任务：GameManager

用途：成为所有游戏的模板

前端语言：react-JS

后端语言：node-js

后端模板：express

前后端路由：react-router

JS标准：ES6

学习内容：react（16）, ES6，react-router（4.2.0），node-js（8.9.1），express（4.12.3），

Webpack（提供前端打包服务，显著减少包大小），react-script

任务1（18.2.8）express架构下的**静态资源管理器**

功能：上传文件+显示

疑问：有了express，那react-router在哪里运用？前后端的交互？前后端的分离？[中间件到底是个什么概念？](#中间件是什么)express和express.router有什么区别？

前后端交互：通过create-react-app explorer\_client生成前端代码框架，yarn start启动

通过express [--view=pug] explorer\_server生成后端代码框架，npm start启动

注：前后端使用不同的端口为了前后端的彻底分离，前后端能够分别进行测试。

前端框架结构分析： 1. [package.json](#_Package.json代码示例：)为配置文件。它定义了这个项目所需要的各种模块，以及配置信息。Scripts部分为运行脚本命令的npm命令行缩写，比如yarn build可以将依赖库重新initiate。Dependencies(项目运行依赖模块)/DevDependencies（项目开发依赖模块）分指定版本1.1.2，~1.2.2=1.2.x，^1.2.2=1.x.x。bin字段指定内部指令对应的可执行文件位置。

2. app.js表示react组件

3. index.js用来渲染reactDOM,并包括[BrowserRouter](#browserRouter)(注意react-router 4.x.x版本已无hashHistory,单用Router会莫名报错)。

4. index.html为默认html页面。

细节： 1. 支持不同文件格式（.jpg,.doc,.txt）的上传

2. 包括一定方式的浏览（图片浏览+文字浏览）

3. 支持刷新（上传后自动刷新）

4. 资源浏览器界面（界面设计）

代码描述：BrowserRouter是前端路由，根据不同的URL执行不同的函数。

步骤： 先实现图片功能

1. 先写前端框架结构（上传图片部分）√

2. 前后端路由搭建（数据传输通路）

3. 后端获取图片并存储

4. 后端获取所有图片并返回给前端

5. 前端展示所有图片

然后实现剩余展示功能，可参考<http://www.cnblogs.com/SheilaSun/p/7271883.html>

<http://www.dengzhr.com/node-js/1199>

文件夹->判断文件后缀->展示->后退

2.26 到手GM代码，可以仔细研究一下了；顺便干着自己的OCT和答辩事宜，这天可是太忙了

2.28.2018 继续看代码，从头开始看。先看后端再看前端。走个game Server查询玩家的流程，request的功能可是太强了，还能够理解。明天继续问GM之间的状态通信问题。

gmServer

|- bin

|-gmServer: 启动项，定义监听端口并处理异常

|- config

|- config.js: exports.XX来设置各种通用配置（这样做可以很方便直观地修改重要default值，保证不同机器的通用的便捷性）

|- logic

|- devOpsLogic.js: 运维逻辑，处理app.js中的devOps逻辑

|- logic.js: 前后端的内部逻辑，涉及很多数据库的操作

|- logicCommon.js: 通用逻辑（加密算法，格式化对象，JSON，时间，获取IP对应的地域信息，超时重发，深度拷贝，GET,POST,FORM-DATA,DELETE,PUT）

|- routes

|- index.js: 设置跨域，处理post请求，在req头部存入key:value值，用来区分不同的请求映射，赋值给后端。和req.params.action的区别在哪？为什么要用正则？

|- app.js:中间件的使用，定义路由接口（outer桥接工具接口，server沟通游戏服务器，dev运维平台数据请求，index处理GM前后端的内部逻辑以及GET静态资源）

gmTools

|- build:

|- static: webpack压缩src部分的代码，包括css,js,media等

|- src:

|- actions

|- components

|- config

|- containers

|- routes

|- reducers

|- static

|- store

|- configureStore.js: 模块热替换？运行时更新各种模块，而无需进行完全刷新

|- utils

App.js

3.1 新的一个月！主要问昨天状态通信的问题，原来是和数据库交互的，接口比较复杂，在cComp内有定义，BUG的解决比较复杂。放弃静态文件管理器，接下来是业务逻辑了。

3.2 ★task1（3 hrs）：勾选逻辑中的打勾就可以确认的状态逻辑。（养成了记日记的习惯也不错，实习答辩应该算是结束了）。一开始以为只是写一个小功能，没想到先要改一个BUG,真正的提交不是“保存”（对应的SYSTEM\_CONFIG\_USER\_EDIT\_SUBMIT），而是SYSTEM\_CONFIG\_PERMISSION\_SUBMIT,修改某一模块已经保存了。哦，不，不是这个问题。所以之前的保存到底是什么意思？ 好的，状态保存，没有写入数据库。

接着做这个功能，记得每次查询需要刷新之后再查看。也是醉了。。。

实现细节：在reducerSystemConfig下，对于action-type为SYSTEM\_CONFIG\_PERMISSION\_MENU\_CHECKED，不仅要修改state.PermissionList. currentPermissionIDCheckListBack， 此外还要修改 state. PermissionList.currentPermissionIDCheckList;一共有三级保存系统：

actionSystemConfigUserEditSubmit

action完成后提交

★task2：删除的时候，顺便把用户删除。

★★★task3：邮件内容加入编辑功能 测试账号ID：108086459300520040

3.5 完成邮件内容编辑功能，逻辑比较复杂，因为加入了组件editor (draft.js)。

首先要给editor一个状态，需要在state里加入一个editorState，然后放入info里；

再在event内加入action，调用init和change函数；

最后进行action层面的修改，draftHtmlStr生成的字符串满足mailContent需求。

★★★★ task 4 : 需要完成一整个操作，写一个新的功能界面-黑白名单，不需要访问DB，只需要走game server的接口。

4. 另外要在reducers/index.js中添加reducer关联词条

5. 另外要在config.js中配置URL

6. 还要在serverLogic.js中加入相应操作函数；然后根据开放的接口修改logicCommon(调用游戏服务器的内容)

7. 还要在constantLinkSplitWords.js中加入词条作为链接

第一次体验加班文化，其实也就干点活，晚上也是清闲许多，还可以健身，还算好。

3.6 先修改邮件小BUG，然后继续做黑白名单.

功能有3：

1. 搜索黑白名单：完成，记得修改reducer状态的时候，会触发dispatch，从而文本框内的内容自动填充到输入框内；info中的内容来自global或者container，container可以看作是reducer和action的合体，存储所有状态
2. 自动填充原有黑白名单（和搜索有类似之处，刷新之前先进行search的感觉）：在constructor里加入初始化函数即可
3. 更新黑白名单内容

#### Package.json代码示例：

"scripts": {

"preinstall": "echo here it comes!",

"postinstall": "echo there it goes!",

"start": "node index.js",

"test": "tap test/\*.js"

}

"devDependencies": {

"browserify": "~13.0.0",

"karma-browserify": "~5.0.1"

}

"name" : "foo",

"config" : { "port" : "8080" },

#### 前后端分离

在传统的web应用开发中，大多数的程序员会将浏览器作为前后端的分界线。将浏览器中为用户进行页面展示的部分称之为前端，而将运行在服务器，为前端提供业务逻辑和数据准备的所有代码统称为后端。

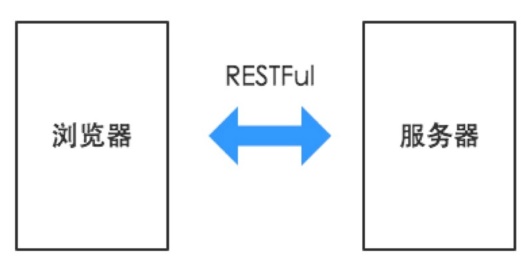
由于前后端分离这个概念相对来说刚出现不久，很多人都是只闻其声，不见其形，所以可能会对它产生一些误解，误以为前后端分离只是一种web应用开发模式，只要在web应用的开发期进行了前后端开发工作的分工就是前后端分离。

其实前后端分离并不只是开发模式，而是web应用的一种架构模式。在开发阶段，前后端工程师约定好数据交互接口，实现并行开发和测试；在运行阶段前后端分离模式需要对web应用进行分离部署，前后端之前使用HTTP或者其他协议进行交互请求。然而作为一种架构模式，我们在实施的过程中主要对以下四个方面来进行比较和重新认识。

前后端分离大概可以从四个方面来理解：

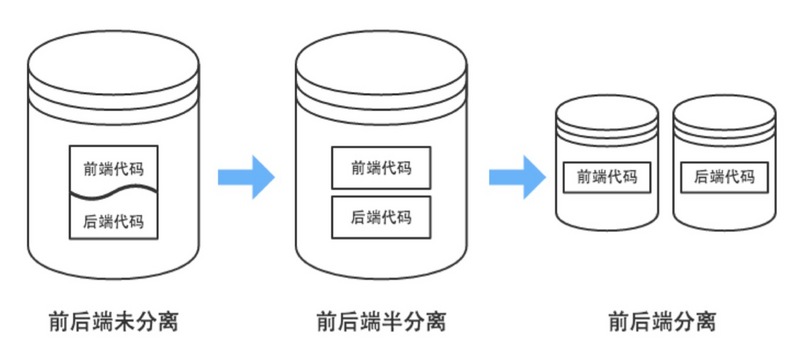
1. 交互形式
2. 代码组织方式
3. 开发模式
4. 数据接口规范流程

### 一、交互形式



* 后端只需要负责按照约定的数据格式向前端提供可调用的API服务即可
* 前后端之间通过HTTP请求进行交互
* 前端获取到数据后，进行页面的组装和渲染，最终返回给浏览器。

### 二、代码组织方式



在传统架构模式中，前后端代码存放于同一个代码库中，甚至是同一工程目录下。页面中还夹杂着后端代码。前后端工程师进行开发时，都必须把整个项目导入到开发工具中。

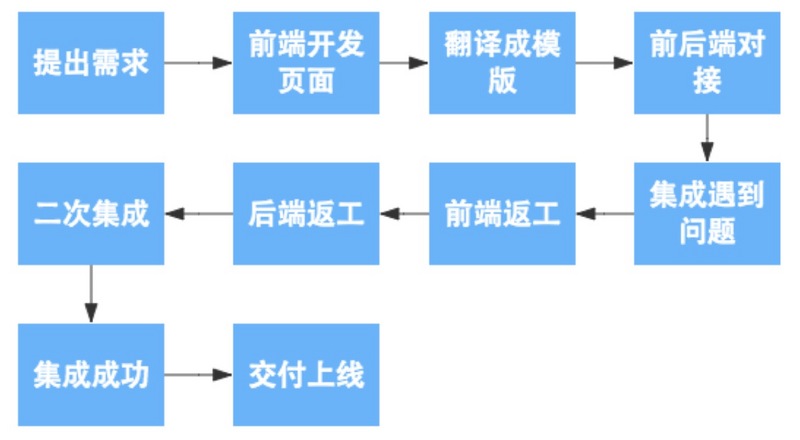
而前后端分离模式在代码组织形式上有以下两种:

* **半分离**  
  前后端共用一个代码库，但是代码分别存放在两个工程中。后端不关心或很少 关心前端元素的输出情况，前端不能独立进行开发和测试，项目中缺乏前后端 交互的测试用例。
* **分离**   
  前后端代码库分离
  + 前端代码中有可以进行Mock测试(通过构造虚拟测试对 象以简化测试环境的方法)的伪后端，能支持前端的独立开发和测试。
  + 后端代码中除了功能实现外，还有着详细的测试用例，以保证API的可用性，降低集成风险。

### 三、开发模式

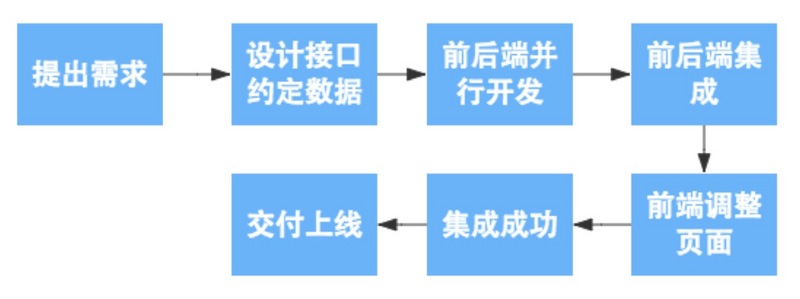
我们之前的架构属于传统的MVC架构，整体没有进行前后端分离，在项目的开发阶段，前端工程师负责编写HTML，完成前端的页面设计并套页面，然后再使用模板技术将写好的前端代码转换为Smarty脚本，同时内嵌一些后端提供的模板变量和一些逻辑操作。应用运行期，将全部代码进行打包，和后端代码部署到同一服务器上，同时会进行简单的动静态分离部署。

此时，应用的开发流程如下图所示。



而在实现前后端分离架构之后，前端工程师只需要编写HTML、js、CSS等前端资源，然后通 过HTTP请求调用后端提供的服务即可。除了开发期的分离，在运行期前后端资源也 会进行分离部署。

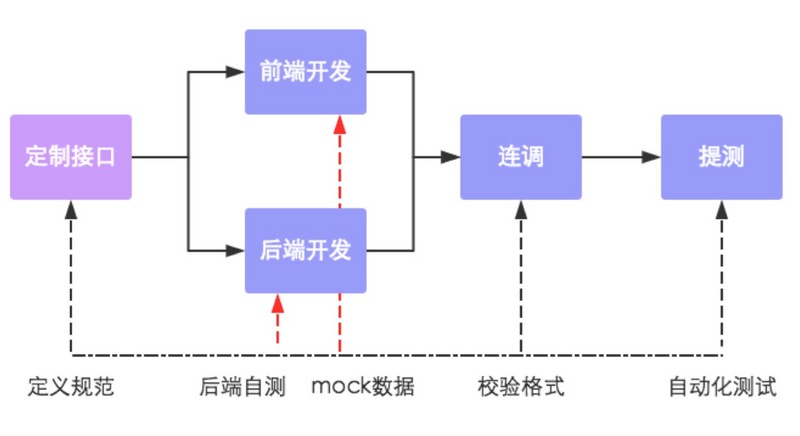
前后端分离之后，开发流程将如下图所示。



通过上面的两幅流程图，不难发现，在开发模式上，前后段分离不仅仅只是工程师的分工开发，更重要的意义在于实现了前后端的并行开发，简化了开发流程。

### 四、数据接口规范流程

在开发期间前后端共同商定好数据接口的交互形式和数据格式。然后实现前后端的并行开发，其中前端工程师再开发完成之后可以独自进行mock测试，而后端也可以使用接口测试平台进行接口自测，然后前后端一起进行功能联调并校验格式，最终进行自动化测试。



## 分离的四个好处

前后端分离模式和传统的web应用架构相比有很大的不同，到底分还是不分，这还真是个问题。

从目前应用软件开发的发展趋势来看，主要有两方面需要注意：

1. 越来越注重用户体验，随着互联网的发展，开始多终端化。
2. 大型应用架构模式正在向云化、微服务化发展。

我们主要通过前后端分离架构，为我们带来以下四个方面的提升：

* **为优质产品打造精益团队**  
  通过将开发团队前后端分离化，让前后端工程师只需要专注于前端或后端的开发工作，使得前后端工程师实现自治，培养其独特的技术特性，然后构建出一个全栈式的精益开发团队。
* **提升开发效率**  
  前后端分离以后，可以实现前后端代码的解耦，只要前后端沟通约定好应用所需接口以及接口参数，便可以开始并行开发，无需等待对方的开发工作结束。与此同时，即使需求发生变更，只要接口与数据格式不变，后端开发人员就不需要修改代码，只要前端进行变动即可。如此一来整个应用的开发效率必然会有质的提升。
* **完美应对复杂多变的前端需求**  
  如果开发团队能完成前后端分离的转型，打造优秀的前后端团队，开发独立化，让开发人员做到专注专精，开发能力必然会有所提升，能够完美应对各种复杂多变的前端需求。
* **增强代码可维护性**  
  前后端分离后，应用的代码不再是前后端混合，只有在运行期才会有调用依赖关系。

应用代码将会变得整洁清晰，不论是代码阅读还是代码维护都会比以前轻松。

## 需要进行前后端分离的场景

虽然前后端分离架构能带来许多的好处，但前提是建立在开发团队合适的基础上的。

1. 页面布局复杂，使用了主题和样式。
2. 需要有较高的页面渲染效果
3. 前端页面中包含复杂业务逻辑
4. 页面需要渲染的数据量较大

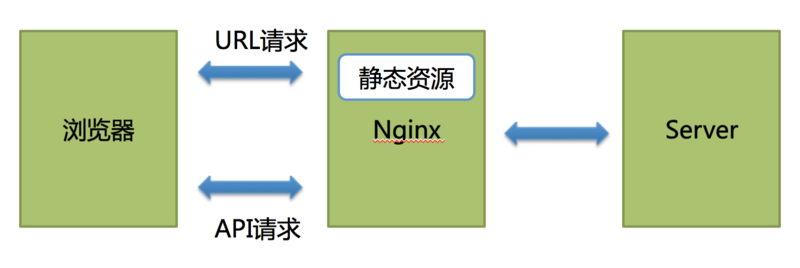
像这种重前端的应用我们综合考虑了各种情况，最终决定采用前后端分离架构。

## 部署方案

前后端分离之后，应用在部署时也需要进行前后端分离。在进行前后端分离方案选择时，需要结合项目的实际情况和用户来考虑。

### 分离之前的架构

前后端分离之前，网盘的后端架构是Nginx服务和后端的PHP服务以及前端的静态资源都是部署在同一台服务器上。当浏览器发起访问请求时，如何请求的是静态资源，Nginx直接把静态资源返回给服务器；如果请求的是页面或后端服务，则经Nginx将请求转发到后端的PHP服务器，完成响应后经Nginx返回到浏览器。

[](https://segmentfault.com/img/bVNja2)

注：此图中的Nginx属于后端机，主要针对前端机Nginx转发过来的请求进行识别弄转发给本机的PHP服务；前端机和后端机各有一个Nginx服务。

这个方案比较简单，易于实现，而且能到达前后端解耦的目的。而且很多公司目前都是基于这种架构或者一定的变形来实现的web应用。

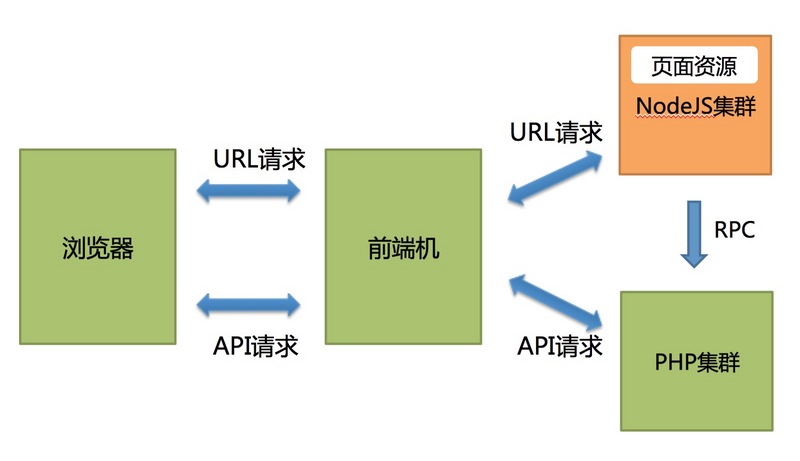
但是对于页面量比较大，需要有良好SEO的应用来说，此方案缺点也较为明显。因为 Nginx只是向浏览器返回页面静态资源，而国内的搜索引擎爬虫只会抓取静态数据， 不会解析页面中的js，这使得应用得不到良好的搜索引擎支持。同时因为Nginx不会进行页面的组装渲染，需要把静态页面返回到浏览器，然后完成渲染工作，这加重了浏览器的渲染负担。

另外，由于这种架构使得前端工程师的工作范围只局限在了浏览器一侧，导致在进行一些特殊的性能优化时，前端工程师无法独立完成，还需要后端开发人员的配合，这也一定程度上影响了双方的进度。

### 分离之后的架构

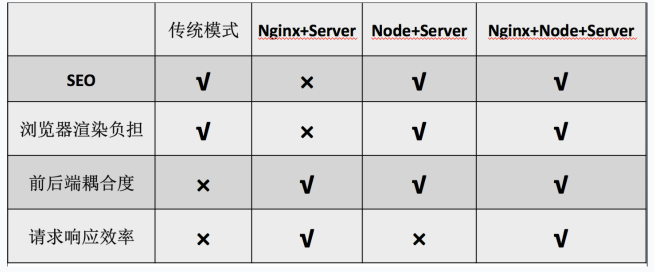
前后端分离之后，我们在原先的架构只上再单独增加了一个Node Server作为中间层，将前端资源部署到Node Server中。Node Server还实现了一层数据代理服务，负责与提供数据的后端服务进行通信。

并且还在这个基础上增加并使用了前端机（前端机是对所有的请求进行预处理和负载均衡，然后再转发给后端机。）的Nginx服务，浏览器发起的请求经过前端机的Nginx进行分发，URL请求统一分发到Node Server，在Node Server中根据请求类型从后端服务器上通过RPC服务请求页面的模板数据，然后进行页面的组装和渲染；API请求则直接转发到后端服务器，完成响应。



注：此图中的Nginx属于前端机。

### 前后端分离方案对比



前端技术，有些已经不会再应用在新系统中，但是还是有很多老系统是使用它们做的。

**语言知识**

• ES5 & ES6 & ES7 //ES语言基础

• HTML5 API & CSS3 //HTML5和CSS特效

· less & sass //CSS预编译语言

• SVG & Canvas & D3.js //图形数据可视化

• WebGL & Three.js //3D场景

• CMD & AMD & CommonJS //语言标准

• RequireJS & SeaJS //ES模块化库

• CoffeeScript & TypeScript //ES语言风格库

• NodeJS & Express & Koa //Node的WEB服务器

• TCP & HTTP & WebSocket //网络协议

• ......

**框架、库**

• jQuery

• Backbone

• Ember

• Angular & Angular2 & Angular4

• React

• Vue & Vue2

• Ionic & Ionic2

• React Native

• Weex

• Electron

• ...

**工具**

• Sublime Text & Atom & Webstorm & VS code //编辑器、IDE

• SVN & Git //代码管理、版本控制

• Chrome Dev Tools & FireFox Developer Edition //浏览器开发者工具

• ESLint & JSLint //JavaScript代码语法检查

• React DevTools //react调试工具

· Redux DevTools //redux调试工具

· Vue DevTools //vue调试工具

• Grunt & Gulp & browserify & Webpack //代码打包及热部署工具

• Babel //ES6、react等语法转换工具，将代码转换成ES5

• forever \* pm2 //nodejs项目部署工具

• karma & mocha & PhantomJS //自动化测试框架

• ......

作者：zollero  
链接：https://www.jianshu.com/p/b6a5a2d155ff  
來源：简书  
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

# Node Express获取参数的几种方式

Node配合Express框架获取参数主要有以下4种形式:

* req.body
* req.params
* req.param()
* req.query

## req.body

req.body在[官方文档](http://expressjs.com/4x/api.html#request)是没有提供的，但是它也是很好用的获取表单参数的方法。req.body是**配合Express中间件bodyParser()中间件**提供的。当bodyParser()中间件使用后，这个对象默认为 {},req.body可以获取到post到body中的内容。  
举个栗子：

|  |  |
| --- | --- |
| 1  2  3 | app.post('/user/signup',function(req,res){  var \_user = req.body.user;  }) |

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | // POST user[name]=tobi&user[email]=tobi@learnboost.com  req.body.user.name  // => "tobi"  req.body.user.email  // => [tobi@learnboost.com](mailto:tobi@learnboost.com)  // POST { "name": "tobi" }  req.body.name  //=> "tobi" |  |

## req.param(name)

req.param()是req.query、req.body、以及req.params获取参数的三种方式的封装。req.param(name)返回name参数的值。  
举个栗子：  
和上面的req.body方法类似

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | // POST name=tobi  app.post('/user?name=tobi',function(req,res){  req.param('name');  // => "tobi"  })  // ?name=tobi  req.param('name')  // => "tobi"  // /user/tobi for /user/:name  req.param('name')  // => "tobi" |

**Web开发中前端，后端和中间件之间的区别**

这是一个细节：

前端层 - >用户界面层通常由HTML，Javascript，CSS，Flash以及ASP.Net，经典ASP，PHP等各种服务器端代码组成。将其视为最接近用户在代码方面。

中间件，中间层 - >一层，通常被称为系统的“管道”部分。 Java和C＃是编写此部分的通用语言，可以被视为UI和数据之间的粘合，可能是Web服务或WCF组件或其他SOA组件。

后端层 - >数据库和其他数据存储通常处于此级别。 Oracle，MS-SQL，MySQL，SAP以及各种现成的软件，都是为数据最后处理的软件而设计的。

任何这些之间可能存在重叠，因为您可以将所有内容都像ASP.Net网站一样倾斜，ASP.Net网站使用内置的AJAX功能生成Javascript，而后面的代码可能包含数据库命令，使代码背后包含中间和后面-end层。 或者，可以使用VBScript作为使用ADO对象的所有层并将所有三个层合并成一个。