## 客户端重构-升级设计文档

## 前言

1. 版本语义说明

版本格式：主版本号.次版本号.修订号，版本号递增规则如下：

1. 主版本号：当你做了不兼容的 API 修改，
2. 次版本号：当你做了向下兼容的功能性新增，
3. 修订号：当你做了向下兼容的问题修正。
4. 先行版本号及版本编译信息可以加到“主版本号.次版本号.修订号”的后面，作为延伸。
5. 版本控制规范

范例：1.0.0-alpha < 1.0.0-alpha.1 < 1.0.0-alpha.beta < 1.0.0-beta < 1.0.0-beta.2 < 1.0.0-beta.11 < 1.0.0- rc.1 < 1.0.0。

1. 版本类型

|  |  |  |
| --- | --- | --- |
| 序号（级别） | 版本名称 | 版本说明 |
| 1 | stable | 稳定版。在开源软件中，都有stable版，这个就是开源软件的最终发行版，用户可以放心大胆的用了。 |
| 2 | RC | 全写：Release Candidate（候选版本），该版本又较beta版更进一步了，该版本功能不再增加，和最终发布版功能一样。这个版本有点像最终发行版之前的一个类似 预览版，这个的发布就标明离最终发行版不远了。作为普通用户，如果你很急着用这个软件的话，也可以下载这个版本。 |
| 3 | Beta | 公开测试版。β是希腊字母的第二个，顾名思义，这个版本比alpha版发布得晚一些，主要是给“部落”用户和忠实用户测试用的，该版本任然存 在很多BUG，但是相对alpha版要稳定一些。这个阶段版本的软件还会不断增加新功能。如果你是发烧友，可以下载这个版本。 |
| 4 | Alpha | 内部测试版。α是希腊字母的第一个，表示最早的版本，一般用户不要下载这个版本，这个版本包含很多BUG，功能也不全，主要是给开发人员和 测试人员测试和找BUG用的。 |

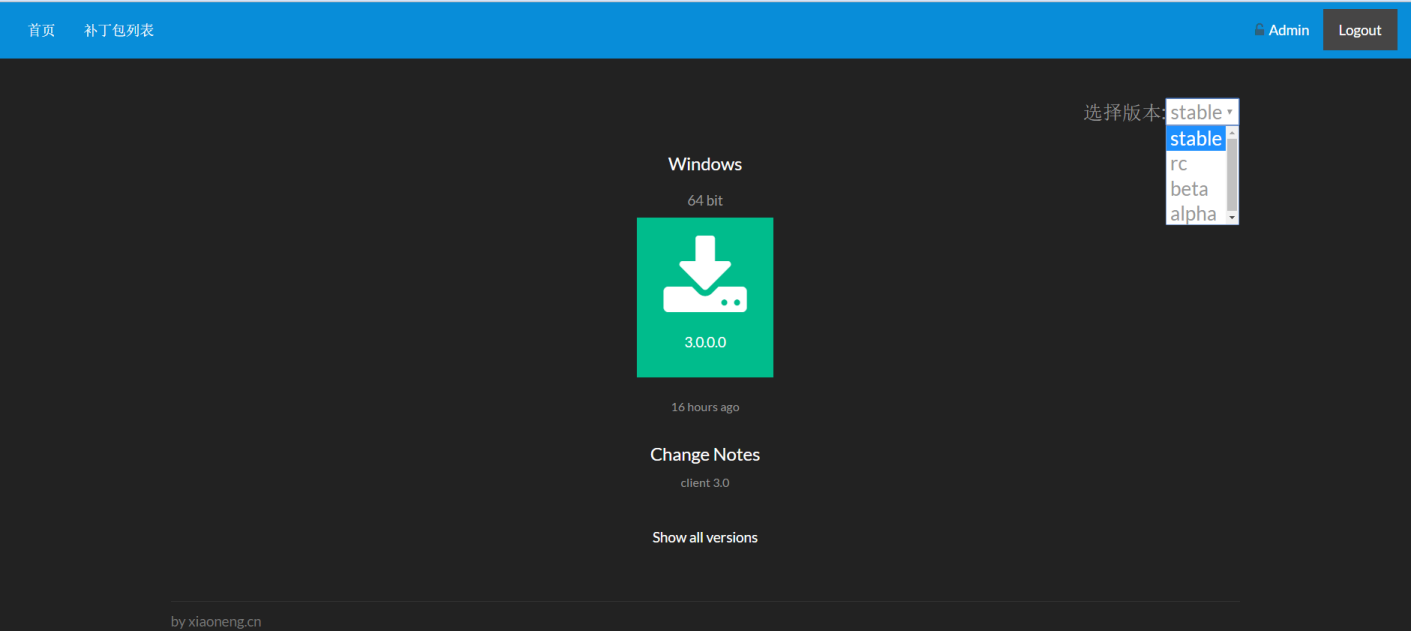
## 升级概要设计（旧版升级流程）



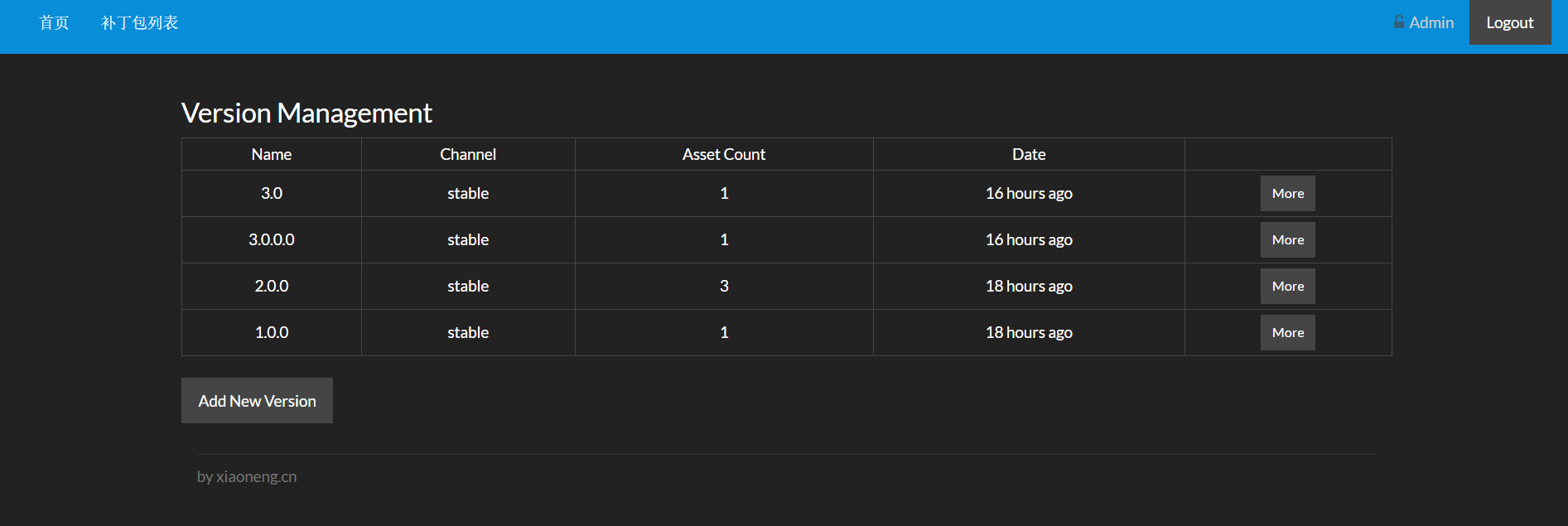
## 方案一：

当前客户端版本信息：package.json中记录当前客户端的版本号

补丁下载



补丁包上传



## 方案二：**Electron autoUpdater实现Windows安装包自动更新（增量更新）**

基本步骤：

electron-prebuilt -> 开发自己的应用 -> electron-packager -> electron-winstall

下面是打包这块的详细解说。

在打包之前我们还需要对我们的应用做一些准备工作。

****建议文件结构****

个人理解这样做的一个原因是将devDependencies和dependencies分开了，另外就是不需要在打包的时候再去指定哪些依赖不需要一起打到安装包里面去了(通过ignore参数)。

首先，我们已经有了一个基于Electron做的应用，参考其他的教程我在项目里面加了两个package.json，目录结构类似于这样：

myapp

-node\_modules

-package.json

-app

-js

-css

-index.html

-main.js

-package.json

外面的package.json内容类似于：

{

"name": "myapp",

"main": "app/main.js",

"scripts": {

"start": "electron ."

},

"devDependencies": {

"electron-prebuilt": "^1.2.7"

}

}

里面的package.json的内容类似于：

{

"name": "myapp",

"version": "1.0",

"main": "main.js",

"description": "my app",

"scripts": {

"start": "electron ."

},

"dependencies": {}

}

注意里面的package.json里面的name，version，description是必填的，接下来打包会用到。

****打包前准备****

为了使最后的安装包能够实现自动更新，我们需要对现有的应用做一些改动，使它可以处理一些启动或者安装时的事件。

我们可以在main.js里面加入一些处理的代码或者方便起见，我们可以直接使用electron-squirrel-startup。

先安装：

npm install electron-squirrel-startup --save

因为需要在main.js里面用到，我们需要将其安装在app里面。

在main.js里面使用它，第一行加入如下代码即可：

if (require('electron-squirrel-startup')) return;

有兴趣的童鞋可以一起跟我去看看electron-squirrel-startup做了什么事情，急着打包的童鞋可以直接忽略这一段：

在myapp/app/node\_modules/electron-squirrel-startup下面有一个index.js：

var path = require('path');

var spawn = require('child\_process').spawn;

var debug = require('debug')('electron-squirrel-startup');

var app = require('electron').app;

var run = function(args, done) {

var updateExe = path.resolve(path.dirname(process.execPath), '..', 'Update.exe');

debug('Spawning `%s` with args `%s`', updateExe, args);

spawn(updateExe, args, {

detached: true

}).on('close', done);

};

var check = function() {

if (process.platform === 'win32') {

var cmd = process.argv[1];

debug('processing squirrel command `%s`', cmd);

var target = path.basename(process.execPath);

if (cmd === '--squirrel-install' || cmd === '--squirrel-updated') {

run(['--createShortcut=' + target + ''], app.quit);

return true;

}

if (cmd === '--squirrel-uninstall') {

run(['--removeShortcut=' + target + ''], app.quit);

return true;

}

if (cmd === '--squirrel-obsolete') {

app.quit();

return true;

}

}

return false;

};

module.exports = check();

打开后我们会发现的它的代码只有短短几十行，做的事情也很简单，注意返回值为true的那几行，基本上来说就是安装时，更新完成时，卸载时main.js都会被调用，我们就需要根据不同的情况做不同的事情，完成这些事情后不要启动应用（会出错），直接退出就好。

正常启动前我们需要去检测是否有新的安装包，之后下载新包，重新安装，重启应用，为了做到这一点，我们需要在main.js里面加入如下代码：

app.on('ready', () => {

//安装后第一次启动不去检测更新，go做的事情就是启动我们的应用

if (process.argv[1] == '--squirrel-firstrun') {

go();

return;

}

/\* 设置自动更新的feedURL，本地测试可以设置为类似于http://127.0.0.1:8080/latest

\* 在latest文件夹下放着三个我们的安装文件(Setup.exe，RELEASES，myapp-1.0-full.nupkg)，下面会讲到

\* /

autoUpdater.setFeedURL(feedURL);

autoUpdater.on('update-downloaded', function() {

// 下载完成，更新前端显示

autoUpdater.quitAndInstall();

});

try {

// 不是安装应用的情况下启动下回出错，此时直接正常启动应用

autoUpdater.checkForUpdates();

} catch (ex) {

go();

return;

}

// createWindow是我们自己定义的方法，用来创建窗口，此处用来创建检测更新的窗口

createWindow({

name: 'updateWindow',

url: 'check-for-updates.html',

title: "checkForUpdates",

icon: icon,

frame: false,

width: 1306,

height: 750

});

});

****搭建我们自己的自动更新后台****

var express = require('express');

var app = express();

app.use(express.static('releases'));

var server = app.listen(8080, function() {

var host = server.address().address

var port = server.address().port

console.log("应用实例，访问地址为 http://%s:%s", host, port);

});

文件结构如下：

autoupdate-backend

-package.json

-index.js

-node\_modules

-releases

-latest

此时latest文件夹里面还是空的，之后我们开始打包，将打包出来的三个文件放在此处即可。

****electron-packager****

在myapp下安装：

npm install electron-packager --save-dev

npm install electron-packager -g

两种安装方式对应两种使用方式，第一种在脚本中使用，第二种的命令行使用。

脚本中使用，我借助了gulp，所以需要安装gulp：

npm install gulp --save-dev

npm install gulp -g

新建GulpFile.js，给我们自己定义一个task：

var gulp = require('gulp');

var platform = 'win32';

var arch = 'ia32';

var appPath = 'app';

var packageOutPath = 'production/package';

var iconPath = 'app/favicon.ico';

gulp.task('generate-package', () => {

generatePackage();

});

function generatePackage(callback) {

var packager = require('electron-packager')

packager({

dir: appPath,

platform: platform,

arch: arch,

out: packageOutPath,

icon: iconPath,

/\*桌面快捷方式名称以及开始菜单文件夹名称\*/

'version-string': {

CompanyName: 'MyCompany Inc.',

ProductName: 'myapp'

}

}, function(err) {

if (err) {

console.log(err);

} else {

callback && callback();

}

});

}

需要打包的时候，打开命令行：

gulp generate-package

这样做的好处是调用方便，当然我们也可以直接通过命令行调用electron-packager，前提是我们全局安装了它或者将其安装目录添加到了环境变量中:

electron-package ./app --platform=win32 --arch=ia32 --icon="app/favicon.ico" --out="productin/package" --version-string.CompanyName="MyCompany Inc." --version-string.ProductName="myapp"

更多参数一一加上即可。

贴上官方文档链接：

github链接：<https://github.com/electron-userland/electron-packager>

下面两个链接在上面的文档里面都能找到，但是个人感觉比较常用，还是贴出来：

参数使用：<https://github.com/electron-userland/electron-packager/blob/master/usage.txt>

脚本使用：<https://github.com/electron-userland/electron-packager/blob/master/docs/api.md>

****打包啦****

myapp下安装electron-winstaller：

npm install electron-winstaller --save-dev

还是在gulp里面添加一个task，连同package的代码一起贴上：

var gulp = require('gulp');

var platform = 'win32';

var arch = 'ia32';

var appPath = 'app';

var outName = 'myapp-win32-' + arch;

var packageOutPath = 'production/package';

var installerOutPath = 'production/installer';

var packagePath = `${packageOutPath}/${outName}`;

var installerPath = `${installerOutPath}/${outName}`;

var iconPath = 'app/favicon.ico';

var gifPath = 'loading.gif';

gulp.task('generate-package', () => {

generatePackage();

});

gulp.task('generate-installer', () => {

isDirExist(packagePath, (exist) => {

if (exist) {

generateInstaller();

} else {

generatePackage(() => {

generateInstaller();

});

}

});

});

function isDirExist(path, callback) {

fs.readdir(path, (err) => {

callback && callback(!err);

});

}

function generatePackage(callback) {

var packager = require('electron-packager')

packager({

dir: appPath,

platform: platform,

arch: arch,

out: packageOutPath,

icon: iconPath,

/\*桌面快捷方式名称以及开始菜单文件夹名称\*/

'version-string': {

CompanyName: 'MyCompany Inc.',

ProductName: 'myapp'

}

}, function(err) {

if (err) {

console.log(err);

} else {

callback && callback();

}

});

}

function generateInstaller() {

var electronInstaller = require('electron-winstaller');

electronInstaller.createWindowsInstaller({

appDirectory: packagePath,

outputDirectory: installerPath,

loadingGif: gifPath,

authors: 'ganyouyin',

exe: 'myapp.exe',

title: 'My APP',

iconUrl: `${\_\_dirname}/${iconPath}`,

setupIcon: iconPath,

setupExe: 'Setup.exe',

noMsi: true

}).then(() => console.log("It worked!"), (e) => console.log(`No dice: ${e.message}`));

}

之后执行任务：

gulp generate-installer

第一次会非常慢，但是执行完成后我们的安装包就出来了。

此时我们的文件结构是：

myapp

-GulpFile.js

-package.json

-node\_modules

-app

-production

-package

-myapp-win32-ia32

- 各种文件，包含一个myapp.exe，双击可以直接运行

-installer

-myapp-win32-ia32

-Setup.exe

-RELEASES

-myapp-1.0-full.nupkg

有了三个文件，将他们粘到之前的autoupdate-backend/releases/latest文件夹下面。

****测试自动更新包****

0 启动我们的自动更新后台；

1 将myapp/app下的package.json里面的version改为1.1，再次打包；

2 将之前的autoupdate-backend中的latest文件夹重命名为1.0；

3 新建文件夹latest，将新打包产生的三个文件粘进去；

4 双击1.0里面的Setup.exe安装应用；

5 关闭应用，双击桌面上的快捷方式myapp.exe再次打开应用;

不出意外此时就会去进行自动更新的操作，结束后自动重启，再次打开时已经是1.1的应用。