



【Android OTA】用nodejs搭建服务器

📅 2017/04/21 📁 Android

OTA，Over-the-Air的简写，OTA升级就是通过GPRS、3G、无线网络下载升级补丁升级，不用通过有线连接来升级。Android的应用或者是整个系统，都可以通过OTA的方式进行版本的更新升级。

OTA具体原理自行google，或者参考[这篇文章](#)。本文和接下来的两篇文章主要介绍的是具体的实现过程。

OTA升级大致过程

1. 设备向服务器进行版本更新检测请求
2. 服务器将更新信息返回到设备端
3. 设备通过返回信息下载指定的更新文件
4. 下载完成后设备安装升级

Android单个应用和整个系统的升级方式存在差异，接下来两篇文章会分别介绍实现。

本文重点实现升级过程中的第二步，搭建一个服务器Demo，以方便后续的测试工作。服务器用的nodejs，使用较为简单，没接触过的也可以跟着下面的步骤将服务器搭建在本地运行起来。

OTA服务器搭建

安装nodejs

用的是Mac系统，安装命令

```
brew install node
```

查看是否安装完成

```
$ node -v  
v6.2.0
```

npm 是专门管理nodejs包的工具，用来方便地安装第三方模块，安装nodejs时应该也默认同时安装了npm，可以命令查看

```
$ npm -v  
3.8.9
```

运行

安装完成后，先实现个简单的Demo，只需简单几行代码便可在本地运行起一个服务器。

先创建一个文件夹 **Server**，在文件夹内创建文件 **SimpleServer.js**

用文本编辑工具打开**SimpleServer.js**，代码实现：

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

var server = http.createServer(function(req, res) {
    res.end('Hello!');
}).listen(3001);
console.log('Server listening at port: 3001');
```

然后打开命令行工具，**cd** 到 **Server** 目录下，敲入命令运行服务器：

```
node SimpleServer.js
```

命令行打印输出：

```
Server listening at port: 3001
```

此时服务器开始监听来自本地端口**3001**的请求。

打开浏览器，地址栏输入 **http://localhost:3001**，可以看到浏览器界面返回文本 **Hello!**。即一个最简单的服务器。

Express框架

上边实现的服务器对任何请求都返回*Hello!* 文本，现实中服务器当然没那么简单。OTA服务器在接到请求时需要返回对应信息，并提供更新文件的下载接口。

Express 是一种保持最低程度规模的灵活 Node.js Web 应用程序框架，为 Web 和移动应用程序提供一组强大的功能。**Express** 提供的各种 HTTP 实用程序方法和中间件能帮助我们方便的处理网络请求。[具体可参考官方文档](#)

使用Express应用生成器快速搭建框架

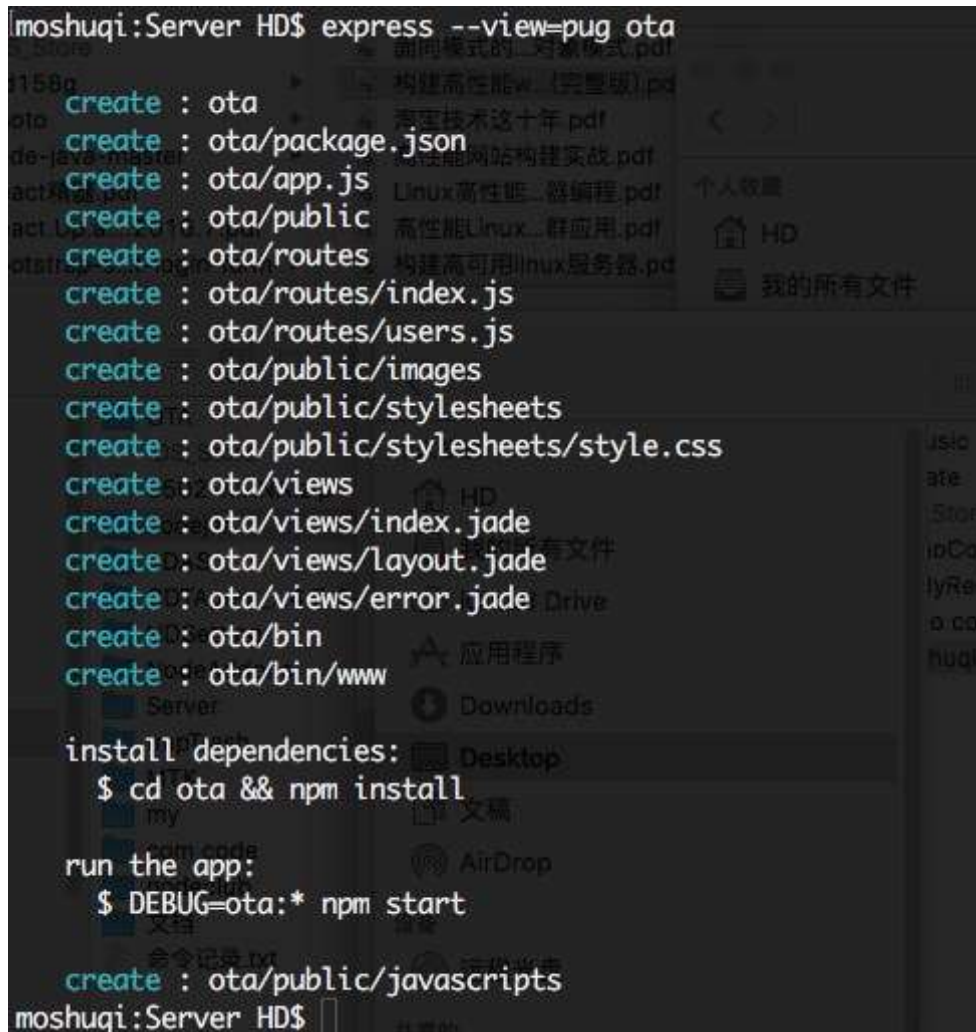
使用以下命令安装 express：

```
npm install express-generator -g
```

输入如下命令，在当前目录下创建名为 **ota** 的 Express 应用

```
express --view=pug ota
```

看结果打印，在ota目录下自动创建了对应文件



```
moshuqi:Server HD$ express --view=pug ota
create : ota
create : ota/package.json
create : ota/app.js
create : ota/public
create : ota/routes
create : ota/routes/index.js
create : ota/routes/users.js
create : ota/public/images
create : ota/public/stylesheets
create : ota/public/stylesheets/style.css
create : ota/views
create : ota/views/index.jade
create : ota/views/layout.jade
create : ota/views/error.jade
create : ota/bin
create : ota/bin/www

install dependencies:
$ cd ota && npm install

run the app:
$ DEBUG=ota:* npm start

create : ota/public/javascripts
moshuqi:Server HD$
```

cd 到 **ota** 目录下，安装需要的依赖

```
cd ota/
npm install
```

完成后，在node_modules目录下会下载好项目需要的第三方依赖模块

```
moshugi@ota HDS npm install
```

```
npm WARN deprecated jade@1.11.0: Jade has been renamed to pug, please install the latest version of pug instead of jade
npm WARN deprecated transformers@2.1.0: Deprecated, use jstransformer

ota@0.0.0 /Users/HD/Desktop/OTA/Server/ota
├─┬ body-parser@1.15.2
│ └─ bytes@2.4.0
│ └─ content-type@1.0.2
│ └─ depd@1.1.0
│ └─ http-errors@1.5.1
│ └─ inherits@2.0.3
│ └─ setprototypeof@1.0.2
│ └─ statuses@1.3.1
│ └─ iconv-lite@0.4.13
├─ on-finished@2.3.0
├─ ee-first@1.1.1
├─ qs@6.2.0
├─ raw-body@2.1.7
├─ unpipe@1.0.0
├─ type-is@1.6.15
├─ media-typer@0.3.0
├─ mime-types@2.1.15
├─ mime-db@1.27.0
├─ ie-parser@1.4.3
├─ cookie@0.3.1
├─ cookie-signature@1.0.6
├─ debug@2.2.0
├─ ms@0.7.1
├─ express@4.13.4
├─ accepts@1.2.13
├─ negotiator@0.5.3
├─ array-flatten@1.1.1
├─ content-disposition@0.5.1
├─ cookie@0.1.5
```

```
var app = require('./app');
var debug = require('debug')('ota:server');
var http = require('http');

/**
 * Get port from environment and store in Express.
 */
var port = normalizePort(process.env.PORT || '3000');
app.set('port', port);

/**
 * Create HTTP server.
 */
var server = http.createServer(app);

module.exports = server;
```

```
bash-3.2$ ls
Server
bash-3.2$ cd Server/
bash-3.2$ node OTAServer.js
Server listening at port: 3001

^C
bash-3.2$
bash-3.2$ node SimpleServer.js
Server listening at port: 3001
```

项目的运行文件为 **bin/www**，打开查看源文件，可看到服务器运行端口默认情况下为 **3000**

```
/**
 * Module dependencies.
 */

var app = require('../app');
var debug = require('debug')('ota:server');
var http = require('http');

/**
 * Get port from environment and store in Express.
 */

var port = normalizePort(process.env.PORT || '3000');
app.set('port', port);

/**
 * Create HTTP server.
 */

var server = http.createServer(app);
```

命令行运行服务器

```
node bin/www
```

浏览器打开网址 <http://localhost:3000>

Express框架默认生成的界面



同时可以看到访问服务器时命令行的打印信息


```
moshuqi:ota HD$
moshuqi:ota HD$ node bin/www

GET / 200 1367.516 ms - 170
GET / 200 26.532 ms - 170
GET /stylesheets/style.css 200 15.793 ms - 111
GET /stylesheets/style.css 304 1.483 ms - -
GET /favicon.ico 404 43.514 ms - 1135
GET /favicon.ico 404 16.273 ms - 1135
GET / 200 26.183 ms - 170
GET /stylesheets/style.css 200 9.594 ms - 111
GET /favicon.ico 404 15.272 ms - 1135
```

Ps. 关闭服务器时，在命令行敲 **Ctrl + C**，如果通过 **Ctrl + Z** 或者直接关闭命令行窗口界面的方式来退出服务器，会导致服务器监听的端口一直被占用，再次重启服务器时会失败。此时要么修改服务器监听的端口，要么将原端口所对应的进程杀掉。

杀指定端口进程方法，通过 **lsof -i:端口号** 查看端口对应PID，然后 **kill -9 PID**

杀掉node相关的进程

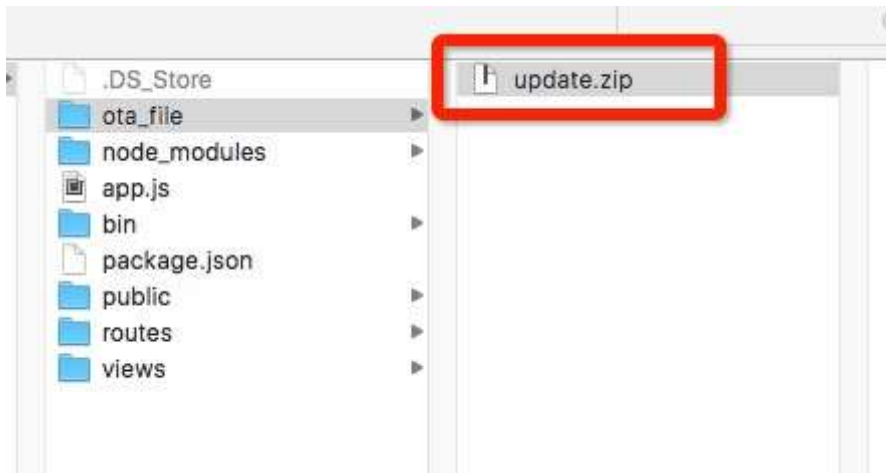
```
moshuqi:ota HD$
moshuqi:ota HD$ lsof -i:3000
COMMAND  PID USER  FD  TYPE             DEVICE SIZE/OFF NODE NAME
Google   8817  HD    14u  IPv6 0x96eb9c14bb29e223 0t0  TCP localhost:64962->localhost:hbc (CLOSED)
Google   8817  HD    84u  IPv6 0x96eb9c14bb29e223 0t0  TCP localhost:64963->localhost:hbc (CLOSED)
node     98998  HD    13u  IPv6 0x96eb9c14bb29d203 0t0  TCP *:hbc (LISTEN)

moshuqi:ota HD$ kill -9 98998
moshuqi:ota HD$
[1]+  Killed: 9 node bin/www
moshuqi:ota HD$
moshuqi:ota HD$ lsof -i:3000
COMMAND  PID USER  FD  TYPE             DEVICE SIZE/OFF NODE NAME
Google   8817  HD    14u  IPv6 0x96eb9c14bb29e223 0t0  TCP localhost:64962->localhost:hbc (CLOSED)
Google   8817  HD    84u  IPv6 0x96eb9c14bb29e223 0t0  TCP localhost:64963->localhost:hbc (CLOSED)
moshuqi:ota HD$
```

OTA服务器实现

ota更新文件

在项目根目录下创建文件夹 **ota_file**，将更新文件命名为 **update.zip** 放在该目录下



路由实现

获取更新信息

通过 `/update_info` 请求更新信息，将设备当前的版本信息作为参数，例如
http://localhost:3000/update_info?versionName=v01

直接打开文件 `routes/index.js`，看到代码

```
/* GET home page. */
router.get('/', function(req, res, next) {
  res.render('index', { title: 'Express' });
});
```

大概可以理解为，对于 `/` 请求，服务器会通过代码 `res.render('index', { title: 'Express' });`，将指定页面返回到浏览器，即我们看到的页面。

对于 `/update_info` 请求，我们实现个类似的处理方式

具体代码：

```
router.get('/update_info', function(req, res, next) {
  var name = req.query.versionName;
  console.log('current version:' + name);

  var info = {
    'url': '/ota_file/update.zip',
    'updateMessage': 'Fix bugs.',
    'versionName': 'v2',
    'md5': ''
  };

  var dir = process.cwd() + '/ota_file'
  var filePath = path.join(dir, 'update.zip');
  var md5 = getFileMD5(filePath);
  info.md5 = md5;
```

```
res.writeHead(200, {'Content-Type': 'application/json;charset=utf-8'});
res.end(JSON.stringify(info));
});
```

当我们在浏览器输入 http://localhost:3000/update_info?versionName=v01 便会运行到该处代码

```
var name = req.query.versionName;
```

获取请求中 *versionName* 参数的值。

检测是否有更新时，设备会将当前版本做为参数发送到服务器，通过与服务器最新的版本进行对比以决定是否需要升级。

```
var info = {
  'url': '/ota_file/update.zip',
  'updateMessage': 'Fix bugs.',
  'versionName': 'v2',
  'md5': ''
};
```

- url: 更新文件的下载地址
- updateMessage: 用来描述新版本的更新信息
- versionName: 新版本的版本名称
- md5: 更新文件的md5值，用来和设备下载完成的更新包后md5值对比，保证下载完成的文件和服务器文件一致

获取更新文件的md5值，**filePath** 为更新文件的目录

```
var dir = process.cwd() + '/ota_file'
var filePath = path.join(dir, 'update.zip');
var md5 = getFileMD5(filePath);
info.md5 = md5;
```

最后将 **info** 转成 **json** 格式返回

```
res.writeHead(200, {'Content-Type': 'application/json;charset=utf-8'});
res.end(JSON.stringify(info));
```

Android设备最终获取到json格式的数据，通过特定字段将对应的值解析出来。

getFileMD5 函数用来获取文件md5值，实现代码：

```
function getFileMD5(filePath) {
  var buffer = fs.readFileSync(filePath);
```



```
var fsHash = crypto.createHash('md5');

fsHash.update(buffer);
var md5 = fsHash.digest('hex');
console.log("文件的MD5是: %s", md5);

return md5;
}
```

一些处理操作作用到了系统特定的模块，使用前需要引入

```
var express = require('express');
var router = express.Router();

// add module
var fs = require('fs');
var path = require('path');
var crypto = require('crypto');
```

加上代码后，重新运行服务器，进行更新信息的请求，可以看到浏览器返回了对应的json数据



提供文件下载

返回的json数据中可看到其中

```
'url': '/ota_file/update.zip'
```

/ota_file/update.zip 即为我们下载文件的路径

文件的下载路由实现代码

```
router.get('/ota_file/:filename', function(req, res, next) {
  var filename = req.params.filename;
  var dir = process.cwd() + '/ota_file'
  var filePath = path.join(dir, filename);

  fs.exists(filePath, function(exist) {
    if (exist) {
      console.log('downloading:' + filename);
      res.download(filePath);
    }
    else {
      res.set('Content-type', 'text/html');
      res.end('File not exist.');
```

/ota_file/update.zip 中的 **update.zip** 对应 **/ota_file/:filename** 中的 **filename**，所以下载请求的文件名可通过 *filename* 参数获取

```
var filename = req.params.filename;
```

通过文件名合成文件路径

```
var dir = process.cwd() + '/ota_file'
var filePath = path.join(dir, filename);
```

fs.exists 检验指定文件路径是否存在，若是则进行文件下载

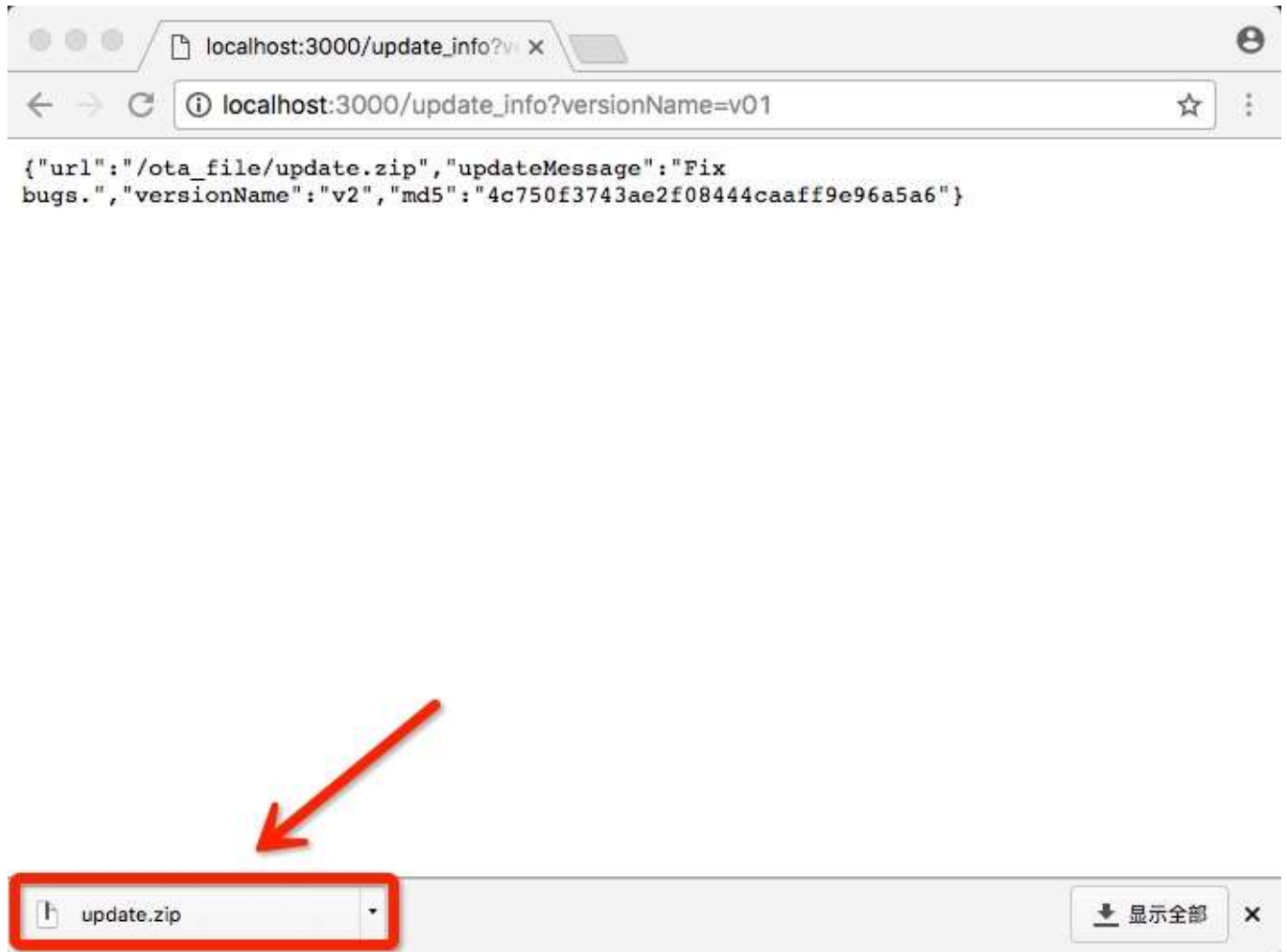
服务器的文件下载通过内置的 **download** 方法实现。

```
res.download(filePath);
```

download 方法实现了http的断点续传协议，用浏览器下载时可尝试暂停下载，再继续下载时会延续之前的下载进度。

Ps. 下载也可通过打开文件将数据流写到客户端的方式来实现，不过过程略麻烦，若要实现断点续传还得根据http协议range属性自行处理。

文件下载实现完成后，重启服务器生效，浏览器输入 `localhost:3000/ota_file/update.zip`，回车后可看到文件下载到本地



以上

服务器大概实现完成。实际生产环境中的处理流程会做更多判断和容错，这里只做了简单处理，用来方便给后续Android OTA升级提供测试。

[源码地址](#)

关注 `ota/routes/index.js` 文件即可

想留言却没看到评论框？[点这里](#)。

Post Directory