NodeJS开发学习

**1. Nodejs的介绍**

Node.js的是建立在Chrome的JavaScript的运行时，可方便地构建快速，可扩展的网络应用程序的平台。Node.js使用事件驱动，非阻塞I/O模型，轻量、高效，可以完美地处理时时数据，运行在不同的设备上。

1.1 Nodejs框架是基于V8的引擎，是目前速度最快的Javascript引擎；

1.2 Nodejs标准的web开发框架Expres；

1.3 Nodejs是单线程机制，Nodejs的另一个特点每一个 API 都是异步的，并作为一个独立线程运行，使用异步函数调用，并处理并发；

1.4 Nodejs 基本上所有的事件机制都是用设计模式中观察者模式实现。

1.5常用到的工具包，如socket.io, moment.js, underscore.js, async.js, express.js, bower.js, grunt.js, forever.js…

1.6 Nodejs劣势：

（1）计算密集型应用，让Javascript和C去拼计算性能，估计是不可能赢的；

（2）内存控制，让Javascript和Java比较复杂数据类型定义，也是很困难的。因为Javascript的面向对象是基于JSON的，而Java是直接使用内存结构。所以，通过JSON序列化和反序列的过程控制内存，Javascript就已经输了;

（3）大内存的应用，由于V8引擎有内存设计的限制，32位环境中最大堆是1G，64位环境中最大堆也不到2G，如果要一次读入10G数据，对于Nodejs来说也无法实现；

（4）静态服务器，虽然Nodejs的优势在IO密集集应用，但是和Nginx的处理静态资源还是有很大的差距；

（5）不需要异步的应用：比如系统管理，自行化脚本等，还是Python更顺手，Nodejs的异步调用可能会给编程带来一些麻烦。

1.7 Nodejs应用：<http://www.nodejs.net/a/20141016/232139.html>

**2. Nodejs事件驱动程序**

Node.js 使用事件驱动模型，当web server接收到请求，就把它关闭然后进行处理，然后去服务下一个web请求。

当这个请求完成，它被放回处理队列，当到达队列开头，这个结果被返回给用户。

这个模型非常高效可扩展性非常强，因为webserver一直接受请求而不等待任何读写操作。（这也被称之为非阻塞式IO或者事件驱动IO）

在事件驱动模型中，会生成一个主循环来监听事件，当检测到事件时触发回调函数。



整个事件驱动的流程就是这么实现的，非常简洁。有点类似于观察者模式，事件相当于一个主题(Subject)，而所有注册到这个事件上的处理函数相当于观察者(Observer)。

**3. 缓冲区**

JavaScript 语言自身只有字符串数据类型，没有二进制数据类型。

但在处理像TCP流或文件流时，必须使用到二进制数据。因此在 Node.js中，定义了一个 Buffer 类，该类用来创建一个专门存放二进制数据的缓存区。

在 Node.js 中，Buffer 类是随 Node 内核一起发布的核心库。Buffer 库为 Node.js 带来了一种存储原始数据的方法，可以让 Node.js 处理二进制数据，每当需要在 Node.js 中处理I/O操作中移动的数据时，就有可能使用 Buffer 库。原始数据存储在 Buffer 类的实例中。一个 Buffer 类似于一个整数数组，但它对应于 V8 堆内存之外的一块原始内存。

**4. 流**

Stream 是一个抽象接口，Node 中有很多对象实现了这个接口。例如，对http 服务器发起请求的request 对象就是一个 Stream，还有stdout（标准输出）。

Node.js，Stream 有四种流类型：

* **Readable** - 可读操作。
* **Writable** - 可写操作。
* **Duplex** - 可读可写操作.
* **Transform** - 操作被写入数据，然后读出结果。

所有的 Stream 对象都是 EventEmitter 的实例。常用的事件有：

* **data** - 当有数据可读时触发。
* **end** - 没有更多的数据可读时触发。
* **error** - 在接收和写入过程中发生错误时触发。
* **finish** - 所有数据已被写入到底层系统时触发。

**5. 模块系统**

为了让Node.js的文件可以相互调用，Node.js提供了一个简单的模块系统。

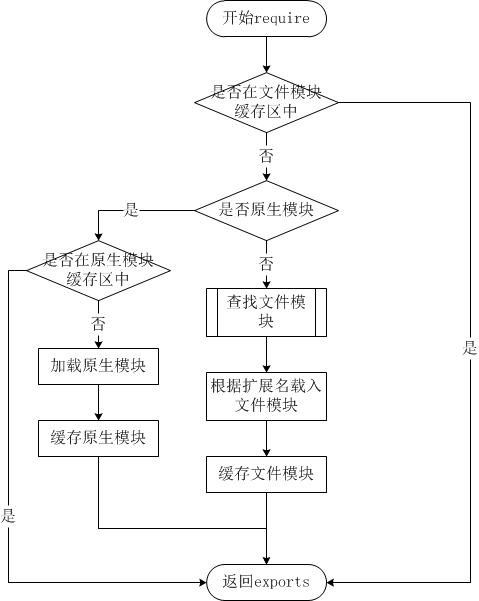
模块是Node.js 应用程序的基本组成部分，文件和模块是一一对应的。换言之，一个 Node.js 文件就是一个模块，这个文件可能是JavaScript 代码、JSON 或者编译过的C/C++ 扩展。

Node.js中自带了一个叫做"http"的模块，我们在我们的代码中请求它并把返回值赋给一个本地变量。

这把我们的本地变量变成了一个拥有所有 http 模块所提供的公共方法的对象。

Node.js 的 require方法中的文件查找策略如下：

由于Node.js中存在4类模块（原生模块和3种文件模块），尽管require方法极其简单，但是内部的加载却是十分复杂的，其加载优先级也各自不同。如下图所示：



从文件模块缓存中加载

尽管原生模块与文件模块的优先级不同，但是都不会优先于从文件模块的缓存中加载已经存在的模块。从原生模块加载

原生模块的优先级仅次于文件模块缓存的优先级。require方法在解析文件名之后，优先检查模块是否在原生模块列表中。以http模块为例，尽管在目录下存在一个http/http.js/http.node/http.json文件，require("http")都不会从这些文件中加载，而是从原生模块中加载。

原生模块也有一个缓存区，同样也是优先从缓存区加载。如果缓存区没有被加载过，则调用原生模块的加载方式进行加载和执行。

从文件加载

当文件模块缓存中不存在，而且不是原生模块的时候，Node.js会解析require方法传入的参数，并从文件系统中加载实际的文件，加载过程中的包装和编译细节在前一节中已经介绍过，这里我们将详细描述查找文件模块的过程，其中，也有一些细节值得知晓。

require方法接受以下几种参数的传递：

* http、fs、path等，原生模块。
* ./mod或../mod，相对路径的文件模块。
* /pathtomodule/mod，绝对路径的文件模块。
* mod，非原生模块的文件模块。

**6. 函数**

在JavaScript中，一个函数可以作为另一个函数接收一个参数。我们可以先定义一个函数，然后传递，也可以在传递参数的地方直接定义函数。

Node.js中函数的使用与Javascript类似，举例来说，你可以这样做：

function say(word) {

console.log(word);

}

function execute(someFunction, value) {

someFunction(value);

}

execute(say, "Hello");

以上代码中，我们把 say 函数作为execute函数的第一个变量进行了传递。这里返回的不是 say 的返回值，而是 say 本身！

这样一来， say 就变成了execute 中的本地变量 someFunction ，execute可以通过调用 someFunction() （带括号的形式）来使用 say 函数。

当然，因为 say 有一个变量， execute 在调用 someFunction 时可以传递这样一个变量。

execute(function(word){ console.log(word) }, "Hello");

我们在 execute 接受第一个参数的地方直接定义了我们准备传递给 execute 的函数。

用这种方式，我们甚至不用给这个函数起名字，这也是为什么它被叫做匿名函数 。

**7. Node.js 路由**

我们要为路由提供请求的URL和其他需要的GET及POST参数，随后路由需要根据这些数据来执行相应的代码。

因此，我们需要查看HTTP请求，从中提取出请求的URL以及GET/POST参数。这一功能应当属于路由还是服务器（甚至作为一个模块自身的功能）确实值得探讨，但这里暂定其为我们的HTTP服务器的功能。

我们需要的所有数据都会包含在request对象中，该对象作为onRequest()回调函数的第一个参数传递。但是为了解析这些数据，我们需要额外的Node.JS模块，它们分别是url和querystring模块。

url.parse(string).query

|

url.parse(string).pathname |

| |

| |

------ -------------------

http://localhost:8888/start?foo=bar&hello=world

--- -----

| |

| |

querystring(string)["foo"] |

|

querystring(string)["hello"]

当然我们也可以用querystring模块来解析POST请求体中的参数

**8. Node.js 全局对象**

JavaScript 中有一个特殊的对象，称为全局对象（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。

在浏览器 JavaScript 中，通常 window 是全局对象， 而 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。

在 Node.js 我们可以直接访问到 global 的属性，而不需要在应用中包含它。

global 最根本的作用是作为全局变量的宿主。按照 ECMAScript 的定义，满足以下条件的变量是全局变量：

* 在最外层定义的变量；
* 全局对象的属性；
* 隐式定义的变量（未定义直接赋值的变量）。

当你定义一个全局变量时，这个变量同时也会成为全局对象的属性，反之亦然。需要注意的是，在 Node.js 中你不可能在最外层定义变量，因为所有用户代码都是属于当前模块的，而模块本身不是最外层上下文。

**注意：** 永远使用 var 定义变量以避免引入全局变量，因为全局变量会污染命名空间，提高代码的耦合风险。

**\_\_filename** 表示当前正在执行的脚本的文件名。它将输出文件所在位置的绝对路径，且和命令行参数所指定的文件名不一定相同。 如果在模块中，返回的值是模块文件的路径。

**\_\_dirname** 表示当前执行脚本所在的目录。

**setTimeout(cb, ms)** 全局函数在指定的毫秒(ms)数后执行指定函数(cb)。：setTimeout() 只执行一次指定函数。返回一个代表定时器的句柄值。

**clearTimeout( t )** 全局函数用于停止一个之前通过 setTimeout() 创建的定时器。 参数 **t** 是通过 setTimeout() 函数创建的计算器。

**setInterval(cb, ms)** 全局函数在指定的毫秒(ms)数后执行指定函数(cb)。返回一个代表定时器的句柄值。可以使用 **clearInterval(t)** 函数来清除定时器。setInterval() 方法会不停地调用函数，直到 clearInterval() 被调用或窗口被关闭。

**console** 用于提供控制台标准输出，它是由 Internet Explorer 的 JScript 引擎提供的调试工具，后来逐渐成为浏览器的事实标准。Node.js 沿用了这个标准，提供与习惯行为一致的 console 对象，用于向标准输出流（stdout）或标准错误流（stderr）输出字符。

**process** 是一个全局变量，即 global 对象的属性。它用于描述当前Node.js 进程状态的对象，提供了一个与操作系统的简单接口。通常在你写本地命令行程序的时候，少不了要 和它打交道。

**9. Node.js 常用工具**

util.inherits(constructor, superConstructor)是一个实现对象间原型继承的函数。

JavaScript 的面向对象特性是基于原型的，与常见的基于类的不同。JavaScript 没有 提供对象继承的语言级别特性，而是通过原型复制来实现的。

在这里我们只介绍util.inherits 的用法，示例如下：

var util = require('util');

function Base() {

this.name = 'base';

this.base = 1991;

this.sayHello = function() {

console.log('Hello ' + this.name);

};

}

Base.prototype.showName = function() {

console.log(this.name);

};

function Sub() {

this.name = 'sub';

}

util.inherits(Sub, Base);

var objBase = new Base();

objBase.showName();

objBase.sayHello();

console.log(objBase);

var objSub = new Sub();

objSub.showName();

//objSub.sayHello();

console.log(objSub);

我们定义了一个基础对象Base 和一个继承自Base 的Sub，Base 有三个在构造函数 内定义的属性和一个原型中定义的函数，通过util.inherits 实现继承。运行结果如下：

base

Hello base

{ name: 'base', base: 1991, sayHello: [Function] }

sub

{ name: 'sub' }

**注意：**Sub 仅仅继承了Base 在**原型中定义的函数**，而构造函数内部创造的 base 属 性和 sayHello 函数都没有被 Sub 继承。同时，在原型中定义的属性不会被console.log 作 为对象的属性输出。

util.inspect(object,[showHidden],[depth],[colors])是一个将任意对象转换为字符串的方法，通常用于调试和错误输出。它至少接受一个参数 object，即要转换的对象。

showHidden 是一个可选参数，如果值为 true，将会输出更多隐藏信息。

depth 表示最大递归的层数，如果对象很复杂，你可以指定层数以控制输出信息的多 少。如果不指定depth，默认会递归2层，指定为 null 表示将不限递归层数完整遍历对象。 如果color 值为 true，输出格式将会以ANSI 颜色编码，通常用于在终端显示更漂亮 的效果。

特别要指出的是，util.inspect 并不会简单地直接把对象转换为字符串，即使该对 象定义了toString 方法也不会调用。

var util = require('util');

function Person() {

this.name = 'byvoid';

this.toString = function() {

return this.name;

};

}

var obj = new Person();

console.log(util.inspect(obj));

console.log(util.inspect(obj, true));

运行结果是：

{ name: 'byvoid', toString: [Function] }

{ toString:

{ [Function]

[prototype]: { [constructor]: [Circular] },

[caller]: null,

[length]: 0,

[name]: '',

[arguments]: null },

name: 'byvoid' }

**util.isArray(object)**

如果给定的参数 "object" 是一个数组返回true，否则返回false。

**util.isRegExp(object)**

如果给定的参数 "object" 是一个正则表达式返回true，否则返回false。

**util.isDate(object)**

如果给定的参数 "object" 是一个日期返回true，否则返回false。

**util.isError(object)**

如果给定的参数 "object" 是一个错误对象返回true，否则返回false。

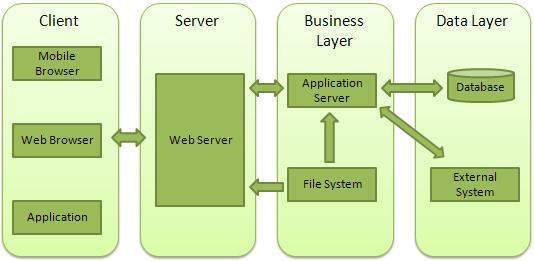
**10. Web模块**

Web服务器一般指网站服务器，是指驻留于因特网上某种类型计算机的程序，Web服务器的基本功能就是提供Web信息浏览服务。它只需支持HTTP协议、HTML文档格式及URL，与客户端的网络浏览器配合。

大多数 web 服务器都支持服务端的脚本语言（php、python、ruby）等，并通过脚本语言从数据库获取数据，将结果返回给客户端浏览器。

目前最主流的三个Web服务器是Apache、Nginx、IIS。

**Web应用架构**



* **Client** - 客户端，一般指浏览器，浏览器可以通过 HTTP 协议向服务器请求数据。
* **Server** - 服务端，一般指 Web 服务器，可以接收客户端请求，并向客户端发送响应数据。
* **Business** - 业务层， 通过 Web 服务器处理应用程序，如与数据库交互，逻辑运算，调用外部程序等。
* **Data** - 数据层，一般由数据库组成。

**11. Express框架**

Express 是一个简洁而灵活的 node.js Web应用框架, 提供了一系列强大特性帮助你创建各种 Web 应用，和丰富的 HTTP 工具。

使用 Express 可以快速地搭建一个完整功能的网站。

Express 框架核心特性：

* 可以设置中间件来响应 HTTP 请求。
* 定义了路由表用于执行不同的 HTTP 请求动作。
* 可以通过向模板传递参数来动态渲染 HTML 页面。

**12. Restful API**

REST即表述性状态传递（英文：Representational State Transfer，简称REST）是Roy Fielding博士在2000年他的博士论文中提出来的一种软件架构风格。

表述性状态转移是一组架构约束条件和原则。满足这些约束条件和原则的应用程序或设计就是RESTful。需要注意的是，REST是设计风格而不是标准。REST通常基于使用HTTP，URI，和XML（标准通用标记语言下的一个子集）以及HTML（标准通用标记语言下的一个应用）这些现有的广泛流行的协议和标准。REST 通常使用 JSON 数据格式。

以下为 REST 基本架构的四个方法：

HTTP方法

* **GET** - 用于获取数据。
* **PUT** - 用于添加数据。
* **DELETE** - 用于删除数据。
* **POST** - 用于更新或添加数据。

RESTful Web Services

Web service是一个平台独立的，低耦合的，自包含的、基于可编程的web的应用程序，可使用开放的XML（标准通用标记语言下的一个子集）标准来描述、发布、发现、协调和配置这些应用程序，用于开发分布式的互操作的应用程序。

基于 REST 架构的 Web Services 即是 RESTful。

由于轻量级以及通过 HTTP 直接传输数据的特性，Web 服务的 RESTful 方法已经成为最常见的替代方法。可以使用各种语言（比如 Java 程序、Perl、Ruby、Python、PHP 和 Javascript[包括 Ajax]）实现客户端。

RESTful Web 服务通常可以通过自动客户端或代表用户的应用程序访问。但是，这种服务的简便性让用户能够与之直接交互，使用它们的 Web 浏览器构建一个 GET URL 并读取返回的内容。

**13. NodeJS多进程**

我们都知道 Node.js 是以单线程的模式运行的，但它使用的是事件驱动来处理并发，这样有助于我们在多核 cpu 的系统上创建多个子进程，从而提高性能。

每个子进程总是带有三个流对象：child.stdin, child.stdout 和child.stderr。他们可能会共享父进程的 stdio 流，或者也可以是独立的被导流的流对象。

Node 提供了 child\_process 模块来创建子进程，方法有：

* **exec** - child\_process.exec 使用子进程执行命令，缓存子进程的输出，并将子进程的输出以回调函数参数的形式返回。
* **spawn** - child\_process.spawn 使用指定的命令行参数创建新进程。
* **fork** - child\_process.fork 是 spawn()的特殊形式，用于在子进程中运行的模块，如 fork('./son.js') 相当于 spawn('node', ['./son.js']) 。与spawn方法不同的是，fork会在父进程与子进程之间，建立一个通信管道，用于进程之间的通信。