<http://www.expressjs.com.cn/>

原生Node开发，会发现有很多问题。比如：

呈递静态页面很不方便，需要处理每个HTTP请求，还要考虑304问题

路由处理代码不直观清晰，需要写很多正则表达式和字符串函数

不能集中精力写业务，要考虑很多其他的东西

Express：基于 [Node.js](https://nodejs.org/en/) 平台，快速、开放、极简的 Web 开发框架

Web 应用程序

Express 是一个保持最小规模的灵活的 Node.js Web 应用程序开发框架，为 Web 和移动应用程序提供一组强大的功能

API

使用您所选择的各种 HTTP 实用工具和中间件，快速方便地创建强大的 API。

性能

Express 提供精简的基本 Web 应用程序功能，而不会隐藏您了解和青睐的 Node.js 功能。

框架

许多流行的开发框架都基于 Express 构建。

**安装**

首先假定你已经安装了 [Node.js](https://nodejs.org/)，接下来为你的应用创建一个目录，然后进入此目录并将其作为当前工作目录。

$ mkdir myapp

$ cd myapp

通过 npm init 命令为你的应用创建一个 package.json 文件。 欲了解 package.json 是如何起作用的，请参考 [Specifics of npm’s package.json handling](https://docs.npmjs.com/files/package.json).

$ npm init

此命令将要求你输入几个参数，例如此应用的名称和版本。 你可以直接按“回车”键接受大部分默认设置即可，下面这个除外：

entry point: (index.js)

键入 app.js 或者你所希望的名称，这是当前应用的入口文件。如果你希望采用默认的 index.js 文件名，只需按“回车”键即可。

接下来在 myapp 目录下安装 Express 并将其保存到依赖列表中。如下：

$ npm install express --save

如果只是临时安装 Express，不想将它添加到依赖列表中，可执行如下命令：

$ npm install express --no-save

<https://blog.csdn.net/weixin_41646716/article/details/82460761>

### 中间件

简单说，中间件（middleware）就是处理HTTP请求的函数。它最大的特点就是，一个中间件处理完，再传递给下一个中间件。App实例在运行过程中，会调用一系列的中间件。

每个中间件可以从App实例，接收三个参数，依次为request对象（代表HTTP请求）、response对象（代表HTTP回应），next回调函数（代表下一个中间件）。每个中间件都可以对HTTP请求（request对象）进行加工，并且决定是否调用next方法，将request对象再传给下一个中间件。

一个不进行任何操作、只传递request对象的中间件，就是下面这样。

1. function uselessMiddleware(req, res, next) {
2. next();
3. }

上面代码的next就是下一个中间件。如果它带有参数，则代表抛出一个错误，参数为错误文本。

1. function uselessMiddleware(req, res, next) {
2. next('出错了！');
3. }

抛出错误以后，后面的中间件将不再执行，直到发现一个错误处理函数为止。

### use方法

use是express注册中间件的方法，它返回一个函数。下面是一个连续调用两个中间件的例子。

1. var express = require("express");
2. var http = require("http");
4. var app = express();
6. app.use(function(request, response, next) {
7. console.log("In comes a " + request.method + " to " + request.url);
8. next();
9. });
11. app.use(function(request, response) {
12. response.writeHead(200, { "Content-Type": "text/plain" });
13. response.end("Hello world!\n");
14. });
16. http.createServer(app).listen(1337);

上面代码使用app.use方法，注册了两个中间件。收到HTTP请求后，先调用第一个中间件，在控制台输出一行信息，然后通过next方法，将执行权传给第二个中间件，输出HTTP回应。由于第二个中间件没有调用next方法，所以request对象就不再向后传递了。

use方法内部可以对访问路径进行判断，据此就能实现简单的路由，根据不同的请求网址，返回不同的网页内容。

1. var express = require("express");
2. var http = require("http");
4. var app = express();
6. app.use(function(request, response, next) {
7. if (request.url == "/") {
8. response.writeHead(200, { "Content-Type": "text/plain" });
9. response.end("Welcome to the homepage!\n");
10. } else {
11. next();
12. }
13. });
15. app.use(function(request, response, next) {
16. if (request.url == "/about") {
17. response.writeHead(200, { "Content-Type": "text/plain" });
18. } else {
19. next();
20. }
21. });
23. app.use(function(request, response) {
24. response.writeHead(404, { "Content-Type": "text/plain" });
25. response.end("404 error!\n");
26. });
28. http.createServer(app).listen(1337);

上面代码通过request.url属性，判断请求的网址，从而返回不同的内容。注意，app.use方法一共登记了三个中间件，只要请求路径匹配，就不会将执行权交给下一个中间件。因此，最后一个中间件会返回404错误，即前面的中间件都没匹配请求路径，找不到所要请求的资源。

除了在回调函数内部判断请求的网址，use方法也允许将请求网址写在第一个参数。这代表，只有请求路径匹配这个参数，后面的中间件才会生效。无疑，这样写更加清晰和方便。

app.use('/path', someMiddleware);

上面代码表示，只对根目录的请求，调用某个中间件。

因此，上面的代码可以写成下面的样子。

1. var express = require("express");
2. var http = require("http");
4. var app = express();
6. app.use("/home", function(request, response, next) {
7. response.writeHead(200, { "Content-Type": "text/plain" });
8. response.end("Welcome to the homepage!\n");
9. });
11. app.use("/about", function(request, response, next) {
12. response.writeHead(200, { "Content-Type": "text/plain" });
13. response.end("Welcome to the about page!\n");
14. });
16. app.use(function(request, response) {
17. response.writeHead(404, { "Content-Type": "text/plain" });
18. response.end("404 error!\n");
19. });
21. http.createServer(app).listen(1337);

## Express的方法

### all方法和HTTP动词方法

针对不同的请求，Express提供了use方法的一些别名。比如，上面代码也可以用别名的形式来写。

app.all("\*", function(request, response, next) {

response.writeHead(200, { "Content-Type": "text/plain" });

next();

});

app.get("/", function(request, response) {

response.end("Welcome to the homepage!");

});

上面代码的all方法表示，所有请求都必须通过该中间件，参数中的“\*”表示对所有路径有效。get方法则是只有GET动词的HTTP请求通过该中间件，它的第一个参数是请求的路径。由于get方法的回调函数没有调用next方法，所以只要有一个中间件被调用了，后面的中间件就不会再被调用了。

除了get方法以外，Express还提供post、put、delete方法，即HTTP动词都是Express的方法。这些方法的第一个参数，都是请求的路径。除了绝对匹配以外，Express允许模式匹配。

### set方法

set方法用于指定变量的值。

1. app.set("views", \_\_dirname + "/views");
3. app.set("view engine", "jade");

上面代码使用set方法，为系统变量“views”和“view engine”指定值

### response对象

**（1）response.redirect方法**

response.redirect方法允许网址的重定向。

1. response.redirect("/hello/anime");
2. response.redirect("http://www.example.com");
3. response.redirect(301, "http://www.example.com");

**（2）response.sendFile方法**

response.sendFile方法用于发送文件。

response.sendFile("/path/to/anime.mp4");

**（3）response.render方法**

response.render方法用于渲染网页模板。

1. app.get("/", function(request, response) {
2. response.render("index", { message: "Hello World" });
3. });

上面代码使用render方法，将message变量传入index模板，渲染成HTML网页。

### requst对象

**（1）request.ip**

request.ip属性用于获得HTTP请求的IP地址。

**（2）request.files**

request.files用于获取上传的文件。

## 项目开发实例

（也在这个网址下）