

NodeJS入门和企业级应用开发

第8天课堂笔记（本课程共8天）

前端与移动开发学院

http://web.itcast.cn

# 目录

[目录 2](#_Toc27194)

[一、Mongoose 3](#_Toc12104)

[二、web Socket和Socket.IO框架 5](#_Toc23280)

# 一、Mongoose

Node自己复习的时候，顺序：

Node.js特点：单线程、异步I/O（非阻塞I/O）、事件驱动（事件环）

适合的程序：就是没有太多的计算，I/O比较多的业务。举例：留言本、考试系统、说说、图片裁切服务器

Node.js原生： http、fs、path、url。 静态服务、简单路由、GET、POST请求。

模块：formidable、gm、express

Express：中间件、MVC建站、模板引擎ejs、静态服务、简单路由、GET、POST请求、MD5加密、图片上传。

服务器的一些概念：Cookie、Session

持久化NoSQL： 非关系型数据库，Not Only SQL。

特点：没有schema，没有行和列。用文档（JSON）来存储。

MongoDB：安装、开启、导入数据、Shell管理数据库、Mongo Vue、Node.js做CRUD（增删改查）、DAO层的封装、索引、操作符$set $lt $gt $push $pull

Mongoose： ODM，不用直接操作数据库，操作对象，这个对象自动持久。

Defining your schema 定义文档结构

schema定义的时候，支持的类型：

The permitted SchemaTypes are

String、Number、Date、Buffer、Boolean、Mixed、ObjectId、Array

转换为对象：

|  |
| --- |
| 1. mongoose.model(modelName, schema): |

定义对象（实例）方法：

|  |
| --- |
| 1. // define a schema 2. var animalSchema = new Schema({ name: String, type: String }); 3. // assign a function to the "methods" object of our animalSchema 4. animalSchema.**methods**.findSimilarTypes = function (cb) { 5. return this.model('Animal').find({ type: this.type }, cb); 6. } |

虚拟属性：

|  |
| --- |
| 1. // define a schema 2. var personSchema = new Schema({ 3. name: { 4. first: String, 5. last: String 6. } 7. }); 8. // compile our model 9. var Person = mongoose.model('Person', personSchema); 10. // create a document 11. var bad = new Person({ 12. name: { first: 'Walter', last: 'White' } 13. }); |

# 二、web Socket和Socket.IO框架

HTTP无法轻松实现实时应用：

● HTTP协议是无状态的，服务器只会响应来自客户端的请求，但是它与客户端之间不具备持续连接。

● 我们可以非常轻松的捕获浏览器上发生的事件（比如用户点击了盒子），这个事件可以轻松产生与服务器的数据交互（比如Ajax）。但是，反过来却是不可能的：服务器端发生了一个事件，服务器无法将这个事件的信息实时**主动**通知它的客户端。只有在客户端查询服务器的当前状态的时候，所发生事件的信息才会从服务器传递到客户端。

但是，确实聊天室确实存在。

方法：

● 长轮询：客户端每隔很短的时间，都会对服务器发出请求，查看是否有新的消息，只要轮询速度足够快，例如1秒，就能给人造成交互是实时进行的印象。这种做法是无奈之举，实际上对服务器、客户端双方都造成了大量的性能浪费。

● 长连接：客户端只请求一次，但是服务器会将连接保持，不会返回结果（想象一下我们没有写res.end()时，浏览器一直转小菊花）。服务器有了新数据，就将数据发回来，又有了新数据，就将数据发回来，而一直保持挂起状态。这种做法的也造成了大量的性能浪费。

WebSocket协议能够让浏览器和服务器全双工实时通信，互相的，服务器也能主动通知客户端了。

● WebSocket的原理非常的简单：利用HTTP请求产生握手，HTTP头部中含有WebSocket协议的请求，所以握手之后，二者转用TCP协议进行交流（QQ的协议）。现在的浏览器和服务器之间，就是QQ和QQ服务器的关系了。

所以WebSocket协议，需要浏览器支持，更需要服务器支持。

● 支持WebSocket协议的浏览器有：Chrome 4、火狐4、IE10、Safari5

● 支持WebSocket协议的服务器有：Node 0、Apach7.0.2、Nginx1.3

Node.js上需要写一些程序，来处理TCP请求。

● Node.js从诞生之日起，就支持WebSocket协议。不过，从底层一步一步搭建一个Socket服务器很费劲（想象一下Node写一个静态文件服务都那么费劲）。所以，有大神帮我们写了一个库Socket.IO。

● Socket.IO是业界良心，新手福音。它屏蔽了所有底层细节，让顶层调用非常简单。并且还为不支持WebSocket协议的浏览器，提供了长轮询的透明模拟机制。

● Node的单线程、非阻塞I/O、事件驱动机制，使它非常适合Socket服务器。

网址：<http://socket.io/>

先要npm下载这个库

|  |
| --- |
| 1. npm install socket.io |

写原生的JS，搭建一个服务器，server创建好之后，创建一个io对象

|  |
| --- |
| 1. var http = require("http"); 2. var server = http.createServer(function(req,res){ 3. res.end("你好"); 4. }); 5. **var io = require('socket.io')(server);**   **//监听连接事件**  **io.on("connection",function(socket){**  **console.log("1个客户端连接了");**  **})**   1. server.listen(3000,"127.0.0.1"); |

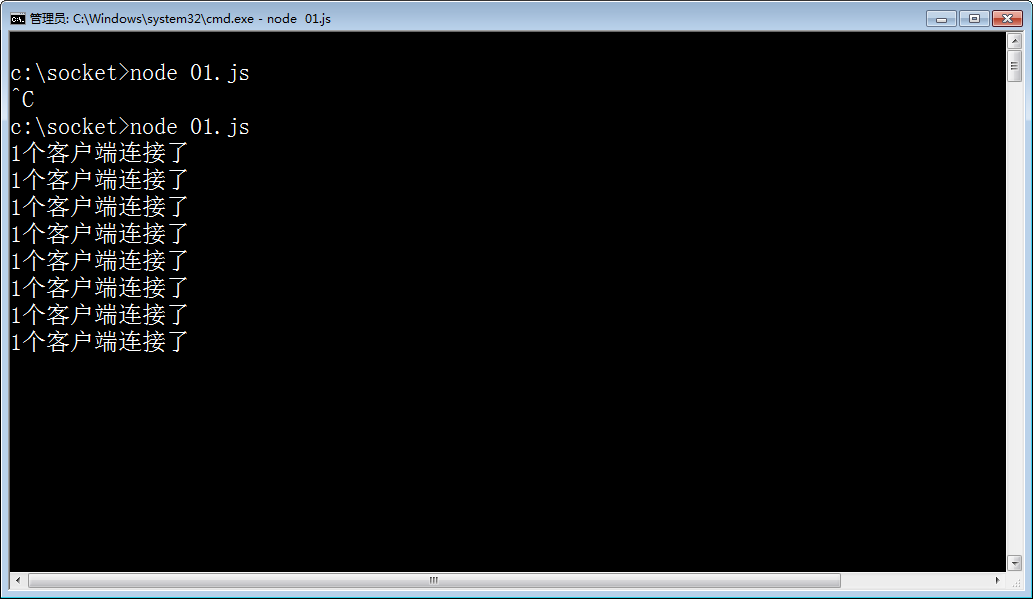
写完这句话之后，你就会发现，http://127.0.0.1:3000/socket.io/socket.io.js 就是一个js文件的地址了。

现在需要制作一个index页面，这个页面中，必须引用秘密js文件。调用io函数，取得socket对象。

|  |
| --- |
| 1. <!DOCTYPE html> 2. <html lang="en"> 3. <head> 4. <meta charset="UTF-8"> 5. <title>Document</title> 6. </head> 7. <body> 8. <h1>我是index页面，我引用了秘密script文件</h1> 9. <script type="text/javascript" src="/socket.io/socket.io.js"></script> 10. <script type="text/javascript"> 11. var socket = io(); 12. </script> 13. </body> 14. </html> |

此时，在服务器上，app.js中就要书写静态文件呈递程序，能够呈递静态页面。

|  |
| --- |
| 1. var server = http.createServer(function(req,res){ 2. if(req.url == "/"){ 3. //显示首页 4. fs.readFile("./index.html",function(err,data){ 5. res.end(data); 6. }); 7. } 8. }); |



至此，服务器和客户端都有socket对象了。服务器的socket对象：

服务器端的：

|  |
| --- |
| 1. var io = require('socket.io')(server); 2. //监听连接事件 3. io.on("connection",function(**socket**){ 4. console.log("1个客户端连接了"); 5. socket.on("tiwen",function(msg){ 6. console.log("本服务器得到了一个提问" + msg); 7. **socket**.emit("huida","吃了"); 8. }); 9. }); |

每一个连接上来的用户，都有一个socket。 由于我们的emit语句，是socket.emit()发出的，所以指的是向这个客户端发出语句。

广播，就是给所有当前连接的用户发送信息：

|  |
| --- |
| 1. //创建一个io对象 2. var io = require('socket.io')(server); 3. //监听连接事件 4. io.on("connection",function(socket){ 5. console.log("1个客户端连接了"); 6. socket.on("tiwen",function(msg){ 7. console.log("本服务器得到了一个提问" + msg); 8. io.emit("huida","吃了"); 9. }); 10. }); |