Technology Tarena 达内科技

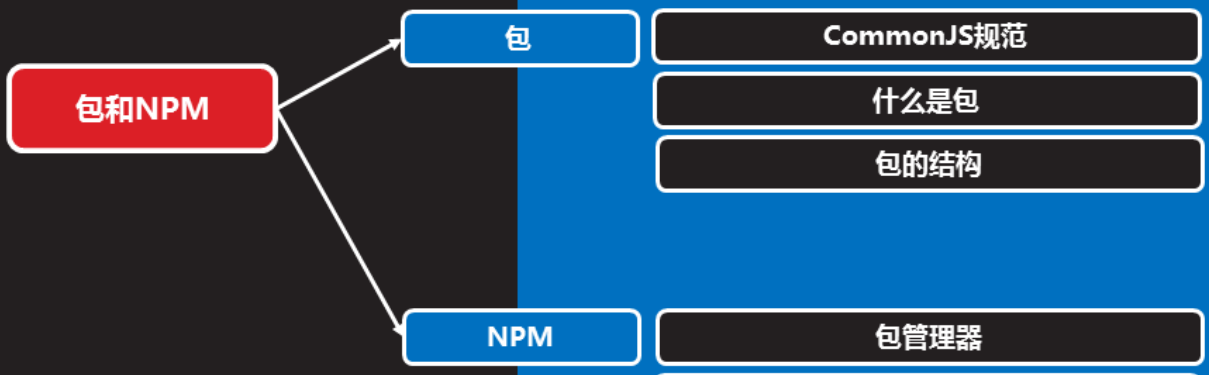
前端全栈 后台开发

Node.js Unit02

内容

上午	09:00 ~ 09:30	作业讲解和回顾
	09:30 ~ 10:20	包和NPM
	10:30 ~ 11:20	
	11:30 ~ 12:00	Node.js 的核心模块
下午	14:00 ~ 14:50	
	15:00 ~ 15:50	
	16:00 ~ 16:50	
	17:00 ~ 17:30	总结和答疑

包和NPM



Technology Tarena 达内科技

包

CommonJS规范

Technology Tarena 达内科技

知识讲解

- JavaScript最初的设计目标是运行于客户端控制页面的交互与行为，对于实现服务器端应用而言，有着不足
  - 没有模块系统：没有原生的支持密闭作用域或依赖管理
  - 没有标准库：除了一些核心对象外，没有文件系统的API，没有IO流API等
  - 没有标准接口：没有如Web Server或者数据库的统一接口
  - 没有包管理系统：不能自动加载和安装依赖
- 由于缺少统一的标准和规范，上述问题限制了服务器端JS更广泛更大规模的的应用

CommonJS规范（续1）

知识讲解

- CommonJS（http://www.commonjs.org/）不是一门语言，而是为JS在更广范围的应用而定义的 API 标准和规范，最终提供类似于Python、Ruby和Java语言一样丰富的功能
  - 只要遵守了这些规范，JS编写的应用就可以在任何兼容标准的解释器和主机上运行，从而使得JS可以应用于更加广泛的领域：服务器端应用、命令行应用、桌面GUI应用、混合应用
- Node.js实现了CommonJS规范，实现了其定义的常用API ，以及模块和包的编写、使用和维护规范

什么是包

知识讲解

- 包（ package ），是在模块基础上更深一步的抽象
  - 类似于C/C++的函数库或者Java/.Net的类库
  - 将独立的功能封装起来，用于发布、更新、依赖管理和版本控制
- Node.js 根据 CommonJS 规范实现了包机制
  - 包是一个目录
  - 目录中包含 js 文件
  - 一个 JSON 格式的包说明文件
- 目录式自定义模块，即为包

包的结构

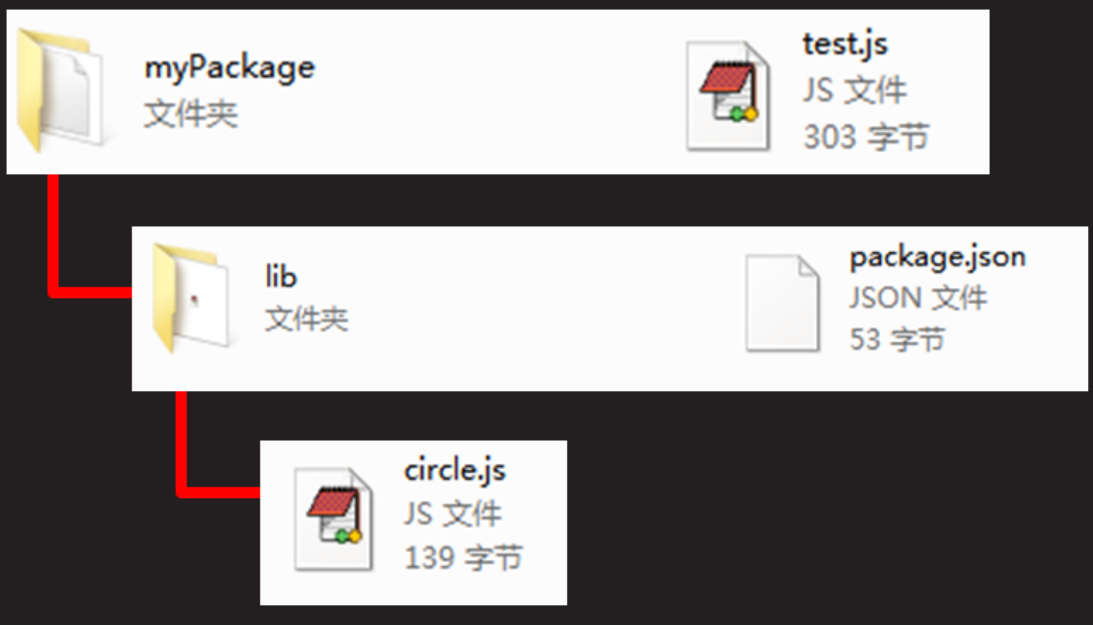
知识讲解

- 根据 CommonJS 包规范，一个包应该具有如下结构：
  - 一个package.json文件应该存在于包顶级目录下
  - 二进制文件应该包含在bin目录下
  - JavaScript代码应该包含在lib目录下
  - 文档应该在doc目录下
  - 单元测试应该在test目录下
- Node.js 中，require( )函数可以引入文件模块之外，还可以引入符合上述规范的包

包的结构（续1）

知识讲解

- 在调用包时，会首先检查包中 package.json 文件的main字段
  - 如果不存在，则尝试寻找 index.js 作为包的接口



包的结构（续2）

知识讲解

package.json

```
{
        "version": "0.0.1",
        "main": "./lib/circle"
}
```

circle.js

```
module.exports = Circle;

function Circle(radius){
    this.radius = radius;
    this.area = (Math.PI * radius * radius).toFixed(4);
}
```

test.js

```
var Circle = require('./myPackage');
var c = new Circle(10);
console.log(c.area);
```

创建并使用包

课堂练习

- 创建包
- 调用

Technology Tarena 达内科技

NPM

Technology Tarena 达内科技

包管理器

知识讲解

- 包管理器（ Node Package Manager ，简称npm）是Node.js 提供的包管理工具，用于下载、安装、升级和删除包，或者发布并维护包
  - 根据 CommonJS 规范，Node.js 已经提供了几十个核心模块，并集成到 node中
  - 也可以使用第三方包
- Node.js 的安装文件中，已经集成了 NPM 包管理工具

包管理器（续1）

知识讲解

- reqire( ) 函数引入包时，参数是包的名称；同时要求包对应的目录处于 node_modules 目录下，其根目录下存在 package.json 文件，其内容格式满足CommonJS Packages/1.1 规范
- 使用 npm install 命令可以下载并安装一个包
  - 安装包将保存在 ./node_modules 下（运行 npm 命令时所在的目录）
  - 可以通过 require() 来引入本地安装的包

```
npm  install  包名
```

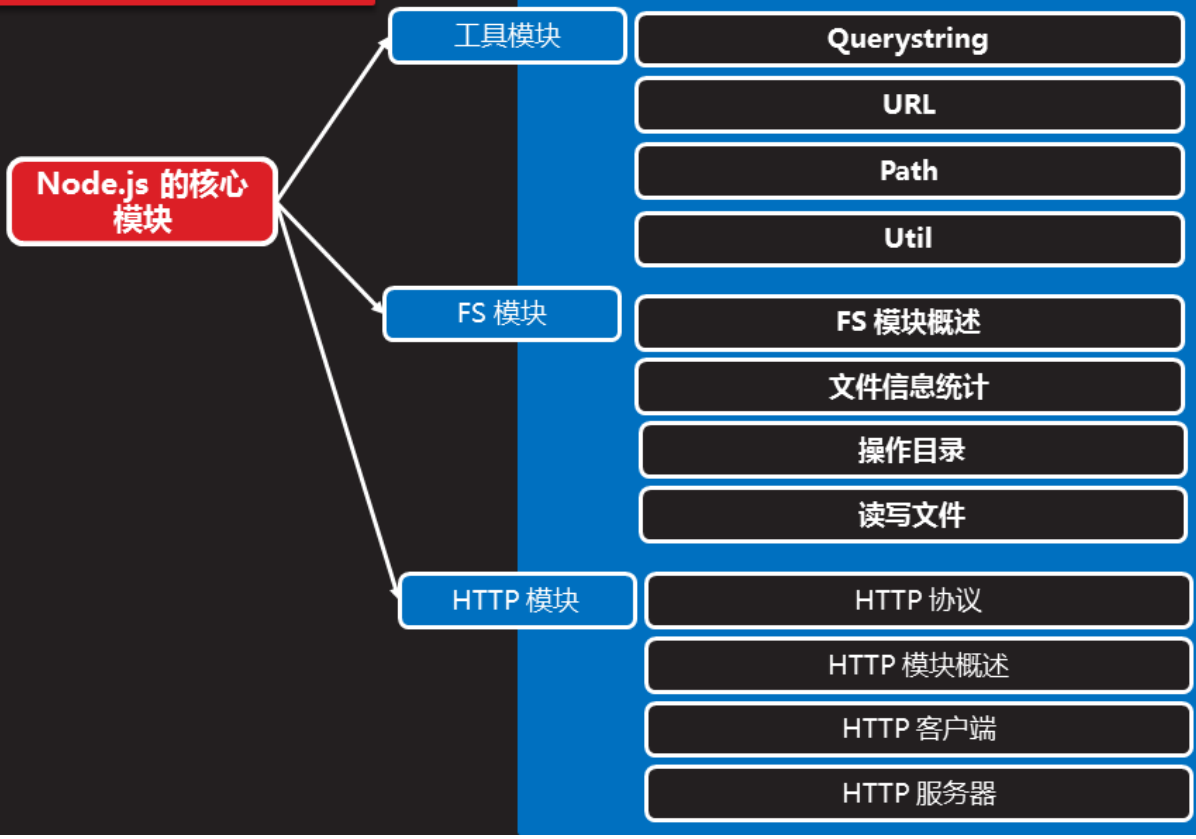
常用包

知识讲解

- 常用的包

包名	说明
colors	命令行彩色输出
event-stream	Stream流操作工具
mocha	单元测试
mysql	连接MySQL
request	HTTP客户端
restify	REST API搭建
socket.io	WebSocket实时通信
xml2js	XML转换工具

Node.js 的核心模块



Technology Tarena 达内科技

工具模块

QueryString

- Query String 模块提供了处理 URL 中的查询字符串部分的相关操作

知识讲解

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

var qs = 'cid=%E4%BD%93%E8%82%B2&size=10&pno=2';

//从查询字符串中解析出数据对象
console.log( querystring.parse(qs) );

//将数据对象转换为查询字符串
var data = { cid: '体育', size: '10', pno: '2' };
console.log( querystring.stringify(data) );
```

URL

- URL 模块提供了处理 URL 中不同部分的相关操作

知识讲解

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

var requestUrl =
    'http://tom:123@tedu.cn:8080/news/n1?pno=2#section3';
//解析出URL中的各组成部分
console.log( url.parse(requestUrl) );
//将查询字符串解析为对象
console.log( url.parse(requestUrl, true) );

var urlObj = { protocol: 'http:', port: '8080',
          hostname:   'www.tedu.cn',  query: { pno: '2' },
          pathname: '/news/n1' };
console.log( url.format(urlObj) );
```

Path

- Path 模块提供了对文件路径进行操作的相关方法
  - 进行路径中字符串的相关转换，与文件系统本身无关

知识讲解

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

//解析路径字符串
console.log( path.parse('c:/user/local/img/1.jpg') );

//将路径对象格式化为字符串
var obj = {  dir: 'c:/user/local/img',  base: '1.jpg' };
console.log( path.format(obj) );

//根据基础路径解析出一个目标路径的绝对路径
console.log( path.resolve('htdocs/css','../img/news') );
//根据基础路径，获取目标路径与其的相对关系
console.log( path.relative('htdocs/css','htdocs/img/news') );
```

Util

- Util 模块提供了若干工具方法，供 Node.js 其它模块和应用开发者使用
- Util.inspect 方法

知识讲解

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

//返回一个对象的字符串形式表示
var data = {name:'Cola', price:2.5, isOnsale: false};
console.log( util.inspect(data) );
```

Util（续1）

Tarena Technology 达内科技

- Util.inherits 方法

知识讲解

```
var util = require('util');

function Base(){
    this.name = 'base';
}
Base.prototype.age = 10;

function Sub(){
    this.name = 'sub';
}
util.inherits(Sub,Base);
var s = new Sub();
console.log(s.name + ',' + s.age);
```

测试常用工具模块

Tarena Technology 达内科技

- 测试常用工具模块

课堂练习

Technology Tarena 达内科技

FS 模块

Technology Tarena 达内科技

FS 模块概述

知识讲解

- fs 模块是 Node.js 核心模块之一，提供了文件的读写、更名、删除、遍历目录等操作
- 操作有同步和异步两种形式
  - 在 I/O 操作密集的应用中，推荐使用异步调用方式，以避免整个处理进程的阻塞
- 常用class
  - fs.Stats：文件或目录的统计信息描述对象
  - fs.ReadStream：stream.Readable 接口的实现对象
  - fs.WriteStream：stream.Writable接口的实现对象
  - fs.FSWatcher：可用于监视文件修改的文件监视器对象

FS 模块概述（续1）

知识讲解

- 常用方法
  - fs.mkdir( )：创建目录
  - fs.rmdir( )：删除目录
  - fs.readFile( )：读取文件内容
  - fs.writeFile( )：向文件中写出内容
  - fs.appendFile( )：向文件中追加内容
  - fs.unlink( )：删除文件
  - fs.rename( )：重命名文件

文件信息统计

知识讲解

- fs.stat() 和 fs.statSync() 方法用于返回一个文件或目录的统计信息对象（fs.Stats 类型）
- fs.Stats 对象方法可用于检查文件的物理特性，比如
  - stats.isFile()：是否为文件
  - stats.isDirectory()：是否为目录

```
var fs = require('fs');
const path = './test.js';
fs.stat(path,(err, stats) => {
    if(err){
        console.log('no file');  //指定文件不存在
    }else {
        console.log(stats);
    }
});
```

操作目录

- 操作目录的常用方法
  - fs.mkdir(path[, mode], callback)：创建指定目录
  - fs.rmdir(path, callback)：删除指定目录
  - fs.readdir(path[, options], callback)：读取目录下内容

知识讲解

```
var fs = require('fs');
const path = './mypackage';
fs.stat(path, (err, stats) => {
    if(err){
        fs.mkdir( path );
    }else {
        fs.readdir( path, function(err,list){
            console.log(list);
        })
    }
});
```

读写文件

- 对于数据量不是很大的文件，可以一次性的读写其中的全部内容
  - fs.readFile(file[, options], callback)：读取文件内容

知识讲解

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

const src = './htdocs/index.html';
fs.readFile( src, (err,data)=>{
        if (err) throw err;
        //读取到的数据保存在一个Buffer实例中
        console.log( data );
        console.log( data.toString() );
})
```

读写文件（续1）

知识讲解

- 写文件时，若目标文件不存在，则 writeFile 方法会自动创建该文件；若目标文件存在，该方法会覆盖原有所有内容
  - fs.writeFile(file, data[, options], callback) ：写出内容

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

const dest = './backup/index.html';
var data = 'Some Data';
fs.writeFile( dest, data, (err)=>{
  if (err) throw err;
  console.log('Write Finished!');
});
```

读写文件（续2）

知识讲解

- 向文件中追加内容时，如果目标文件不存在，则 appendFile 方法会自动创建该文件；若目标文件存在，该方法会在原有内容后面追加新内容。
  - fs.appendFile(file, data[, options], callback)：追加内容

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

const dest = './backup/index.html';
var data = 'Some Data';
fs.appendFile( dest, data, (err)=>{
  if (err) throw err;
  console.log('Write Finished!');
});
```

使用 FS 模块

- 使用 FS 模块
  - 读取文件
  - 写入文件
  - 追加内容

课堂练习

HTTP 模块

HTTP 协议

知识讲解

- HTTP 协议是 TCP/IP 协议族中的应用层协议，用于在 Web 客户端与服务器之间传输“超文本内容”，如 HTML、CSS、JS、图片以及音视频等
- HTTP 协议是典型的基于“请求-响应”模型的协议，只有客户端发出了请求消息，服务器才会给出响应消息，并且一个请求消息只会得到一个响应消息

HTTP 协议（续1）

知识讲解

- 一个请求消息由四部分构成：
  - 请求起始行
  - 请求头部
  - CRLF
  - 请求主体(可能没有)
- 一个响应消息由四部分构成：
  - 响应起始行
  - 响应头部
  - CRLF
  - 响应主体(可能没有)

HTTP 模块概述

Technology Tarena 达内科技

- Node.js 内置的 HTTP 模块提供了一些非常底层的方法，用于创建使用 HTTP 协议的客户端应用或者服务器端应用
- 具体用途可以分为：
  - 创建并发起请求消息，等待并解析响应消息 —— 实现 Web 客户端
  - 接收并解析请求消息，构建并发送响应消息 —— 实现 Web 服务器

知识讲解

HTTP 客户端

Technology Tarena 达内科技

- http.request 是一个客户端工具，用于向 HTTP 服务器发起请求，实现访问
  - 可以模拟浏览器向任意的 Web 服务器发起 HTTP 请求消息，并获取该站点返回的响应数据
- 上述方法返回一个 http.ClientRequest 对象，用以描述一个 HTTP 请求消息
  - 其中的参数 options 可以指定请求消息起始行和请求消息头部

知识讲解

```
//创建一个 HTTP 请求消息对象
var request = http.request( options, [callback] )
//创建一个 HTTP GET 请求消息对象
var request = http.get( options, [callback] )
```

HTTP 客户端（续1）

知识讲解

- ClientRequest 常用方法：
  - write(chunk)：向服务器追加请求主体数据
  - end(chunk)：提交请求消息主体结束
  - setTimeout(timeout , fn)：设置请求消息超时时间
  - abort( )：终止请求
- ClientRequest 常用事件：
  - response：接收到响应消息
  - abort：请求终止事件
  - error：请求发生错误

HTTP 客户端（续2）

知识讲解

```
var http = require('http');
var options = {hostname:'www.tmooc.cn',port:80,path:'/'};
var req = http.get(options,function(res){
    console.log(`状态码：${res.statusCode}`);
    console.log(`响应头：${JSON.stringify(res.headers)}`);
    res.setEncoding('utf8');
    res.on('data',function(chunk){
        console.log(`响应内容：${chunk}`);
    });
});
req.setTimeout(3000,function(){
    req.abort();
    console.log('超时，取消请求');
});
req.on('error',function(err){
    console.log(`发生错误：${err}`);
});
```

HTTP 服务器

知识讲解

- http.server 是一个基于事件的 HTTP 服务器，其核心由 Node.js 下层 C++ 部分实现，接口由 JS 封装，兼顾了高性能与易用性
  - 用于创建一个非常底层的 Web 服务器应用，用以接收客户端请求消息，并返回响应消息，提供 Web 内容服务
  - 上述方法返回一个 http.Server 类型的对象，用以描述当前所创建的Web服务器

```
//创建一个基于 HTTP 协议的 Web 服务器
var server= http.createServer( [requestHandler] );
```

HTTP 服务器（续1）

知识讲解

- http.Server 常用方法：
  - listen(port, [host])：监听指定的服务端口
  - close( )：停止服务器的运行
  - setTimeout(timeout , fn)：设置服务器响应消息超时时间
- http.Server 常用事件：
  - connection：出现客户端连接
  - request：接收到请求消息
  - close：服务器停止事件
  - error：响应过程发生错误

HTTP 服务器（续2）

- 使用 createServer 时
  - 可以使用一个可选参数：参数为一个回调函数
  - 如果不用参数，则监听该方法创建对象的 request 事件

知识讲解

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

//创建server
var server = ???;

server.listen( 80 );
server.on('error', (err)=>{
});
```

```
var server = http.createServer(
  function(req,res){
        //解析请求消息
        //输出响应消息
});
```

```
var server = http.createServer( );
server.on('request', (request, response)=>{
                  //解析请求消息
                  //输出响应消息
});
```

HTTP 服务器（续3）

- Server 对象的 response 事件回调函数中
  - 第一个参数是一个 IncomingMessage对象，封装着客户端提交的请求消息数据
  - 第二个参数是一个 ServerResponse对象，用于构建向客户端输出的响应消息数据

知识讲解

```
server.on( 'request' ,(request, response)=>{
   console.log( request.method );
   console.log( request.headers );
   response.writeHead(200, 'OK' ,{
        'content-type' : 'text/html;charset=UTF-8'
   });
   response.write('<h1>文本内容</h1>');
});

server.listen(8080);
server.on('error', (err)=>{ console.log(`error:${err}`);});
```

使用 HTTP 模块构建服务器应用

- 使用 hTTP 模块构建服务器应用
- 使用客户端测试

课堂练习

总结和答疑