Express

> Express 是一个简洁而灵活的 node.js Web应用框架, 提供了强大特性帮助我们创建各种Web应用。
>
> Express 不对 node.js 已有的特性进行封装，只是在它之上增加了扩展功能。丰富的HTTP工具以及强大的中间件可以帮助我们快速开发。
>
> 更多参考：`http://www.expressjs.com.cn/`

本地安装

> express 是nodejs的框架，依赖于node，需要提起安装node环境
> 安装完node之后，我们可以通过 `npm init -y` 创建一个 package.json 文件
>
> 本地安装
>
> - `npm install express -S`

使用Express

> 1. 引入express模块
> 2. 调用express函数，得到一个express实例
> 3. 创建一个路由，返回 “Hello World” 字符串
> 4. 启动服务，监听8080端口

```
let express = require('express');
let app = express();

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

app.listen(8080);
```

Express 路由

> 路由是指如何定义URL以及如何响应客户端的请求
>
> 路由是由一个 URI、HTTP 请求（GET、POST等）和若干个句柄组成的，它的结构是 `app.METHOD(path, [callback...], callback)`
>
> - app 是 express 对象的一个实例
> - METHOD 是一个 HTTP 请求方法
> - path 是服务器上的路径
> - callback 是当路由匹配时要执行的函数。

路由的方法

> 路由方法源于 HTTP 请求方法，有 GET、POST、PUT、DELETE等

GET请求

> app.get(path,function(request, response));
>
> - 第一个参数path 是请求的路径
> - 第二个参数是请求的回调函数，里面包含两个参数，request代表请求信息，response代表响应信息

```
/**
 * 当客户端浏览器通过 get方法访问服务器/路径的时候，会由对应的监听函数来进行处理
 * 当服务器收到客户端请求后，先由app来进行处理，app不做具体响应请求，而是进行判断应该由那个路由来处理
 */
app.get('/',function (req, res) {
    res.end('home');
});

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

ALL方法

> app.all() 是一个特殊的路由方法，可以匹配到所有的 HTTP 方法（这里指的是可以匹配GET、POST、PUT等所有方法）请求，它的作用是对于一个路径上的所有请求加载中间件。
>
> app.all('*') 可以匹配所有的路径请求，我们可以利用此特性来进行一些错误异常处理等公共操作，`需要放到所有路由的最下面`

```
//--> 表示匹配所有selete的请求
app.all('/selete',function (req, res, next) {
    console.log('selete');
    next();
})

//--> 表示匹配所有的方法和所有的路径
app.all('*',function (req, res) {
    res.end('404');
})
```

路径参数

> req.params可以用来获取请求URL中的参数值

```
app.get('/users/:id',function (req, res) {
    let id = req.params.id;

    res.end(id);
})
```

获取请求参数

> req.method 获取请求方法
> req.url 获取请求url
> req.path 获取请求的URL的路径名
> req.query 获取查询字符串
> req.headers 获取请求头对象
> req.host 返回请求头里取的主机名(不包含端口号)

```
app.get('/signup',function (req, res) {
    //--> req和以前用的http模块里的req对象是一个对象，只不过多了一些方法和属性
    console.log(req.method);
    console.log(req.url);//url地址

    console.log(req.path); //路径
    console.log(req.query); //查询字符串，是个对象{}

    console.log(req.headers); // 请求头对象

    //res.header <=> res.setHeader 设置请求头
    res.header('Content-Type','text/html;charset=utf8')
    res.end(`
        <form action="/signup" method="post">
            用户名：<input type="text" name="username">
            密码：<input type="password" name="pass">

            <input type="submit" value="提交">
        </form>`);
});
```

curl使用

> 如果有一天，你忘记了请求和响应的格式。 打开gitbash或者cmd，使用curl命令可以查看请求和响应的格式

`-H 指定请求头`

```
curl -H 'content-type:application/json;charset=utf-8' http://localhost:8080/users
```

`-X POST 指定请求方法`

```
curl -X POST http://localhost:8080/users
```

`--data 指定请求体`

```
curl --data "name=zfpx&age=8" http://localhost:8080/users
```

例如

```
curl --verbose http://localhost:8080/user
```

```
C:\Users\LZW>curl --verbose http://localhost:8080/user
* timeout on name lookup is not supported
*   Trying ::1...
* Connected to localhost (::1) port 8080 (#0)
> GET /user HTTP/1.1
> Host: localhost:8080
> User-Agent: curl/7.46.0
> Accept: */*
>
< HTTP/1.1 200 OK
< X-Powered-By: Express
< Date: Sat, 28 Oct 2017 03:26:35 GMT
< Connection: keep-alive
< Content-Length: 3
<
404* Connection #0 to host localhost left intact
```

响应方法

> 响应对象（res）的方法向客户端返回响应，结束请求。如果在路由不没有结束响应，那么这个请求会一直挂起。

方法	描述
res.download()	提示下载文件。
res.end()	终结响应处理流程。
res.json()	发送一个 JSON 格式的响应。
res.jsonp()	发送一个支持 JSONP 的 JSON 格式的响应。
res.redirect()	重定向请求。
res.render()	渲染视图模板。
res.send()	发送各种类型的响应。
res.sendFile	以八位字节流的形式发送文件。
res.sendStatus()	设置响应状态代码，并将其以字符串形式作为响应体的一部分发送。

send 方法

send方法向浏览器发送一个响应信息，并可以智能处理不同类型的数据 并在输出响应时会自动进行一些设置，比如HEAD信息、HTTP缓存支持等
语法：`res.send([body|status], [body])`

- 当参数为一个String时，Content-Type默认设置为”text/html”
- 当参数为Array或Object时，Express会返回一个JSON
- 当参数为一个Number时，并且没有上面提到的任何一条在响应体里，Express会帮你设置一个响应体，比如：200会返回字符”OK”

```
app.get('/users',function (req, res) {
    //--> send会进行数据类型转换，把其它类型都转成end能处理的类型（字符串/buffer）
    res.send([{id:1,name:'名称'}]);
})

//sendfile 发送文件
app.get('/users/json',function (req, res) {
    //--> 路径必须是绝对路径，或者指定root根目录
    // res.sendfile(path.resolve('./users.json'));
    //指定 root根目录
    res.sendfile('./users.json',{root:__dirname});
})

// 返回一个状态 404
app.get('/other',function (req, res) {
    //--> res.status(404) <> res.statusCode = 404
    res.status(404); //设置状态码

    // sendStatus 不仅发送状态码还结束了请求
    res.sendStatus(404);
})
```

中间件

express 的一个强大之处就在于可以调用各种中间件

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 处于请求-响应流程中的中间件（一般被命名为 `next` 的变量）

中间件的功能

- 执行任何代码。
- 修改请求和响应对象。
- 终结请求-响应循环。
- 调用堆栈中的下一个中间件

`如果当前中间件没有结束请求响应，则必须调用 next() 方法将控制权交给下一个中间件，否则请求就会挂起`

Express 中的中间件包含

- 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

应用中间件

应用级中间件绑定到 app 对象上，app.use() 使用中间件。

```
1. // 挂载至 /user/:id 的中间件，任何指向 /user/:id 的请求都会执行它
2. app.use('/user/:id', function (req, res, next) {
3.   console.log('Request Type:', req.method);
4.   next();
5. });
6.
```

路由中间件

路由中间件和应用中间件一样，只是它绑定的对象为 express.Router()
用 `app.use('路由路径', 要使用的路由中间件)` 使用中间件

```
let express = require('express');

//-->调用Router方法会返回一个路由对象
let router = express.Router();

// 路由中间件的用法和app很像
router.get('/signup',function (req, res) {
    res.send('注册');
});

router.post('/signup',function (req, res) {
    res.send('提交注册');
});

//--> 使用路由中间件,当服务器接收到客户端请求的时候会判断请求路径是不是以 /user开头的，如果是以/user 开头的，会交给这个中间件来处理
app.use('/user', router);
```

错误处理中间件

> 错误处理中间件有 4 个参数，定义错误处理中间件时必须使用这 4 个参数。即使不需要 next 对象，也必须声明它，否则中间件会被识别为一个常规中间件，不能处理错误。

```
// 没有挂载路径的中间件，应用的每个请求都会执行该中间件
app.use(function (req,res,next) {
    console.log('中间件');
    res.header('Content-Type','text/html;charset=utf8');

    // 读取一个文件的内容，并且把读取到的内容赋值给 req.msg
    fs.readFile('2.txt','utf8',function (err, data) {
        // -->如果在中间件里出错的话，应该交给统一的处理函数来处理
        if(err) {
            // -->同样调用next方法继续执行，但是会传入错误对象
            // --> 如果next的参数不为null 就表示有错误了，则会跳过正常的业务逻辑，交由错误处理中间件来处理
            next(err);
        }else {
            req.msg = data;
            next();
        }

    });
});

/**
 * 错误处理中间件，多了一个err参数
 */
app.use(function (err, req, res, next) {
    console.log(err);
    res.send('404');
});
```

内置中间件

从 4.x 版本开始，`express.static` 是 Express 唯一内置的中间件,负责处理静态资源

语法 **`express.static(root, [options])`**

- root 指提供静态资源的根目录
- options 参数配置项，可参考 serve-static

```
let express = require('express');
let path = require('path');
let app = express();

/**
 * 客户端访问的路径如果是静态文件根目录的子路径 ，中间件接收到请求后，先取到静态文件的根目录，然后拼上客户端访问的路径，会得到文件的绝对路径，再通过fs模块读取此文件路径，将内容返回给客户端
 */
app.use(express.static(path.resolve('public')));
//--> 可以使用多个静态目录
app.use(express.static(path.resolve('uploads')));
app.use(express.static(path.resolve('files')));
```

第三方中间件

> 安装所需功能的 node 模块，通过require 引入并使用
> 常用的第三方中间件请参考
> http://www.expressjs.com.cn/resources/middleware.html

```
//用来解析请求体的中间件，它会把请求体的数据变成一个对象赋给req.body
let bodyParser = require('body-parser');
// 用于解析 cookie 的中间件
let cookieParser = require('cookie-parser');
// 用于处理 session的中间件
let session = require('express-session');

//使用bodyParser中间件，并解析json格式数据
app.use(bodyParser.json());
//使用cookieParser中间件
app.use(cookieParser());
//使用session中间件
app.use(session({
    resave: true,
    saveUninitialized: true,
    secret: 'ping'
}));

```

next方法

> next方法将控制权交给下一个路由，next 的内部实现原理就是递归调用

```
//简述next的实现原理
let stack = [
    function (req, res, next) {
        console.log(1);
        next();
    },
    function (req, res, next) {
        console.log(2);
        next();
    },
    function (req, res, next) {
        console.log(3);
        next();
    }
];

let index = 0;
function next() {
    var fn = stack[index++];
    fn && fn(null,null,next);
}
```

模板引擎

> 在nodejs中使用express框架，它默认的是ejs和jade渲染模板

ejs

安装ejs

> `npm install ejs -S`

使用ejs模板

> 1. 设置模板引擎
> 2. 设置模板存放的根路径
> 3. 渲染模板文件

```
let express = require('express');
// path模块：处理文件与目录的路径
let path = require('path');

let app = express();

//-->设置模板引擎
//--> 模板引擎的值 和 模板文件的后缀要一致，这里指定模板文件的后缀名为 ejs
app.set('view engine','ejs');
//-->设置模板存放的根路径
app.set('views',path.resolve('views'));

app.get('/',function (req, res) {
    //--> '.' 指的是模板存放的根目录，而非当前模板所在目录
    //--> 后缀可以不写，找模板的时候会自动添加后缀 .ejs
    res.render('./index',{title:'首页',users:[{id:1,name:'小花'},{id:2,name:'小明'}],msg:'<h1>hello</h1>'});
});
```

使用html后缀模板

> 使用html后缀模板，需要增加一步操作，指定使用ejs模板引擎来渲染模板

```
let express = require('express');
let path = require('path');

let app = express();

//-->设置模板引擎
app.set('view engine','html');
//-->设置模板存放的根路径
app.set('views',path.resolve('views'));
// -->指定html类型的模板使用ejs模板引擎来进行渲染
app.engine('html', require('ejs').__express)

let data = {
    title:'首页',
    users:[{id:1,name:'小花'},{id:2,name:'小明'}],
    msg:'<h1>hello</h1>'
};

app.get('/',function (req, res) {
    res.render('./index',data);
});
```

页面使用模板数据

> 变量使用：<%=变量名%>

```
<ul>
    <%
        for(let i = 0; i<users.length; i++){
            let user = users[i]; %>
             <li><%=user.name%></li>
    <%
        }
    %>
</ul>
<%-msg%>
```