识讲解



第一个 Node.js 应用

- 创建并测试 Node.js 应用

课堂练习



Node.js 的特点

- 简单，避免过度设计
- 单线程逻辑处理
- 非阻塞的异步I/O处理
- 事件驱动编程
- 无锁机制，不会产生死锁
- 支持数万个并发连接

知识讲解



Node.js 的特点（续1）

知识讲解

- 适合搭建响应速度快、易于扩展的网络应用
 - 命令行工具
 - 带有GUI界面的本地应用程序
 - 交互式终端程序
 - 单元测试工具
 - 基于社交网络的大规模Web应用
 - Web Socket服务器
 - TCP/UDP套接字程序
 - 客户端Javascript编译器



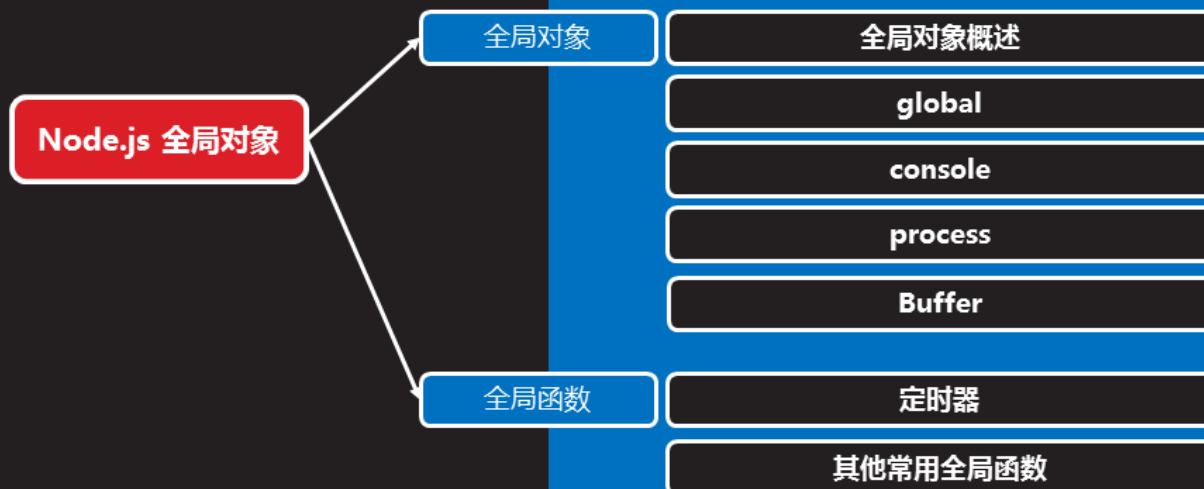
Node.js 的特点（续2）

知识讲解

- Node.js不适合CPU密集型应用
 - 深层次的嵌套和递归
 - 复杂加密和解密算法
 - 高可靠性运算
 - 严格内存管理
 - 数据挖掘和数据分析



Node.js 全局对象



全局对象

全局对象概述

知识讲解

- 全局对象（Global Object），它及其所有属性都可以在程序的任何地方访问，即全局变量
- 嵌入在浏览器中的V8引擎，支持ES原生对象、BOM和DOM对象，全局对象为window
 - 声明全局作用域的变量和函数默认为window对象的成员——全局对象污染！
- Node.js 中的全局对象是 global，所有全局变量（除了 global 本身以外）都是 global 对象的属性
 - 独立的Node.js环境中，不支持BOM和DOM对象



global

知识讲解

- global 作为全局变量的宿主
 - 声明的全局变量和函数在交互模式下是global对象的成员——全局对象污染
 - 脚本模式下不是global对象的成员——避免了全局对象污染



console

知识讲解

- Node.js 中的 console 模块提供了一种控制台调试输出机制，类似于 Chrome 浏览器中 console 对象的功能
 - 用于向标准输出流（stdout）或标准错误流（stderr）输出字符
- 常用方法
 - log
 - info
 - error
 - warn
 - trace

```
var data = { id: 3, count: 36 };
```

```
console.log( `COUNT IS: %d`, data.count );  
console.info( `COUNT IS: ${data.count}` );  
console.warn( `COUNT IS: ${data.count}` );  
console.error( `COUNT IS: ${data.count}` );  
console.trace('Some Msg');
```



console (续1)

知识讲解

- 常用方法
 - dir
 - assert
 - time、timeEnd

```
var data = { id: 3, count: 36, list:[ ] };
```

```
console.dir(data);  
console.assert(data.count > 0, 'COUNT IS ZERO');  
console.assert(data.list.length > 0, 'NO DATA AVAILABLE');
```

```
console.time( 'LONG-LOOP' );  
for( var i=0; i<1000; i++ ){  
    ;  
}
```

```
console.timeEnd( 'LONG-LOOP' );
```



console (续2)

- 可以向任意指定输出流中执行输入
- 常用类
 - Console()

知识讲解

```
var out=fs.createWriteStream( 'out.log' );
var err=fs.createWriteStream( 'err.log' );

var logger=new console.Console(out,err);
logger.log( '输出到out文件中的日志' )
logger.err( '输出到err文件中的错误日志' )
```



process

- process 是一个全局变量，即 global 对象的属性
- 表示执行当前解释器运行所在的进程对象
 - 可以使用该对象获取当前操作系统及运行时信息，并操作脚本所在执行进程
- 当操作系统启动 Node.js 解释器执行一个脚本文件时（该脚本文件称为“主模块” - mainModule），会将必需的文件从文件系统调入内容，分配必需的内存空间，执行其中的代码，从而创建一个执行进程
 - 全局对象 global.process 就是这个进程的代码表示

知识讲解




```
console.log( process.arch )           //获取CPU架构类型
console.log( process.platform )       //获取操作系统类型
console.log( process.env )            //获取操作系统环境变量

console.log( process.cwd() )          //获取当前所在工作目录
console.log( process.execPath )       //获取解释器所在目录
console.log( process.versions )       //获取Node.js版本信息
console.log( process.uptime() )       //获取Node.js解释器运行时间(s)
console.log( process.memoryUsage() )  //获取内存信息

console.log( process.pid )            //获取进程ID号
process.kill( pid )                  //向指定进程ID号发送退出信号
```



Buffer

- 一块专用于存储数据的内存区域
- Buffer对象的实例，可以通过读取文件获得，也可以直接构造

```
//创建一个长度为10字节的缓冲区
var buf1 = new Buffer(32);
//创建一个长度为3字节的缓冲区
var buf2 = new Buffer([65,66,67]);
//创建一个长度为4字节的缓冲区
var buf3 = new Buffer('ABCD');
```



Buffer (续1)

```
//向Buffer中写入内容
var s1 = "AA";
var s2 = "BB ";
var b2 = new Buffer(30);
b2.write(s1);//默认写到最开头
b2.write(s2,2);//第二个参数指定写入的位置

//将Buffer内容转换为特定编码的string
console.log(buf3.toString('utf8'));

//测试是否支持某种编码方式
console.log(Buffer.isEncoding('utf8'));
```



使用全局对象

- 使用 global
- 使用 console
- 使用 process



全局函数

定时器

- Node.js 提供了四种形式的定时器
 - `global.setTimeout()` : 一次性定时器
 - `global.setInterval()` : 周期性定时器
 - `process.nextTick()` : 本次事件循环结束时立即执行的定时器
 - `global.setImmediate()` : 下次事件循环立即执行的定时器

使用全局函数

- 使用定时器

课堂练习



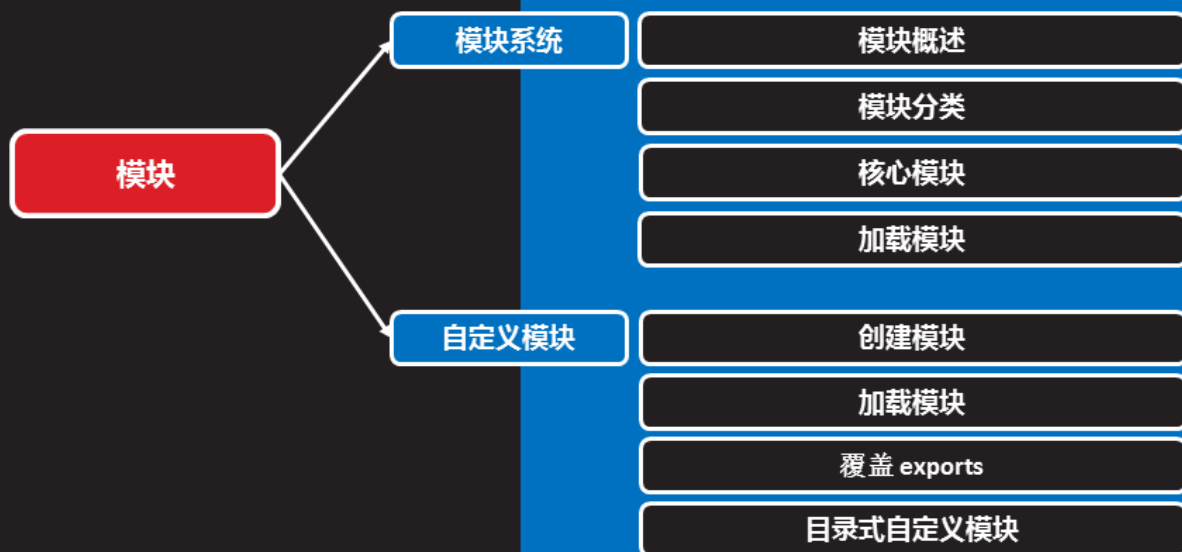
其他常用全局函数

知识讲解

函数名	说明
decodeURI()	解码一个编码的 URI
decodeURIComponent()	解码一个编码的 URI 组件
encodeURI()	把字符串编码为 URI
encodeURIComponent()	把字符串编码为 URI 组件
escape()	对字符串进行编码
unescape()	对由 escape() 编码的字符串进行解码
parseInt()	解析一个字符串并返回一个整数
parseFloat()	解析一个字符串并返回一个浮点数
isNaN()	检查是否为 NaN
isFinite()	检查是否为有穷大的数字
eval()	计算指定字符串，将其作为语句来执行



模块



模块系统

模块概述

知识讲解

- Node.js使用“Module（模块）”来区分不同的功能对象
 - Node.js中每一个 .js 文件对应一个模块，包含相应的功能和函数
 - 模块内声明的变量或函数的作用域是“模块作用域”——默认只能在当前JS文件（即当前模块）中使用，而不是全局作用域
 - 每个模块可以导出（exports）自己内部的对象供其它模块使用，也可以引入（require）并使用其它模块中导出的对象
 - Node.js启动时运行的第一个模块称为“主模块”（main module）



模块分类

知识讲解

- 核心模块
 - node.js 提供，被编译进二进制执行文件，可以被解释器直接使用，加载速度最快
- 第三方模块
 - 基于Node.js核心模块，很多组织和个人提供了大量的第三方扩展应用模块 <https://www.npmjs.com/>
 - 常用的第三方模块：express、less、grunt
- 自定义模块
 - 文件式
 - 目录式



核心模块

- 也叫绝对模块，指 node.js 的内置模块
 - 其内部 node_modules 查找到的模块

知识讲解

模块名	说明
global	全局对象模块
console	控制台模块
util	提供常用函数的集合，用于弥补核心JS的功能过于精简的不足
events	实现了Node.js的事件驱动型的异步调用
fs	文件系统I/O操作模块
http	提供基于HTTP协议的请求和响应
net	提供基于Socket的网络连接
dns	DNS解析服务模块
crypto	加密和解密功能模块



加载模块

- require() 函数用于加载一个模块，其可用的参数有如下形式
 - 原生模块名，如 http、fs、url等
 - 文件模块的相对路径，如 ./mod1、../mod2等
 - 文件模块的绝对路径，如 /path/to/module/mod3
 - 非原生模块的文件模块名，如 mod4

知识讲解



加载模块（续1）

- 使用 require 加载并调用模块

知识讲解

```
//引入http模块
var http = require('http');
var serv = http.createServer(...);
serv.listen("3000");

//引入querystring模块
var qs = require('querystring');
//解析一个HTTP请求消息中的查询字符串
var str = 'uname=tom&upwd=123&age=20';
var obj = qs.parse(str);
console.log(obj);
```



自定义模块

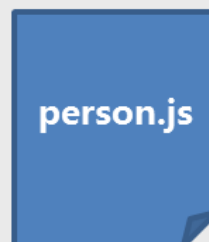
创建模块

- 文件式自定义模块
 - 创建 js 文件，包含功能，并使用 exports 公开接口
 - exports：对module.exports的引用，为一个对象，用于暴露模块的 API，供 require 调用

知识讲解

```
//功能代码
console.log("cod in person.js");

//公开接口
var data = "mary";
exports.name = data;
exports.work = function(){
    console.log('i am ' + data);
};
```



创建模块（续1）

- Node.js中的每个.js文件都自成一个模块，有自己专属的成员属性和方法——“模块作用域”变量
 - 模块文件中声明的变量和函数也都属于“模块作用域”
- Node.js在编译模块文件时会对其首尾进行如下的包装：

知识讲解

```
(function (exports, require, module, __filename, __dirname) {
    module.exports = {};
    exports = module.exports ;

    //模块文件中原有的全部内容

    return module.exports ;
});
```



加载模块

- 其他 js 文件中调用
 - 使用 require 指向一个相对工作目录中的 js 文件

知识讲解

```
//调用自定义模块：注意路径
//此时，此js文件需要和 person.js 位于同一路径下
var p = require('./person');

console.log(p.name);
p.work();
```



覆盖 exports

- 重写 module.exports，实现对象的封装
 - require 调用时会得到一个对象

知识讲解

```
module.exports = Person;
```

```
//对象的构造
```

```
function Person(name,age){
  this.name = name;
  this.age = age;
```

```
  this.intro = function(){
    console.log(`i am ${this.name},${this.age} years old.`);
  };
}
```


person.js

覆盖 exports (续1)

- exports 本身是一个普通的空对象，专门用来声明接口
 - 但是不能通过对 exports 直接赋值，而是赋给 module.exports

知识讲解

```
//调用  
var Person = require('./person');  
var p = new Person('mary',20);  
p.intro();
```



自定义模块—文件式

- 创建自定义模块
- 调用自定义模块
 - 简单调用
 - 覆盖exports

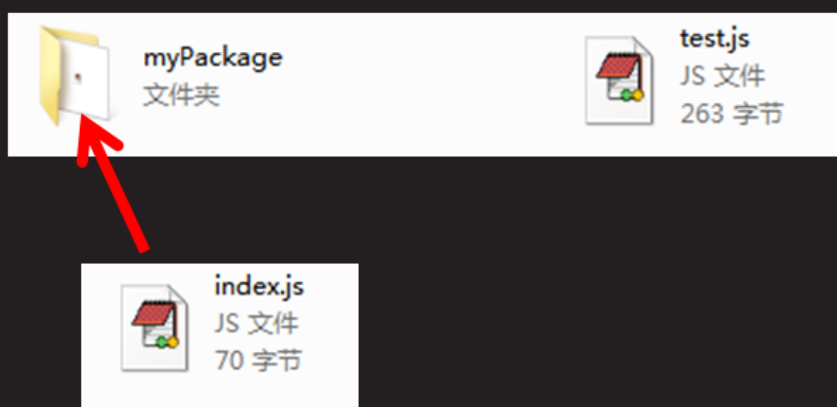
课堂练习



目录式自定义模块

- 模块与文件一一对应
 - 可以是单个文件：js代码文件，或者二进制代码文件
 - 可以是文件夹—文件夹封装为模块，即为 **包**
- 自定义文件夹
 - 创建 index.js 文件

知识讲解



目录式自定义模块（续1）

- 为 index.js 文件添加代码，并在文件夹外的 js 文件中调用

知识讲解

```
//定义模块中的功能
exports.hello = function(){
    console.log("hello in index.js");
};
```

myPackage/
index.js

```
//调用目录中的模块
var p = require('./myPackage');
p.hello();
```

test.js



自定义模块—目录式

课堂练习

- 创建文件夹
- 创建 index.js 文件，添加功能
- 调用目录中的模块
 - 简单调用
 - 复杂调用



总结和答疑
