1 概述

1.1 简介：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。

1.2 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。

1.3 安装：npm install

1.4 以下几个重要的模块是需要与 express 框架一起安装的：
- body-parser - node.js 中间件，用于处理 JSON, Raw, Text 和 URL 编码的数据。
- cookie-parser - 这就是一个解析Cookie的工具。通过req.cookies可以取到传过来的cookie，并把它们转成对象。
- multer - node.js 中间件，用于处理 enctype="multipart/form-data"（设置表单的MIME编码）的表单数据。

1.5 hello world

```
//hello world

var express = require('express');
var app = express();

app.get('/', function (req, res) {
   res.send('Hello World');
})

var server = app.listen(8081, function () {
   var host = server.address().address
   var port = server.address().port
   console.log('Example app listening at http://%s:%s', host, port);
});
```

```
//启动一个服务并监听从3000端口进入的所有连接请求，它将所有(/)URL或路由返回“Hello World”字符串
```

1.6 Express 应用生成器： 通过应用生成器工具 express 可以快速创建一个应用的骨架。

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

在当前工作目录下创建一个命名为 *myapp* 的应用。

```
$ express myapp


   create : myapp
   create : myapp/package.json
   create : myapp/app.js
   create : myapp/public
   create : myapp/public/javascripts
   create : myapp/public/images
   create : myapp/routes
   create : myapp/routes/index.js
   create : myapp/routes/users.js
   create : myapp/public/stylesheets
   create : myapp/public/stylesheets/style.css
   create : myapp/views
   create : myapp/views/index.jade
   create : myapp/views/layout.jade
   create : myapp/views/error.jade
   create : myapp/bin
   create : myapp/bin/www
```

安装所有依赖包后便可以运行

```
$ cd myapp
$ npm install

## 启动这个应用（MacOS 或 Linux 平台）：
$ DEBUG=myapp npm start

## Windows 平台使用如下命令：
> set DEBUG=myapp & npm start
```

在浏览器中打开http://localhost:3000/就可以看到这个应用

通过 Express 应用生成器创建的应用一般都有如下目录结构：

```
.
├── app.js
├── bin
│   └── www
├── package.json
├── public
│   ├── images
│   ├── javascripts
│   └── stylesheets
│       └── style.css
├── routes
│   ├── index.js
│   └── users.js
└── views
    ├── error.jade
    ├── index.jade
    └── layout.jade

7 directories, 9 files
```

2 http服务端

2.1 发送一个http get请求

```
app.get('/', function (res ,req)  {
  // ...
})
```

2.2 **Request对象：**request 对象表示 HTTP 请求，包含了请求查询字符串，参数，内容，HTTP 头部等属性。

属性	说明
req.app	当callback为外部文件时，用req.app访问express的实例
req.baseUrl	获取路由当前安装的URL路径
req.body / req.cookies	获得「请求主体」/ Cookies

req.fresh / req.stale	判断请求是否还「新鲜」
req.hostname / req.ip	获取主机名和IP地址
req.originalUrl	获取原始请求URL
req.params	获取路由的parameters
req.path	获取请求路径
req.protocol	获取协议类型
req.query	获取URL的查询参数串
req.route	获取当前匹配的路由
req.subdomains	获取子域名
req.accepts()	检查可接受的请求的文档类型
req.acceptsCharsets req.acceptsEncodings req.acceptsLanguages	返回指定字符集的第一个可接受字符编码
req.get()	获取指定的HTTP请求头
req.is()	判断请求头Content-Type的MIME类型

2.3 Response 对象：response 对象表示 HTTP 响应，即在接收到请求时向客户端发送的 HTTP 响应数据。

属性	描述
res.app	同req.app一样
res.append()	追加指定HTTP头
res.set()	在res.append()后将重置之前设置的头
res.cookie(name，value [，option])	设置Cookie
opition	domain / expires / httpOnly / maxAge /
res.clearCookie()	清除Cookie
res.download()	传送指定路径的文件
res.get()	返回指定的HTTP头

res.json()	传送JSON响应
res.jsonp()	传送JSONP响应
res.location()	只设置响应的Location HTTP头，不设置状态
res.redirect()	设置响应的Location HTTP头，并且设置状态
res.send()	传送HTTP响应
res.sendFile(path [，options] [，fn])	传送指定路径的文件 -会自动根据文件exten
res.set()	设置HTTP头，传入object可以一次设置多个
res.status()	设置HTTP状态码
res.type()	设置Content-Type的MIME类型

2.4 **路由**：路由决定了由谁(指定脚本)去响应客户端请求，在HTTP请求中，我们可以通过路由提取出请求的URL以及GET/POST参数

实现方法：为不同的url路径设置不同的响应

```
'use strict';
const express = require('express');
const app = express();

app.get('/', (req, res) => {
   console.log("主页 GET 请求");
   res.send('Hello GET');
});

app.get('/del_user', (req, res) => {
   console.log("/del_user 响应 DELETE 请求");
   res.send('删除页面');
});

app.get('/list_user', function (req, res) {
   console.log("/list_user GET 请求");
```

```
    res.send('用户列表页面');
})


app.get('/ab*cd', function (req, res) {
    console.log("/ab*cd GET 请求");
    res.send('正则匹配');
});
```

3 静态文件

Express 提供了内置的中间件 express.static 来设置静态文件如：图片， CSS, JavaScript 等。

如果在public文件夹有所需要的静态文件，可以引入：

```
app.use(express.static('public'));
```

访问：http://localhost:9000/file.txt

如果静态资源存放在多个目录下面，可以多次调用 express.static

访问静态资源文件时，express.static 中间件会根据目录添加的顺序查找所需的文件。

如果你希望所有通过 express.static 访问的文件都存放在一个“虚拟（virtual）”目录（即目录根本不存在）下面，可以通过为静态资源目录指定一个挂载路径的方式来实现:

```
app.use('/static', express.static('public'));
```

现在，你就爱可以通过带有 “/static” 前缀的地址来访问 public 目录下面的文件了。

```
http://localhost:3000/static/images/kitten.jpg
http://localhost:3000/static/css/style.css
http://localhost:3000/static/js/app.js
http://localhost:3000/static/images/bg.png
http://localhost:3000/static/hello.html
```

4 中间件

4.1 概述：Express 是一个自身功能极简，完全是由路由和中间件构成一个的 web 开发框架：从本质上来说，一个 Express 应用就是在调用各种中间件。

4.2 简介：中间件（Middleware） 是一个函数，它可以访问请求对象（*request object (req)*），响应对象（*response object (res)*），和 web 应用中处于请求-响应循环流程中的中间件，一般被命名为 next 的变量。

4.3 中间件的功能：
- 执行任何代码
- 修改请求和相应对象
- 终结请求-响应循环
- 调用堆栈中的下一个中间件

4.4 Express应用可使用的几种中间件：
- 应用级中间件
- 路由级中间件
- 错误处理中间件
- 内置中间件
- 第三方中间件

4.5 应用级中间件： 应用级中间件绑定到 app 对象 使用 app.use() 和 app.METHOD()，其中， METHOD 是需要处理的 HTTP 请求的方法，例如 GET, PUT, POST 等等，全部小写。

4.6 路由级中间件：和应用级中间件一样，只是它绑定的对象为 express.Router();
router.use(PATH, HANDLERS);

4.7 错误处理中间件： 定义错误处理中间件和定义其他中间件一样，除了需要 4 个参数，而不是 3 个，其格式如下 app.use(err, req, res, next)。

4.8 内置中间件
从 4.x 版本开始，, Express 已经不再依赖 Connect 了。除了 express.static, Express 以前内置的中间件现在已经全部单独作为模块安装使用了。express.static(root, [options]);

4.9 第三方中间件

通过使用第三方中间件从而为 Express 应用增加更多功能。
安装所需功能的 node 模块，并在应用中加载，可以在应用级加载，也可以在路由级加载。　　app.use(MODULES)

5 模版引擎

app.set('views', './views'); 设置模版文件目录
app.set('view engine', 'jade'); 设置模版引擎

res.render('index', {'title': 'Hey'});　使用模版引擎渲染带参数的动态页面

6 调试

Express 内部使用 debug 模块记录路由匹配、使用到的中间件、应用模式以及请求-响应循环。

```
debug 有点像改装过的 console.log，不同的是，您不需要在生产代码中注释掉 debug。它会默认关闭，而且使用一个名为 DEBUG 的环境变量还可以打开。
```

在启动应用时，设置 DEBUG 环境变量为 express:*，可以查看 Express 中用到的所有内部日志。

```
# Linux
$ DEBUG=express:* node index.js

# Windows
> set DEBUG=express:* & node index.js
```

设置 DEBUG 的值为 express:router，只查看路由部分的日志；设置 DEBUG 的值为 express:application，只查看应用部分的日志，依此类推。

设置环境变量
process.env.NODE_ENV;　　设置运行环境变量
process.env.prot;　　设置环境变量端口

---

express渲染变量数据

app.locals 和 res.locals

express 中有两个对象可用于模板的渲染：app.locals 和 res.locals。

res.render 的时候，express 合并（merge）了 3 处的结果后传入要渲染的模板

优先级：res.render 传入的对象> res.locals 对象 > app.locals 对象

app.locals 和 res.locals 几乎没有区别，都用来渲染模板

使用上的区别在于：
app.locals 上通常挂载常量信息（如博客名、描述、作者这种不会变的信息）
res.locals 上通常挂载变量信息，即每次请求可能的值都不一样（如请求者信息，
res.locals.user = req.session.user）。

中间件

详细内容

其他

文件上传
COOKIE管理
模板引擎