什么是 express?

Express 是基于 Node.js 平台，快速、开放、极简的 Web 开发框架它提供一系列强大的特性，帮助你创建各种 Web 和移动设备应用。

EXPRESS 特点

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。

快速入门

安装 EXPRESS

首先假设您已经安装了 node，新建一个项目目录
mkdir myExpress ,
cd myExpress
npm init
接下来在 myapp 目录下安装 Express 并将其保存到依赖列表中
npm install express --save

以上命令会将 Express 框架安装在当前目录的 node_modules 目录中， node_modules 目录下会自动创建 express 目录。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。

npm install body-parser cookie-parser multer --save

第一个例子 Hello world

新建 index.js 配置文件，代码如下

```
创建一个 Express 应用。express()是一个 express 模块导出的入口函数
const express = require('express')
const app = express()
app.get('/', (req, res) => res.send('Hello World!'))
app.listen(3003, () => console.log('Example app listening on port 3003!'))
```

在控制台输入命令， 运行

```
node index.js
```

在浏览器输入： http://localhost:3003/ 查看

请求和响应

Express 应用使用回调函数的参数： request 和 response 对象来处理请求和响应的数据。

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

request 和 response 对象的具体介绍：

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。常见属性有：

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

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。

常见属性有:

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 / path / secure / signed
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，不设置状态码或者 close response
res.redirect()：设置响应的 Location HTTP 头，并且设置状态码 302
res.render(view,[locals],callback)：渲染一个 view，同时向 callback 传递渲染后的字符串，如果在渲染过程中有错误发生 next(err)将会被自动调用。callback 将会被传入一个可能发生的错误以及渲染后的页面，这样就不会自动输出了。
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 extension 设定 Content-Type
res.set()：设置 HTTP 头，传入 object 可以一次设置多个头
res.status()：设置 HTTP 状态码
res.type()：设置 Content-Type 的 MIME 类型

编写用于 Express 应用程序的中间件

中间件函数是能够访问请求对象(req)、响应对象(res)和应用程序请求-响应周期中的 next 函数的函数。next 函数是 Express 路由器中的一个函数，当调用该函数时，它将执行当前中间件之后的中间件。

中间件功能可以执行以下任务:

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

如果当前中间件函数没有结束请求-响应周期，则必须调用 next()将控制权传递给下一个中间件函数。否则，请求将挂起。

路由器级别中间件 express.Router() 上面已讲

错误处理中间件

内置的中间件:

从版本 4 开始。x, Express 不再依赖于 Connect。以前包含在 Express 中的中间件功能现在位于单独的模块中

第三方中间件 ：

如 cookie-parser，需要单独安装使用

```
var express = require('express')
var app = express()
var cookieParser = require('cookie-parser')
```

```
// load the cookie-parsing middleware
app.use(cookieParser())
```

路由

路由是指确定应用程序如何响应到特定端点的客户机请求，该端点是 URI(或路径)和特定的 HTTP 请求方法(GET、POST 等)。
每个路由可以有一个或多个处理程序函数，这些函数在匹配路由时执行。

app.METHOD(PATH, HANDLER)

说明:
app 是 express 的一个实例。
METHOD 是 HTTP 请求方法(小写)，比如 post,get,put 等。
PATH 是服务器上的路径。
HANDLER 是在匹配路由时执行的函数。

```
app.get('/ab?cd', function (req, res) {
  res.send('ab?cd')
})
```

```
app.get('/users/:userId/books/:bookId', function (req, res) {
  res.send(req.params)
})
```

路由处理程序 -- Route handlers

您可以提供多个回调函数，它们的行为类似于中间件来处理请求。唯一的例外是，这些回调可能调用 next('route')来绕过剩余的路由回调。您可以使用此机制对路由施加前置条件，然后如果没有理由继续当前路由，则将控制权传递给后续路由。

```
app.get('/example/b', function (req, res, next) {
  console.log('the response will be sent by the next function ...')
  next()
}, function (req, res) {
```

```
  res.send('Hello from B!')
})
```

响应头方法 -- Response methods

下表中响应对象(res)上的方法可以向客户机发送响应，并终止请求-响应周期。如果没有从路由处理程序调用这些方法，客户机请求将挂起。

方法	描述
res.download()	提示要下载的文件。
res.end()	结束响应过程。
res.json()	发送一个 JSON 响应
res.jsonp()	发送一个支持 JSONP 的 JSON 响应。
res.redirect()	重定向请求
res.render()	呈现视图模板
res.send()	发送各种类型的响应
res.sendFile()	以八隅体流的形式发送文件
res.sendStatus()	设置响应状态代码，并将其字符串表示形式作为响应体发送

app.route()

您可以使用 app.route()为路由路径创建可链接的路由处理程序。由于路径是在单个位置指定的，因此创建模块化路由很有帮助，减少冗余和拼写错误也是如此。

```
app.route('/book')
  .get(function (req, res) {
    res.send('Get a random book')
  })
  .post(function (req, res) {
```

```
    res.send('Add a book')
  })
  .put(function (req, res) {
    res.send('Update the book')
  })
```

可挂载路由处理程序 -- express.Router

使用 express 来创建模块化的可挂载路由处理程序。路由器实例是一个完整的中间件和路由系统;因此，它经常被称为“迷你应用程序”。
下面的示例创建一个路由器作为模块，在其中加载一个中间件函数，定义一些路由，并将路由器模块挂载在主应用程序中的路径上。
在 app 目录中创建一个名为 birds.js 的路由器文件，内容如下:

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

// 路由器的中间件
router.use(function timeLog (req, res, next) {
  console.log('Time: ', Date.now())
  next()
})
// 首页路由
router.get('/home', function (req, res) {
  res.send('Birds home page')
})
// 关于我们
router.get('/about', function (req, res) {
  res.send('About birds')
})
module.exports = router;
```

然后在 app 中加载路由器模块

```
var birds = require('./birds')
// ...
app.use('/birds', birds)
```

该应用程序现在将能够处理/birds 和/birds/about 的请求，以及调用特定于路由的 timeLog 中间件功能。

利用 Express 托管静态文件

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。

你可以使用 express.static 中间件来设置静态文件路径。例如，如果你将图片， CSS, JavaScript 文件放在 public 目录下，你可以这么写：

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

我们可以到 public/images 目录下放些图片,如下所示：

```
public/images
public/images/logo.png
```

此函数特征如下：

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

例如，通过如下代码就可以将 public 目录下的图片、CSS 文件、JavaScript 文件对外开放访问了：

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

现在，你就可以访问 public 目录中的所有文件了：

```
http://localhost:3000/images/kitten.jpg
http://localhost:3000/css/style.css
```

Express 在静态目录查找文件，因此，存放静态文件的目录名不会出现在 URL 中。
如果要使用多个静态资源目录，请多次调用 express.static 中间件函数：

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

为 express 提供的文件创建虚拟路径前缀(其中路径实际上并不存在于文件系统中)。静态函数，为静态目录指定一个挂载路径，如下所示:

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

现在，你就可以通过带有 /static 前缀地址来访问 public 目录中的文件了。

```
http://localhost:3000/static/images/kitten.jpg
http://localhost:3000/static/css/style.css
```

如果您从另一个目录运行 express 应用程序，使用您想要服务的目录的绝对路径更安全

```
app.use('/static', express.static(path.join(__dirname, 'public')))
```

热加载

node 可以实时的更新,每次接完接口或者配置无需手动重启，方便快速开发
npm install hotnode --save

NPM 全局包安装位置

Npm install 模块 -g
Yarn global add 模块

当使用 npm 安装一些全局的软件包时，不知道安装到了什么位置，可以使用命令 npm root -g 进行查询，通常默认会保存在以下位置：

Windows

```
C:\Users\username\AppData\Roaming\npm\node_modules
```

如果想修改这个全局安装包的路径可以使用命令设置：

修改全局包安装路径

npm config set prefix D:\software\NODEJS\npm

注意这里没有引号，网上很多有引号是有问题的

如何删除 npm 之前设置的 npm config set prefix...

在安装 npm 时，可能根据某个教程设置了例如：

npm config set prefix D:\software\NODEJS\npm

之类的东西，可是后来不想要了，想要恢复默认值，怎么办呢？

方法是删除 C:\Users\Administrator\.npmrc 这个文件。如果.npmrc 不在这个目录下，就全局搜一下啦。

Express 应用程序生成器

通过应用生成器工具 express-generator 可以快速创建一个应用的骨架。
express-generator 包含了 express 命令行工具。通过如下命令即可安装：

```
C:\Users\RM>yarn global add express express-generator
yarn global v1.19.1
[1/4] Resolving packages...
[2/4] Fetching packages...
info fsevents@1.2.9: The platform "win32" is incompatible with thi
info "fsevents@1.2.9" is an optional dependency and failed compati
[3/4] Linking dependencies...
[4/4] Building fresh packages...
warning "express@4.17.1" has no binaries
success Installed "express-generator@4.16.1" with binaries:
      - express
Done in 42.65s.
```

安装: npm install express express-generator -g 或者

yarn global add express express-generator

注意：express 4.0 以上的 版本需要同时全局安装 express 和 express-generator，需要这 2 个安装后还是提示以下错误，就需要配置环境变量

win10 安装 express 后仍报错：'express' 不是内部或外部命令，也不是可运行的程序 或批处理文件
修改完，**一定记得**重启控制台服务

环境变量的配置参考:https://blog.csdn.net/m0_37750720/article/details/82937463

-h 参数可以列出所有可用的命令行参数：
express -h

```
  Usage: express [options] [dir]
  Options:
    -h, --help          输出使用方法
        --version       输出版本号
    -e, --ejs           添加对 ejs 模板引擎的支持
```

```
    --hbs              添加对 handlebars 模板引擎的支持
    --pug              添加对 pug 模板引擎的支持
-H, --hogan            添加对 hogan.js 模板引擎的支持
    --no-view          创建不带视图引擎的项目
-v, --view <engine> 添加对视图引擎（view） <engine> 的支持 (ejs|hbs|hjs|jade|pug|twig|vash) （默认是 jade 模板引擎）
-c, --css <engine>  添加样式表引擎 <engine> 的支持 (less|stylus|compass|sass) （默认是普通的 css 文件）
    --git              添加 .gitignore
-f, --force            强制在非空目录下创建
```

```
express --view=ejs myapp
```

命令创建了一个名称为 myapp 的 Express 应用。此应用将在当前目录下的 myapp 目录中创建，并且设置为使用 Pug 模板引擎（view engine）
然后安装所有依赖包：

```
$ cd myapp
$ npm install
```

在 MacOS 或 Linux 中，通过如下命令启动此应用：

```
$ DEBUG=myapp:* npm start
```

在 Windows 中，通过如下命令启动此应用：

```
> set DEBUG=myapp:* & npm start
```

然后在浏览器中打开 http://localhost:3000/ 网址就可以看到这个应用了。
通过生成器创建的应用一般都有如下目录结构：

```
├── app.js      //入口文件(主文件) 总路由 (其他的路由 要由它来分配)
├── bin      //启动目录 里面包含了一个启动文件 www 默认监听端口是 3000 (不用)
|   └── www
├── package.json    //包描述文件  最重要的是 依赖的模板列表 dependencies
                    //依赖列表里面的所有模板 可以通过 cnpm i 一次性全部安装
├── public      //所有的前端静态资源  html css image  js
|   ├── images
|   ├── javascripts
|   └── stylesheets
|       └── style.css
├── routes
        放的是 路由 文件 (默认有两个)
        路由主要定义 url 和 资源 的映射关系 ( 一一对应关系 )
        主要用来接收前端发送的请求 响应数据给前端
```

```
|   ├── index.js
|   └── users.js
└── views    //主要放置 ejs 后端模板文件
    ├── error.pug
    ├── index.pug
    └── layout.pug
node_modules:  //所有安装的依赖模块 都在这个文件夹里面
```

通过 Express 应用生成器创建应用只是众多方法中的一种。你可以不使用它，也可以修改它让它符合你的需求。

跨域

只有 Web 才有跨域 CORS，移动端 iOS 与 Android 就没有，谁让 Web 能看源代码呢，沙盒机制也不如移动端健全。

- 同源策略的限制：XmlHttpRequest 只允许请求当前源（域名、协议、端口）的资源，所以 AJAX 是不允许跨域的。
- 相反就是跨域：如果想让 XmlHttpRequest 按照自己意愿（域名、协议、端口）请求数据，那就需要跨域

那为什么有同源策略限制?

没有同源策略的话，资源（如 HTTP 头、Cookie、DOM、localStorage 等）就能相互随意访问，那就没有安全了。同源策略就是把每个网站都关在一个笼子里，每个网站互相访问不到数据，只有用户和网站开发者可以访问数据，这样就安全了。

下面是 Node 如何跨域的两种方法，均是在服务器设置，并不是 JSOP 这样不优雅的方式。

方法一 Access-Control-Allow-Origin：

```
app.use('*',function (req, res, next) {
  res.header('Access-Control-Allow-Origin', '*');
//这个表示任意域名都可以访问，这样写不能携带 cookie 了。
//res.header('Access-Control-Allow-Origin', 'http://www.baidu.com');
//这样写，只有 www.baidu.com 可以访问。
  res.header('Access-Control-Allow-Headers', 'Content-Type, Content-Length, Authorization,
```

```
Accept, X-Requested-With , yourHeaderFeild');
  res.header('Access-Control-Allow-Methods', 'PUT, POST, GET, DELETE, OPTIONS');//设置方法
  if (req.method == 'OPTIONS') {
    res.send(200); // 意思是，在正常的请求之前，会发送一个验证，是否可以请求。
  }
  else {
    next();
  }
});
```

方法二 cors：

```
npm install cors
var cors = require('cors');

app.use(cors());
```

Express 中间件

http://www.expressjs.com.cn/resources/middleware.html

使用 Express 的模板引擎

模板引擎允许您在应用程序中使用静态模板文件。在运行时，模板引擎用实际值替换模板文件中的变量，并将模板转换为发送给客户机的 HTML 文件。这种方法使设计 HTML 页面变得更容易。

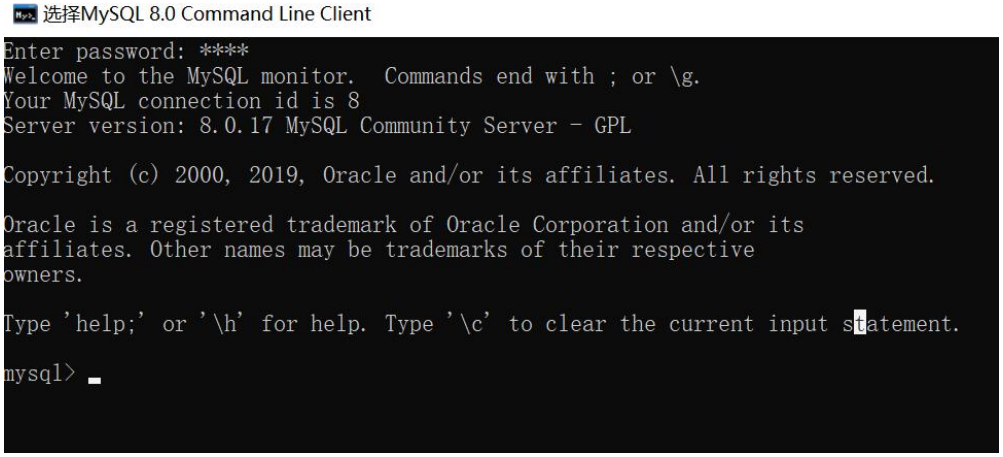
使用 Express 的一些流行模板引擎是 Pug、Mustache 和 EJS。Express 应用程序生成器使用 Jade 作为默认值，但它还支持其他一些功能。

```
npm install ejs--save
app.set('views', '.views');  //指定视图目录
app.set('view engine', ejs) // 注册模板引擎
```

MYSQL 数据库命令

1、数据库安装 设置密码 root 端口默认 3306 根用户 root 管理员 root

2、操作数据库



3、查看数据库

```
show databases;
```

4、创建数据库

```
create database jd;
```

5、删除数据库

```
drop database jd;
```

6、使用数据库

```
use jd;
source D:\vip\vue-jd\jd.sql
```

7、查看当前使用的数据库

```
mysql> select database();
```

8、启 node 服务 在 DOC

```
node server.js
```

表操作

操作之前应连接某个数据库

1、建表

命令：create table <表名> (<字段名> <类型> [,..<字段名 n> <类型 n>]);

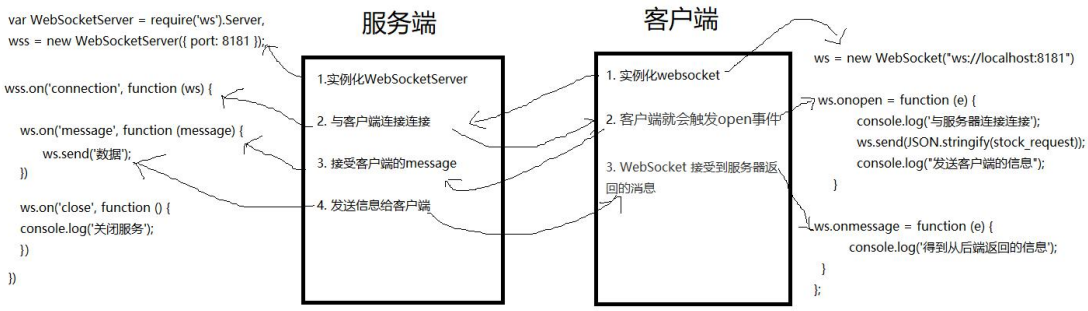
```
mysql> create table MyClass(
> id int(4) not null primary key auto_increment,
> name char(20) not null,
> score char(50) not null )
> ;
```

2、获取表结构

命令：desc 表名，或者 show columns from 表名
mysql>DESCRIBE MyClass
mysql> desc MyClass;
mysql> show columns from MyClass;
3、删除表
命令：drop table <表名>
例如：删除表名为 MyClass 的表
mysql> drop table MyClass;

4、插入数据
命令：insert into <表名> [(<字段名>[,..<字段名 n >])] values (值)[, (值 n)]
例如，往表 MyClass 中插入二条记录，这二条记录表示：编号为的名为 Tom 的成绩为.45，编号为 的名为 Joan 的成绩为.99，编号为 的名为 Wang 的成绩为.5.
mysql> insert into MyClass values(1,’Tom’,96.45),(2,’Joan’,82.99), (2,’Wang’, 96.59);
5、查询表中的数据
1)、查询所有行
命令：select <字段，字段，...> from < 表名 > where < 表达式 >
例如：查看表 MyClass 中所有数据
mysql> select * from MyClass;
2）、查询前几行数据
例如：查看表 MyClass 中前行数据
mysql> select * from MyClass order by id limit 0,2;
或者：
mysql> select * from MyClass limit 0,2;
6、删除表中数据
命令：delete from 表名 where 表达式
例如：删除表 MyClass 中编号为 的记录
mysql> delete from MyClass where id=1;
7、修改表中数据：update 表名 set 字段=新值,…where 条件
mysql> update MyClass set name=’Mary’ where id=1;
7、在表中增加字段：
命令：alter table 表名 add 字段 类型 其他;
例如：在表 MyClass 中添加了一个字段 passtest，类型为 int(4)，默认值为
mysql> alter table MyClass add passtest char(50) default ‘’;
8、更改表名：
命令：rename table 原表名 to 新表名;
例如：在表 MyClass 名字更改为 YouClass
mysql> rename table MyClass to YouClass;
更新字段内容
update 表名 set 字段名 = 新内容
update 表名 set 字段名 = replace(字段名,’旧内容’,'新内容’)

图例演示



EJS 模板在 express 中的使用攻略

一、什么是 ejs?

ejs 当中的"E" 代表 "effective"，即【高效】。EJS 是一套非常简单的模板语言，可以帮你利用普通的 JavaScript 代码快速生成 HTML 页面。EJS 没有如何组织内容的教条；也没有再造一套迭代和控制流语法；有的只是普通的 JavaScript 代码而已。

二、快速使用 EJS

1、安装 ejs 与 express

```
cnpm install ejs express -D
```

2、在项目中新建 demo.js：

```
const express = require("express");
const ejs = require("ejs");
const app = express();
app.get("/", (req, res) => {
    // 创建用于渲染的数据
    var data = {
        siteName: "张培跃",
        siteUrl: "http://www.zhangpeiyue.com"
    }
    // 创建模板内容
    var template =
        "<a href='<%=siteUrl%>'>" +
        "<%=siteName%>" +
        "</a>";
    // 通过 ejs.render 将数据放到模板中，转为 HTML 数据
    let html = ejs.render(template, data);
    // 将数据在浏览器进行展现
    res.send(html)
    ;})
    app.listen(80, function () {
    console.log("开启服务成功！")
    }
    )
```

代码解析：

ejs.render()方法：用于将数据（data）在指定的模板（template）中进行展示，生成 HTML

<%= 数据的属性 %>：用于将数据的属性在模板中进行输出

注意：数据的类型需要是对象

3、效果：

三、以文件形式使用模板

在上个例子中，我们将模板放到变量 template 中，数据量少的话还可以，倘若数据量比较大的话，将是一件十分恐怖的事情。所以我们可以将模板放到文件中，现在对以上示例进行改造。

1、创建 views 文件夹

2、在 views 文件夹内创建 one.ejs 模板文件：

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Title</title>
</head><body>
    <a href="<%= siteUrl %>">
        <%= siteName %>
</a>
</body>
</html>
```

3、demo.js 调整如下：

```
const express = require("express");
const ejs = require("ejs");
const app = express();
app.get("/", (req, res) => {
    // 创建用于渲染的数据
    var data = {
        siteName: "张培跃",
        siteUrl: "http://www.zhangpeiyue.com"
    }
    // 将数据在浏览器进行展现
    res.render("one.ejs",data);
    })
app.listen(80, function () {
    console.log("开启服务成功！")})
```