辰华网络

当代前端技术及其在开发平台的应用

马超

2018/12/13

目录

概览

前后端分离与单页面应用

ES6 带来的改变

JS 模块化技术演进与 Webpack 构建平台

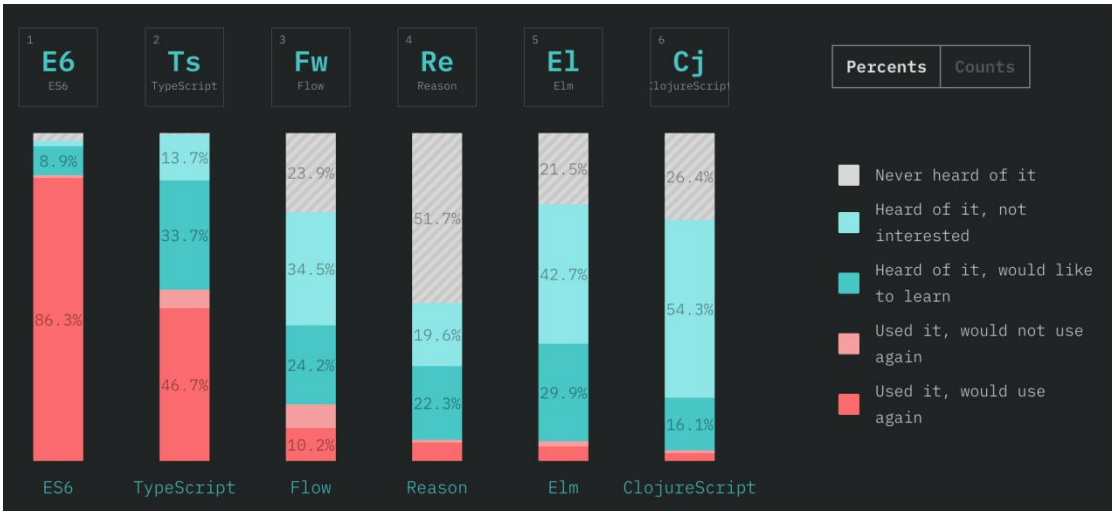
前端组件化与 React（router,axios）

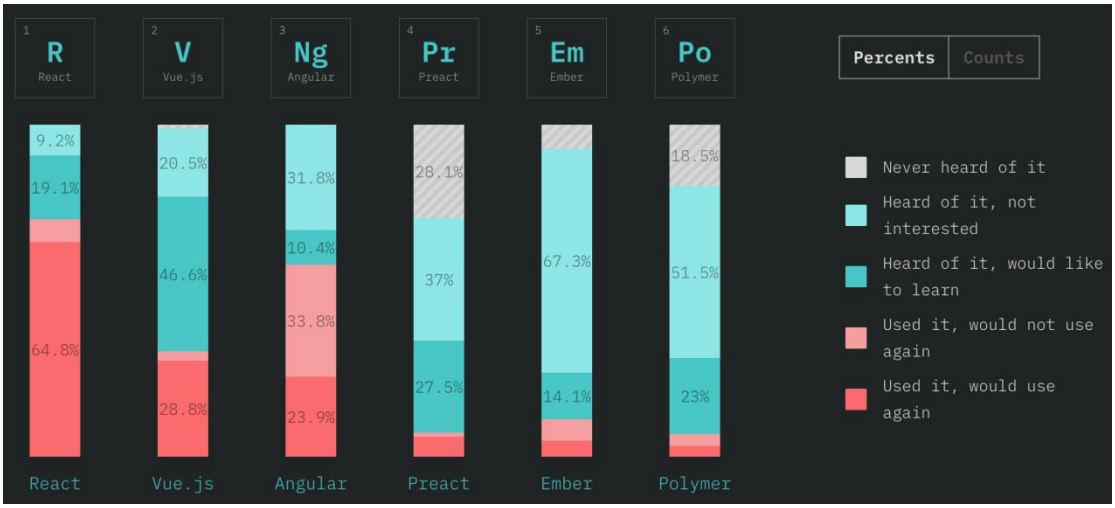
Ant Design 用法用例

脚手架与 Mock Server 使用方法

ES6，React,模块化，webpack 等技术新开发平台前端应用

调试与测试方法





前后端分离

疑问

什么是前后端分离

为什么前后端分离

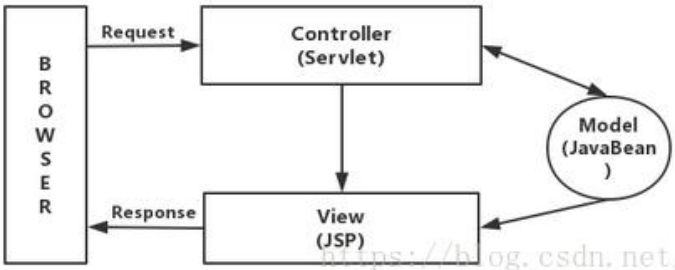
怎么做前后端分离

单页面应用简介

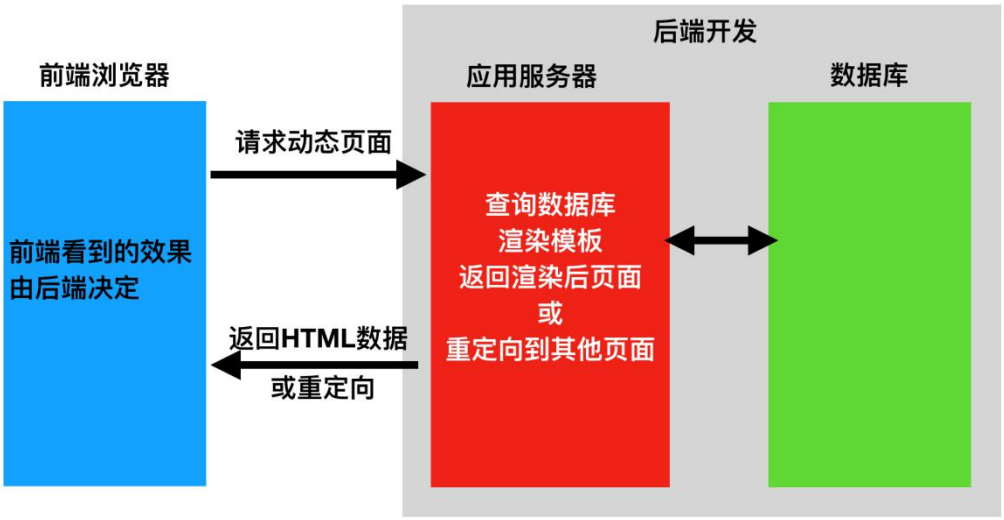
在前后端不分离的应用模式中，前端页面看到的效果都是由后端控制，由后端渲染页面或重定向，也就是后端需要控制前端的展示，前端与后端的耦合度很高。这种应用模式比较适合纯网页应用，但是当后端对接 App 时，App 可能并不需要后端返回一个 HTML 网页，而仅仅是数据本身，所以后端原本返回网页的接口不再适用于前端 App 应用，为了对接 App 后端还需再开发一套接口。
在前后端分离的应用模式中，后端仅返回前端所需的数据，不再渲染 HTML 页面，不再控制前端的效果。至于前端用户看到什么效果，从后端请求的数据如何加载到前端中，都由前端自己决定，网页有网页的处理方式，App 有 App 的处理方式，但无论哪种前端，所需的数据基本相同，后端仅需开发一套逻辑对外提供数据即可。
在前后端分离的应用模式中 ，前端与后端的耦合度相对较低。
在前后端分离的应用模式中，我们通常将后端开发的每个视图都称为一个接口，或者 API，前端通过访问接口来对数据进行增删改查。
前端可以不依赖后端，前端自己起一个服务，如果前端页面处理好，后端的 API 还没有提供，那么，前端可以用 mock 模拟数据。

Web 应用的模式演进：

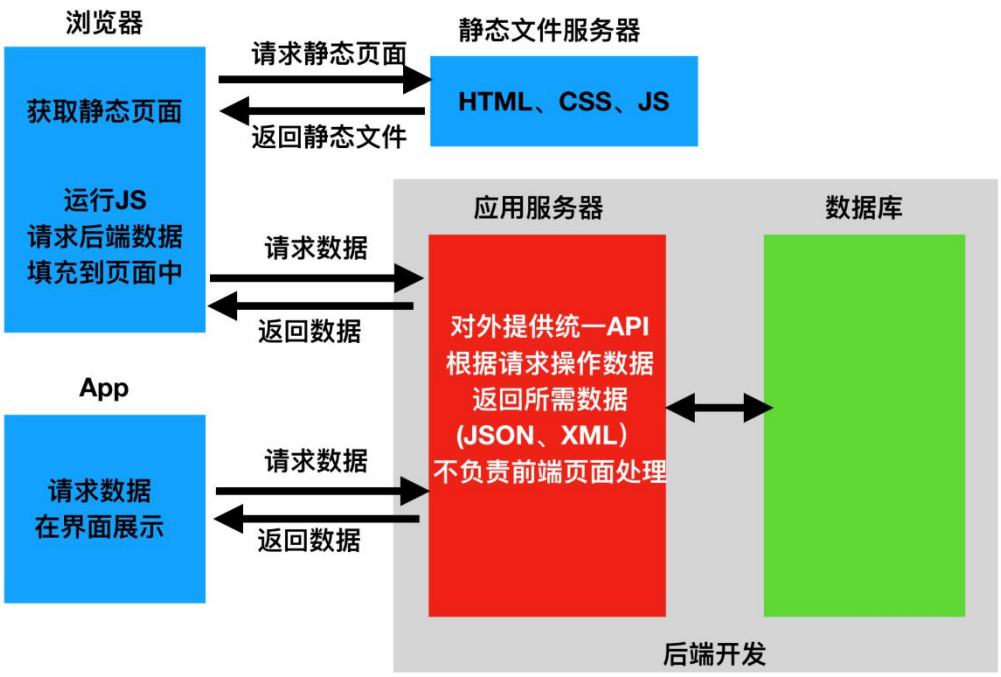
1 传统 MVC



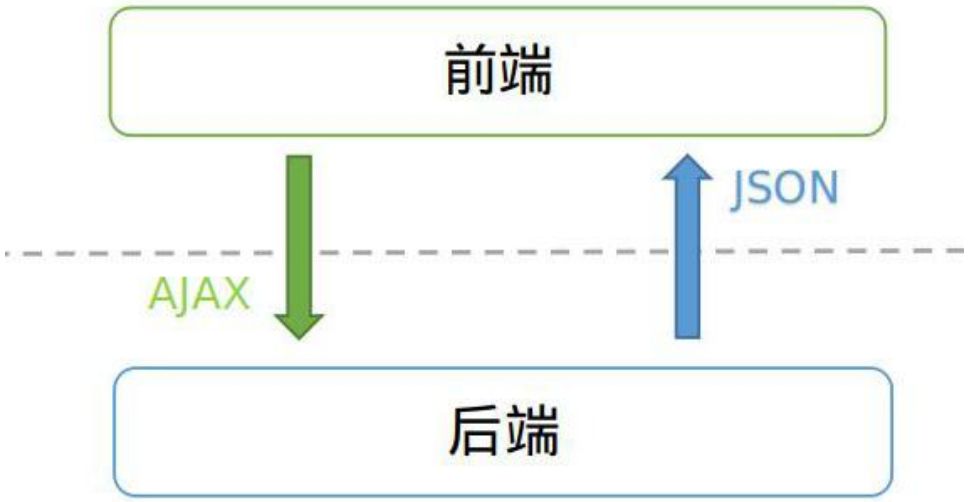
服务端解析模板，组装数据并渲染 html，前端显示出来即可



2 前后端分离



后端提供 API 接口，界面交互，路由控制，数据展示，组件渲染交给前端，前后端通过 json 交流，简洁高效，开发及应用时前后端都能相对独立。



传统模式痛点

1 代码之间耦合性大

数据可能在服务端注入可能在前端 JS 异步获取，页面控件可能在服务端生成也可能是 JS 渲染，前端展示与后台数据处理相杂糅。

2 前后端技术牵制性高，沟通困难

前后端紧密相连，相互纠缠，前后端技术扩展受到限制，小小的改变牵动各处修改，容易带来 bug。

3 系统扩展性差

前后端高度耦合导致系统架构调整困难，后端接口扩展难度大。

4 开发效率低，体验差

界面每次调试要重启整个应用，耗时
前端依赖于后端，后台修改导致页面无法调试
前后端无法并行开发，前端要等后端接口，后端要等前端模板嵌套

5 代码维护成本高

代码职责不分离，可读性差，扩展性低，导致后期面对需求改动时修改的成本较高

6 耗费服务器资源

每次服务端解析模板，组装数据并渲染 html，耗费服务器资源，无法充分发挥客户端浏览器的作用

前后端分离模式的特点

前端静态化

前端有且仅有静态内容，只有 HTML/CSS/JS， 其内容来自于完全静态的资源而不需要任何后台技术进行动态化组装。前端内容的运行环境和引擎完全基于浏览器本身。

后端数据化

后端可以用任何语言,技术和平台实现，但它们必须遵循一个原则：只提供数据,不提供任何和界面表现有关的内容。换言之，他们提供的数据可以用于任何其他客户端(如本地化程序,移动端程序)。

平台无关化

前端 3 大技术本身就是平台无关的,而后台连接部分的本质是实现合适的 RESTful 接口和交互 Json 数据,就这 2 者而言,任何技术和平台都可以实现.

构架分离化

前端架构完全基于 HTML/CSS 的发展和 JS 框架的演变,与我们耳熟能详的后台语言(如 C#, Java, NodeJs 等)完全无关。后端构架几乎可以基于任何语言和平台的任何解决方案，大型构架方面, RESTful Api 可以考虑负载均衡；而数据，业务实现等可以考虑数据库优化和分布式。总而言之，前后端的分离也实现了前后端构架的分离.

前后端分离模式的优势

精简网络请求，页面性能提高

我们知道,前后端流量的大头是 HTML/JS/IMG 资源,而数据仅仅是小头，页面的绝大部分内容都是本地缓存直接加载,远程获取的仅仅是 1-2 个 10K 的内容,其加载时间百毫秒内,这和本地页面几无区别,其前端加载和响应速度得到非常大的提高是显而易见的.

前后端平台无关和技术无关

前后端可以选用适合的技术架构，相互不影响，关注双方之间的接口即可。可以一套后端，多套前端（桌面浏览器，Android，IOS）；一套前端，多套后端（由 Nginx 反向代理）。

安全性方面的集中优化

前端静态化以后,前端页面攻击几无可能,一些注入式攻击在分离模式下被很好的规避；而后端安全问题集中化了，后端只能接受 Json，,处理一个 RESTful 接口的安全考虑安全架设和集中优化变得更明确和便利.

开发的分离和构架的分离

开发时前端可独立开发，利用根据接口约定的 mock 数据和反向代理，无缝切换真实接口。前后端的构架也可以分开考虑和架设，前端构架就能集中考虑性能和优化，而业务,安全和存储等问题就能集中到后端考虑.

页面逻辑和呈现效果:

JS 已经无所不能,依托于目前的各种 JS 函数库和框架，在获取到合理的数据以后，几乎没有做不出来的逻辑和效果。数据校验,页面白屏,路由控制,代码复用等等问题，前端技术已经完全可以解决.

服务器性能和优化:

由于前端内容是完全的静态内容,在初次获取以后的大部分时间内,浏览器使用的就是本地缓存,也就是说,服务器的压力主要来自于承载数据的 RESTful API 调用,压力的大幅降低不言而喻.加上对交互数据的合理设计,可以说对客户端-服务端的交互量控制已经接近极限.

安全性:

由于前端静态内容仅仅只能获取,而后端只能接受 Json,应该说,屏蔽了大量可能发生的注入型问题,而一些其他问题,比如非法对象,数据加密,DDOS 等问题,这些本身就是后端人员无法回避的责任,在任何模式下都必须考虑。

单页面应用简介

多页应用

每一次页面跳转的时候，后台服务器都会给返回一个新的 html 文档，这种类型的网站也就是多页网站，也叫做多页应用。

多页应用缺点：

网络请求数据重复读高，网页渲染重复度高，网页之间切换慢

因为每次跳转都需要发出一个 http 请求，如果网络比较慢，在页面之间来回跳转时，就会发现明显的卡顿。

单页应用

第一次进入页面的时候会请求一个 html 文件，刷新清除一下。切换到其他组件，此时路径也相应变化，但是并没有新的 html 文件请求，页面内容也变化了。

原理

前端展示全权交给 JS，JS 会感知到 url 的变化，通过这一点，可以用 JS 动态的将当前页面的内容清除掉，然后将下一个页面的内容挂载到当前页面上，这个时候的路由不是后端来做了，而是前端来做，判断页面到底是显示哪个组件，清除不需要的，显示需要的组件。这种过程就是单页应用，每次跳转的时候不需要再请求 html 文件了。

为什么页面切换快？

页面每次切换跳转时，并不需要做 html 文件的请求，这样就节约了很多 http 发送时延，我们在切换页面的时候速度很快。

Node.js

简介

Node.js 是一个 Javascript 运行环境(runtime environment)，发布于 2009 年 5 月，由 Ryan Dahl 开发，实质是对 Chrome V8 引擎进行了封装。Node.js 不是一个 JavaScript 框架，不同于 CakePHP、Django、Rails。Node.js 更不是浏览器端的库，不能与 jQuery、ExtJS 相提并论。

Node.js 是一个让 JavaScript 运行在服务端的开发平台！

Node 采用一系列“非阻塞”库来支持事件循环的方式。本质上就是为文件系统、数据库之类的资源提供接口。向文件系统发送一个请求时，无需等待硬盘（寻址并检索文件），硬盘准备好的时候非阻塞接口会通知 Node。该模型以可扩展的方式简化了对慢资源的访问，直观，易懂。尤其是对于熟悉 onmouseover、onclick 等 DOM 事件的用户，更有一种似曾相识的感觉。

虽然让 Javascript 运行于服务器端不是 Node 的独特之处，但却是其一强大功能。不得不承认，浏览器环境限制了我们选择编程语言的自由。任何服务器与日益复杂的浏览器客户端应用程序间共享代码的愿望只能通过 Javascript 来实现。虽然还存在其他一些支持 Javascript 在服务器端 运行的平台，但因为上述特性，Node 发展迅猛，成为事实上的平台。在 Node 启动的很短时间内，社区就已经贡献了大量的扩展库（模块）。其中很多是连接数据库或是其他软件的驱动，但还有很多是凭他们的实力制作出来的非常有用的软件。

Node.js 的优点

服务端的 JS 运行环境，丰富的 API

包管理器 npm，世界上最大的包管理平台，各种开源包应有尽有，npm isntall package_name 即可从 npm 服务器下载第三方包

完善的模块化

擅长 web 应用开发，有许多大型高流量网站都采用 Node.JS 做后台服务器或中间件

易用简洁成熟的 WEB 框架，如何 Express，KOA

单线程

Node.js 可以在不新增额外线程的情况下，依然可以对任务进行并发处理 —— Node.js 是单线程的。它通过事件循环（event loop）来实现并发操作，对此，我们应该要充分利用这一点 —— 尽可能的避免阻塞操作，取而代之，多使用非阻塞操作。

非阻塞 IO

V8 虚拟机，C++，高性能

事件驱动

ES6---新一代 JS

ECMAScript 6.0 于 2015 年 6 月正式发布了 ,成为 JavaScript 语言的下一代标准。ES6 增加的语言特性解决了哪些问题，会给开发者哪些便利呢？

JS 发展历程

1995 JS1.0 处理简单页面交互

ECMA 标准化

ES1

ES2

ES3 （我们公司使用的）

ES5

ES6（当前前端开发 JS 市场占有率 90%以上）

ES6 的优势与解决的问题精讲

Promise

在 promise 之前代码过多的回调或者嵌套，可读性差、耦合度高、扩展性低。通过 Promise 机制，扁平化的代码机构，大大提高了代码可读性；用同步编程的方式来编写异步代码，保存线性的代码逻辑，极大的降低了代码耦合性而提高了程序的可扩展性。同步的方式去写异步代码。

```
fetch('/api/todos')
  .then(res => res.json())
  .then(data => ({ data }))
  .catch(err => ({ err }));
```

let， const的变量声明方式，块级作用域

不存在变量提升，不允许重复声明，使语言使用更加严谨，使得变量的作用域更利于管理。

```
function aa() {
  if(flag) {
      var test = 'hello man'
  } else {
      console.log(test)
  }
}
```

```
function aa() {
  if(flag) {
      let test = 'hello man'
  } else {
      //test 在此处访问不到
      console.log(test)
  }
}
```

箭头函数

箭头函数最直观的三个特点。

- 不需要 function 关键字来创建函数
- 省略 return 关键字

- 继承当前上下文的 `this` 关键字，this 指向明确，不用手动

bind(this)

```
//例如：
    [1,2,3].map(x => x + 1)

//等同于：
    [1,2,3].map((function(x){
        return x + 1
    }).bind(this))
```

模版字符串

使得多行字符串及夹杂变量的字符串写起来更加方便。增加遍历器机制，为各种不同的数据结构提供统一的访问机制。

```
//ES5
var name = 'lux'
console.log('hello' + name)
//es6
const name = 'lux'
console.log(`hello ${name}`) //hello lux
```

解构语法

取值更直观方便。

```
const people = {
    name: 'lux',
    age: 20
}
const name = people.name
const age = people.age
console.log(name + ' --- ' + age)
```

```
//对象
const people = {
    name: 'lux',
    age: 20
}
const { name, age } = people
console.log(`${name} --- ${age}`)
//数组
const color = ['red', 'blue']
const [first, second] = color
console.log(first) //'red'
console.log(second) //'blue'
```

展开运算符

Spread Operator 也是三个点儿...

```
//数组
const number = [1,2,3,4,5]
const [first, ...rest] = number
console.log(rest) //2,3,4,5
//对象
const user = {
    username: 'lux',
    gender: 'female',
    age: 19,
    address: 'peking'
}
const { username, ...rest } = user
console.log(rest) //{"address": "peking", "age": 19, "gender": "female"
```

类的支持，class 关键字来定义类

过去，生成实例对象的传统方法是通过构造函数。不再和传统面向对象语言有大的差异。且支持 extends。

```
//类的定义
class Animal {
        //ES6中新型构造器
    constructor(name) {
        this.name = name;
    }
    //实例方法
    sayName() {
        console.log('My name is '+this.name);
    }
}
//类的继承
class Programmer extends Animal {
    constructor(name) {
        //直接调用父类构造器进行初始化
        super(name);
    }
    program() {
        console.log("I'm coding...");
    }
}
```

JS 模块化技术演进与 webpack 构建平台

为什么会有模块化

1. 随着前后端分离模式下前端处理能力的提高当，一个项目开发的越来越复杂的时候，会遇到一些问题，比如：

- **命名冲突**：当项目由团队进行协作开发的时候，不同开发人员的变量和函数命名可能相同；即使是一个开发，当开发周期比较长的时候，也有可能会忘记之前使用了什么变量，从而导致重复命名，导致命名冲突。
- **文件依赖**：代码重用时，引入 js 文件的数目可能少了，或者引入的顺序不对，比如使用 boostrap 的时候，需要引入 jQuery，并且 jQuery 的文件必须要比 boostrap 的 js 文件先引入。

2.模块化可以避免以上问题，提高开发效率，方便后期维护：

- **提升开发效率**：代码方便重用，别人开发的模块直接拿过来就可以使用，不需要重复开发法类似的功能。

- **方便后期维护**：代码方便重用，别人开发的模块直接拿过来就可以使用，不需要重复开发法类似的功能。

所以总结来说，在生产角度，模块化开发是一种生产方式，这种方式生产效率高，维护成本低。从软件开发角度来说，模块化开发是一种开发模式，写代码的一种方式，开发效率高，方便后期维护。

模块化开发的演变过程

1. 全局函数

```
function add(a , b) {
    return parseFloat(a) + parseFloat(b);
}
function substract(a ,b) {}
function multiply(a ,b) {}
function divide(a ,b) {}
```

- 污染了全局变量，无法保证不与其他模块发生变量名冲突。
- 模块成员之间看不出直接关系。

2. 对象封装-命名空间

```
var calculator = {
  add: function(a, b) {
    return parseFloat(a) + parseFloat(b);
  },
  subtract: function(a, b) {},
  multiply: function(a, b) {},
  divide: function(a, b) {}
};
```

- 暴露了所有的模块成员，内部状态可以被外部改写，不安全。
- 命名空间越来越长。

3. 自调用函数

```
var calculator = (function () {
    // 这里形成一个单独的私有的空间
    // 私有成员的作用:
    //    1、将一个成员私有化
    //    2、抽象公共方法（其他成员中会用到的）

    // 私有的转换逻辑
    function convert(input){
        return parseInt(input);
    }

    function add(a, b) {
        return convert(a) + convert(b);
    }
    function subtract(a, b) {}
    function multiply(a, b) {}
    function divide(a, b) {}
    return {
        add : add,
        subtract : subtract,
        multiply : multiply,
        divide : divide
    }
})();
```

1. 利用此种方式将函数包装成一个独立的作用域，私有空间的变量和函数不会影响到全局作用域。
2. 以返回值的方式得到模块的公共成员，公开公有方法，隐藏私有空间内部的属性、元素，比如注册方法中可能会记录日志。
3. 可以有选择的对外暴露自身成员。

模块化规范的流行：AMD 和 CMD 时代　　（2009）

为什么需要模块化规范

以往，js 代码堆砌在一个或几个雍长的 js 文件中，在开发和维护时非常不便。带来变量冲突，问题点查找困难，一方改动时其他地方牵制不清，维护时阅读性低等等问题。

有了模块化，我们就可以更方便地组织代码，想要什么功能，就开发并加载什么模块。不会再像以往那样使用标签引入的方式，而是采用了一种模块的方式去开发。

模块化的优势：

项目组织结构清晰

js 代码职责分离，模块化输入输出

低耦合性

极大地促进代码复用。

SeaJS

RequireJS

CommonJS

```
// CMD
define(function (require) {
    var a = require('./a'); // <- 运行到此处才开始加载并运行模块a
    var b = require('./b'); // <- 运行到此处才开始加载并运行模块b
    // more code ..
})
```

```
// AMD
define(
    ['./a', './b'], // <- 前置声明，也就是在主体运行前就已经加载并运行了模块a和模块b
    function (a, b) {
        // more code ..
    }
)
```

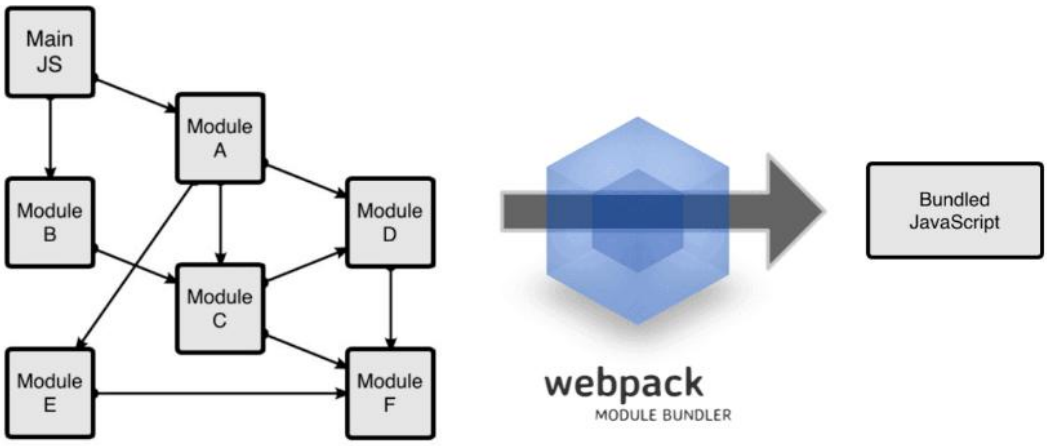
es6 模块化支持（2015）

```
//lib.js 文件
let bar = "stringBar";
let foo = "stringFoo";
let fn0 = function() {
    console.log("fn0");
};
let fn1 = function() {
    console.log("fn1");
};
export{ bar , foo, fn0, fn1}

//main.js文件
import {bar,foo, fn0, fn1} from "./lib";
console.log(bar+"_"+foo);
fn0();
fn1();
```

```
// lib/greeting.js
const helloInLang = {
    en: 'Hello world!',
    es: '¡Hola mundo!',
    ru: 'Привет мир!'
};

export const getHello = (lang) => (
    helloInLang[lang];
);

export const sayHello = (lang) => {
    console.log(getHello(lang));
};
```

```
// hello.js
import { sayHello } from './lib/greeting';

sayHello('ru');
```

webpack



前端代码有分拆就有合并，因为最终一个完整的页面需要所有资源才能运行，所以需要一个打包的过程，webpack 是其中典型的代表。

模块化，让我们可以把复杂的程序细化为小的文件。各种各样的资源都可以认为是一种独特的模块资源，比如 css，js，png，json 等然而细碎的文件怎么让客户端方便加载呢？而我们通过 webpack，可以将这些资源打包压缩在指定的文件中。

Webpack 基于 Node.js 环境运行
Node.js 是操作系统的 js 运行环境，与 webpack 一样，是当代前端开发的必备工具。

1、什么是 Webpack

WebPack 可以看做是**模块打包工具**：它做的事情是，分析你的项目结构，找到 JavaScript 模块以及其它的一些浏览器不能直接运行的拓展语言（Scss，TypeScript 等），并将其打包为合适的格式以供浏览器使用。

2、为什要使用 WebPack

很多网页其实可以看做是功能丰富的应用，它们拥有着复杂的 JavaScript 代码和一大堆依赖包。为了简化开发的复杂度，前端社区涌现出了很多好的实践方法

打包模块;

JS 预处理;

利用社区丰富的包达到想要的功能，如 IE 兼容

scss，less 等 CSS 预处理器

css 模块化

代码压缩

代码混淆

自动把被改动的 js/css 刷新 hash 码，防止客户端缓存了旧代码

启动开发时前端应用服务器

反向代理

源代码映射

自动监测代码变动，重新编译并刷新浏览器

这些改进确实大大的提高了我们的开发效率，但是利用它们开发的文件往往需要进行额外的处理才能让浏览器识别,而手动处理又是非常反锁的，这就为 WebPack 类的工具的出现提供了需求。

使用方法

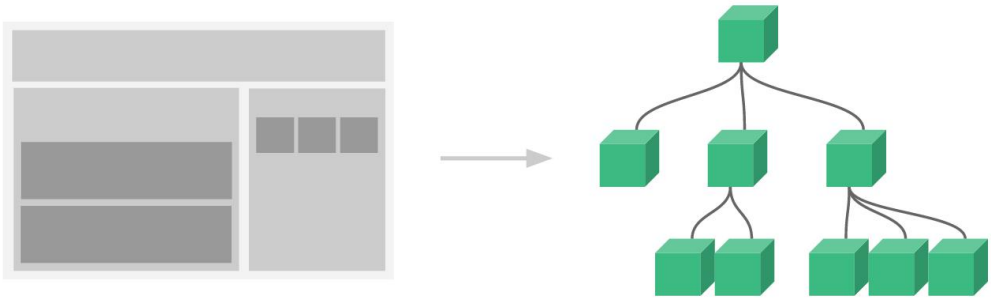
```
//全局安装
npm install -g webpack
//安装到你的项目目录
npm install --save-dev webpack
```

开发模式转变

模块化源代码-》前端独立开发服务器-》打包-》发布

前端组件化与 React

组件化



前端组件化，这个组件化就不仅是 UI 组件，而且包扩具体业务的业务组件。

这种开发的思想其实也就是分而治之，应用分成组件模块分而治之，这样的话开发和需求改动每次只需要改动对应的模块即可，以达到最大程度的降低开发难度与维护成本的效果。

组件化的优势：

① 组件化拆分，使得主控制业务逻辑清晰简单

② 各个业务组件模块功能相对独立，可维护性可测试性大大提升

③ 组件之间可以任意组合，有很强的可重用性

④ 增删模块不会怕影响其他组件运行

⑤ 一个业务模块所需代码全部在一个目录，比较好操作

⑤ 复用组件时候<ComponentName/>即可，不必大段复制 html/js，简洁高效。

ReactJS

React 是由顶尖的 IT 公司 Facebook 和 Instagram 协同开发者社区维护的一个开源 JavaScript 库，是一个组件化思想应用较为极致的前端框架。这个框架现在被广泛地应用于开发 Web 应用程序的用户界面。

严格意义上来说，React 并不是一个框架，只是一个类库，它专注于 MVC 中的 V，但是正是由于这种专注，简单易学，上手快，学习 React，你只需了解如何编写 JSX 风格的 React 组件就可以上手了。而不像其他前台框架较大的学习成本，比如 Angular，你要学习作用域，双向绑定，指令，控制器，模块，模板，依赖注入等等一系列知识。

react 特点

JSX

React 可以非常轻松地利用 JSX 创建用户交互界面。为你应用的每一个状态设计简洁的视图，在数据改变时 React 也可以高效地更新渲染界面。

你只需要告诉计算机你需要什么就行了，不需要关心怎么去做的。现在用统一通俗的例子来比较 jQuery 的命令和 React 的声明式。React 是一个聪明的建筑工人，而 jQuery 是一个比较傻的建筑工人，开发者你是一个建筑工程师，如果是 jQuery 这个建筑工人为你工作，你不得不事无巨细地告诉 jQuery“如何去做”，要告诉他这面墙要拆掉重建，那面墙上要新开一个窗户，反之，如果是 React 这个建筑工人为你工作，你所要做的就是告诉这个工人“我想要什么样子”，只需要把图纸递给 React 这个工人，他就会替你搞定一切，当然他不会把整个建筑拆掉重建，而是很聪明地把这次的图纸和上次的图纸做一个对比，发现不同之处，然后只去做适当的修改就完成任务了。

react 的解决方式：在第一构建出 DOM 树之后，还会构建出一个 Virtual DOM，是对 DOM 树的抽象，是一个 JavaScript 对象，重新渲染的时候，会对比这一次产生的 Virtual DOM 和上一次渲染的 Virtual DOM，对比发现差异之后，只需修改真正的 DOM 树时就只需要触及差别中的部分就行

组件化

创建好拥有各自状态的组件，再由组件构成更加复杂的界面。

无需再用模版代码，通过使用 JavaScript 编写的组件你可以更好地传递数据，将应用状态和 DOM 拆分开来。

数据决定视图，不用操作 DOM

React 让数据和业务处理逻辑与 UI 处理分开，并帮我们处理 UI，我们专注于数据与业务处理即可即可，更少的代码做更多的事。

灵活

可以与已知的库或框架良好地配合使用

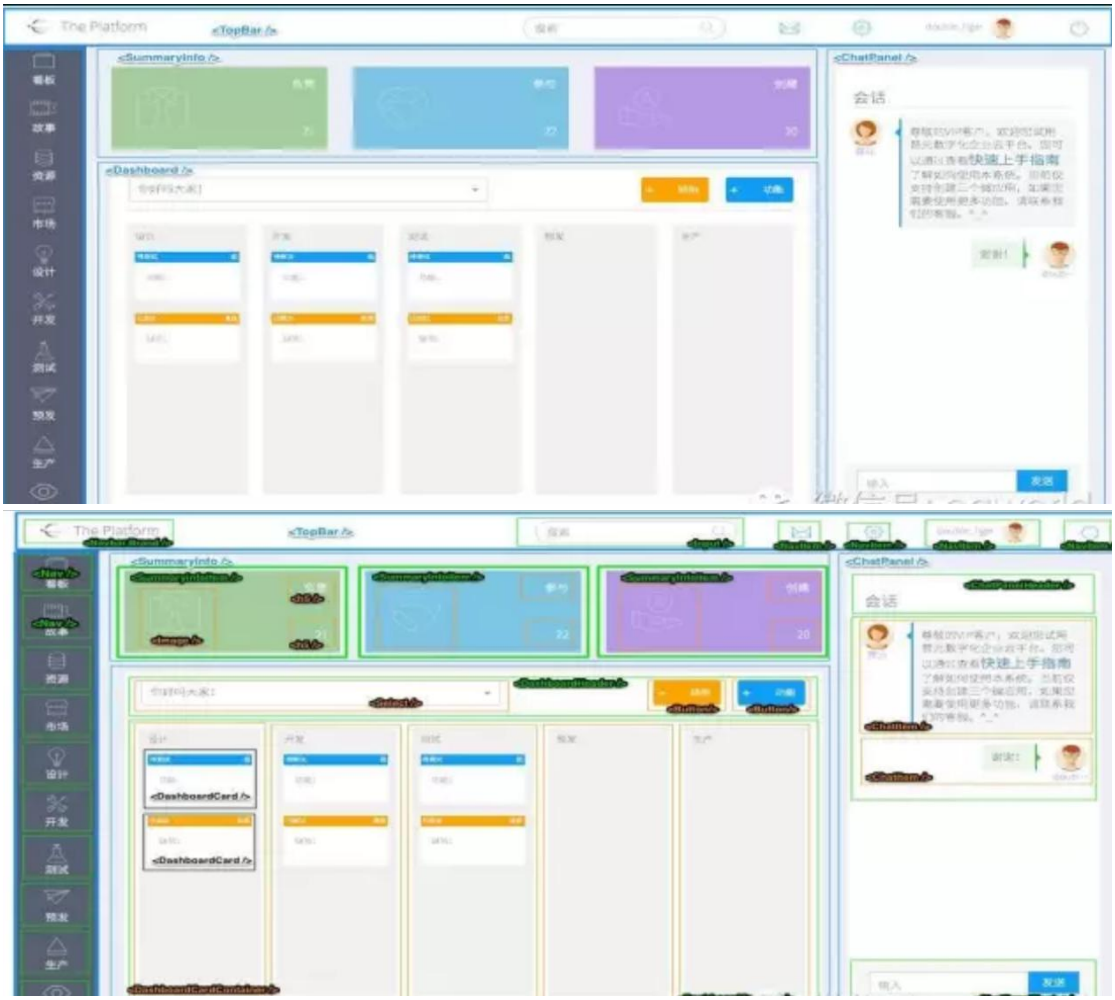
react 项目结构清晰

各个组件通过模块化来组织代码，各代码文件职责明确，便于开发，调试与维护。

强大的开发工具

Facebook 提供了 React Developer Tools 以方便开发者页面调试，包括查看组件的层次结构，实时查看和编辑组件的属性、状态等等，大大提高了开发者开发效率。目前已支持 Chrome/FireFox。

除了 React Developer Tools，另一个强大的工具就是 React Hot Loader。通过 React Hot Loader 结合 Webpack, 真正实现不用重启服务器、不用刷新浏览器，即可见刚刚修改代码后的页面，给予开发者与众不同的开发体验，加快开发效率。



React 组件化优势：

1. 组件职责单一，代码更易理解，减少重构、维护成本

由于组件职责单一，高内聚，不会出现以前改了一个组件上的小功能，还要级联更新其他依赖组件。

2. 组件之间松耦合，代码更易复用、扩展

上述例子中，卡片面板中，设计、开发、测试、预发、生产五种不同的卡片容器用的是同一个组件 DashboardCardContainer。
而以后如果我们想在卡片上添加新的功能，比如删除功能，我们只需修改 DashboardCard 组件就可以了，而不需要修改其他组件。

3. 各组件可同时交由不同开发人员开发，加快开发效率

如聊天面板和卡片面板完全可以交由不同开发人员开发，两者互不影响。

4. 代码更易测试，提高代码质量

在这种细粒度的组件划分下，各组件之间松耦合，方便编写前端测试代码，提高代码测试覆盖率，保证产品质量。

选择 React 的原因总结

简单易学，上手快
模块化组织代码
灵活易用
组件化构建界面
JSX 使编程更简单
虚拟 DOM 技术
数据驱动，单向数据绑定
强大的开发与调试工具
社区强大，各种组件和工具库应有仅有，只需 npm install 即可使用

如何利用 React 进行开发

第一步：把 UI 划分出组件层级

第二步：用 React 创建一个静态版本

第三步：定义 UI 状态的最小(但完整)表示

第四步：确定你的 State 应该位于哪里

第五步：添加反向数据流

传统 jquery(操作 data 与 dom) 对比 React(操作 data)

```
 <div>
  <button id="add_btn" type="button" name="button">add</button>
  <ul id="list"></ul>
  <input id="query_value" />
  <button id="query_btn" type="button" name="query">query</button>
</div>

<script src="./jquery-3.2.1.js" charset="utf-8"></script>
<script type="text/javascript">

    getRandom3(){
       Math.floor(Math.random () * 900) + 100;  //data
    }

    $(function(){
    //增加按钮
      $("#add_btn").click(function(event) { //dom
          var $list = $("#list"); //dom
          var liString = "<li>" + getRandom3() + "</li>";  //dom
          var newLi = $(liString);  //dom
          $list.append(newLi);  //dom
      });
    //查询按钮
      $("#query_btn").click(function(event) {  //dom
          var query = $("#query_value").val(); //dom
          var $list = $("#list");//dom
          $list.find('li').each(function(el){ //dom
            if(parseInt(el.text()) < parseInt(query)){ //data
              el.remove(); //dom
            }
          })
      });
    })
</script>
```

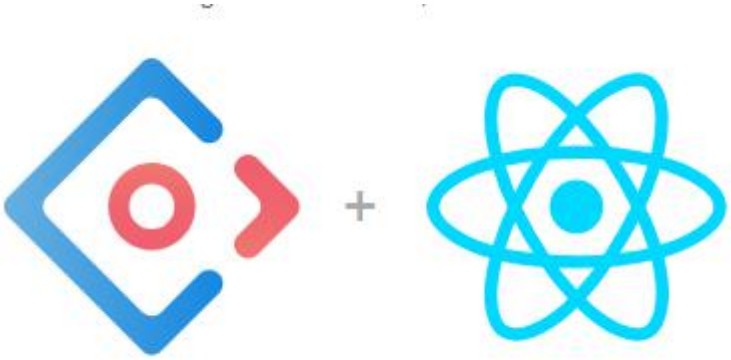
```
const getRandom3 = () =>  Math.floor(Math.random () * 900) + 100;

class App extends Component {
  state{
    query:0
    state = {values : []};
  }
  //增加按钮
  onAddClick = () => {
    this.setState({
      values : values.concat(getRandom3())
    });
  }
  //查询按钮
   onQueryClick = () => {
   const query = this.state.query;
   query&&this.setState({
      values : values.filter(item => item>query)
    });
  }

  render() {
  const {values} = this.state;
    return (
      <div>
        <button onClick={this.onAddClick}>add</button>
        <ul>
          {values.map(item => <li>item</li>)}
        </ul>
        <input value={query} onClick=(value=>{this.setState({value})})/>
        <button onClick={this.onQueryClick}>query</button>
      </div>
    );
  }
}
```

数据驱动，屏蔽 DOM 操作，逻辑与 UI 分离

结构清晰，组件状态明确

复用方便，import *加上<*/>即可

Ant Design(antd)



简介

Ant Design 是一套蚂蚁金服开发维护的，基于 React 技术的前端组件库，组件丰富，用法简洁。

谁在使用

蚂蚁金服

阿里巴巴

口碑

新美大

滴滴

示例介绍

InputNumber 数字输入框： 3 台机器

我是标题： 唧唧复唧唧木兰当户织呀 链接文字

* Switch 开关：

* Slider 滑动输入条：

A B C D E F G

* Select 选择器：

* 级联选择： Please select

* DatePicker 日期选择框： 请选择日期 - 请选择日期

* TimePicker 时间选择器： 请选择时间

选项： 选项一 选项二 选项三

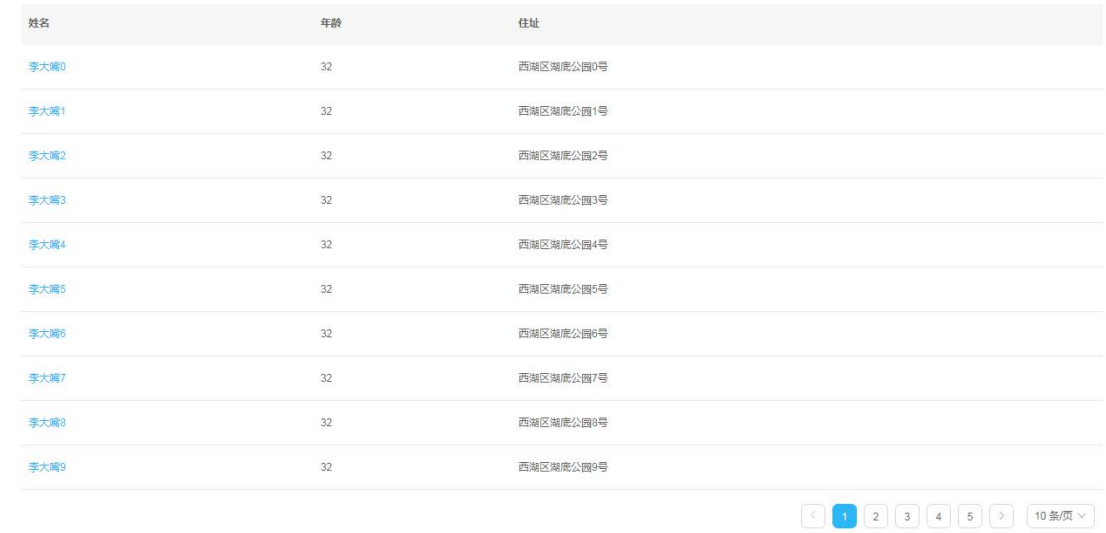
logo图： 点击上传

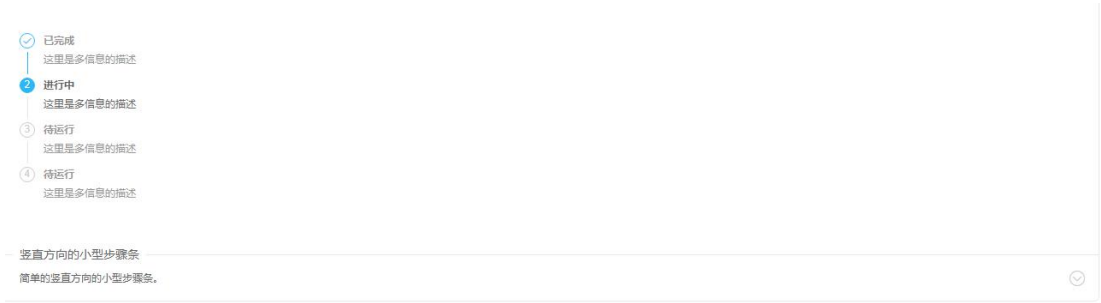
提示信息要长长长长长长长长长长长长长长长

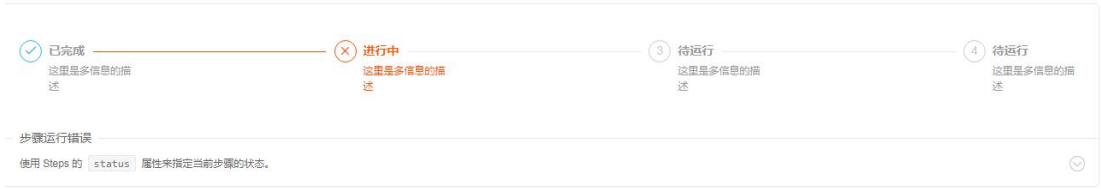
确 定


失败校验：
无效选择
请输入数字和字母组合
警告校验：
前方高能预警
校验中：
我是被校验的内容
信息审核中...
成功校验：
我是正文
警告校验：
前方高能预警
失败校验：
无效选择
请输入数字和字母组合
行内校验：
请选择日期
-
请选择日期
请选择正确日期


第一个 Modal
对话框的内容
对话框的内容
对话框的内容
取 消
确 定
希望跳转页面以致打断工作流程时，可以使用 Modal 在当前页面正中打开一个浮层，承载相应的操作。
从框询问用户时，可以使用精心封装好的 ant.Modal.confirm() 等方法。







模块化，组件化构建，使用方式简单

```
import { Form, Input, Button, Checkbox } from 'antd';

<Form inline onSubmit={this.handleSubmit}>
    <FormItem>
         <Checkbox {...getFieldProps('agreement')}>记住我</Checkbox>
    </FormItem>
    <Button type="primary" htmlType="submit">登录</Button>
</Form>
```

自研脚手架与 mock 服务器的使用

脚手架是什么？为什么要用脚手架？

在实际的开发过程中，从零开始建立项目的结构是一件让人头疼的事情，所以各种各样的脚手架工具应运而生。较流行的的 `yoeman`，`create-react-app` 和 `vue-cli` 便是当中之一。它们功能丰富，但最核心的功能都是能够快速搭建一个完整的项目的结构，开发者只需要在生成的项目结构的基础上进行开发即可，非常简单高效。

我们的脚手架做了什么工作？

上述提高的这些技术，都非常优秀，但是 react 如何使用？又需要哪些周边技术?项目如何创建?模块化如何组织？webpack 如何使用？
本脚手架就是为了解决这些疑问而生！

如何使用？非常简单！

一条指令：npm start 即可使用。

脚手架做了哪些工作？

一套包含了 react，react-router-dom,axios,ant design(蚂蚁金服出品 react UI 组件库，组件丰富)，webpack(前端构建平台)的脚手架，

用于构建前端的项目的基础，使用简便，不用增加任何配置即可拥有以下功能！

* 启动独立的前端开发服务器

* 兼容 ie8，让 react webpack antd 等现代前端技术可以用在开发 ie8 的应用上

* 支持用 ES6 语法开发，获取全新开发体验

* 开发时模块化，自动响应代码变化进行打包编译

* 支持 HappyPack 并行快速打包

* 支持 CommonsChunkPlugin，自动抽取公用模块

* 支持模块分析：start 后访问 localhost:9998

* 自带前端独立开发服务器，命令行 npm run start 后访问 localhost:8888 即可，检测源代码变化自动刷新浏览器

* 一条指令生产环境打包编译

* 支持懒加载，按需加载当前页面所需模块

```
* 支持开发时源代码映射，将打包后的代码映射到源代码以方便调试

* 支持多页面应用

* css 提取

* css 模块化

* 支持 less

* js 语法检查

* 生产环境自动删除注释以及 console 语句,简化代码

* js，css 等静态文件压缩/混淆，提高安全性和网络传输速度

* 支持 js，css 等静态文件 hash 码命名，以防止用户浏览器缓存旧的静态文件（没改变的文件 hash 不会变）

* 支持反向代理,配置简单(dev-proxy-config.js)，方便 api 接口切换到后端服务器或 mock 数据服务器
```

详细可见脚手架项目文件根目录下的 README.md 文件。

自研 mock 服务器

自研 mock 数据简介

在开发过程中，为了提高开发效率，前端会用到模拟数据，有的时本地变量，有的是 json 文件引用，有的时 Mock 服务。

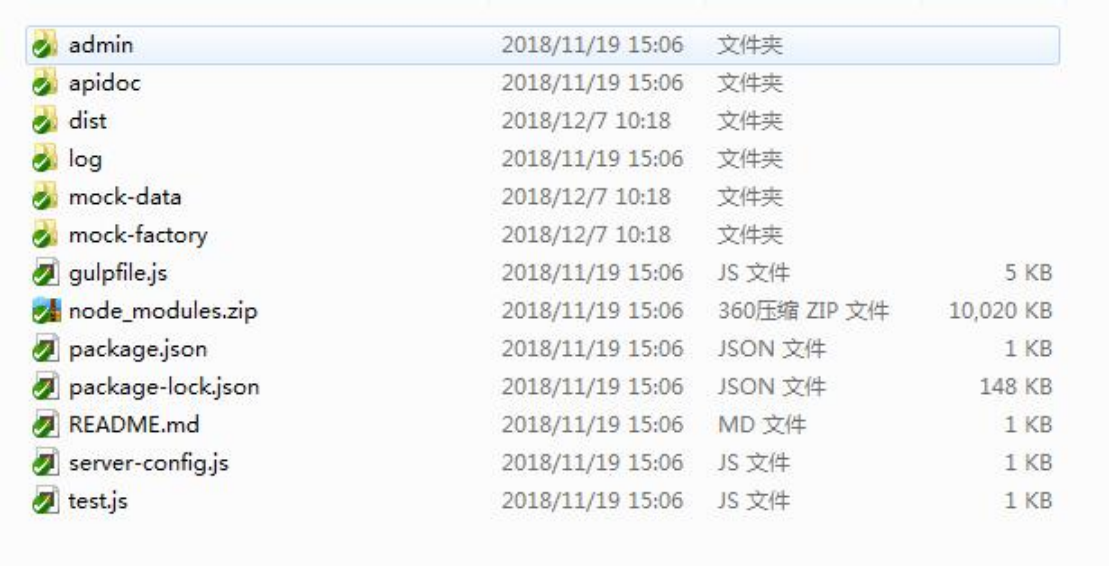
其中，Mock 服务时最接近真实环境的数据模拟方案。

本自研 Mock 服务器基于 Node.js 开发，可独立使用。

支持返回各种基本数据，支持返回 JSON 数据，支持返回文件流，支持各种 http 状态模拟，支持数据模板，可以此根据规则自动生成模拟数据！

```
var data = Mock.mock({
    // 属性 list 的值是一个数组，其中含有 1 到 10 个元素
    'list|1-10': [{
        // 属性 id 是一个自增数，起始值为 1，每次增 1
        'id|+1': 1
    }]
});
// 输出结果
[1, 2, 3, 4, 5, 6, 7, 8]
```

项目目录如下：



mock 数据运行机制

上文已经说到，前端是独立的应用，运行在服务器中，负责返回静态文件，动态 API 接口则请求到后端服务器。在开发时，后端服务器的相关 API 接口可能尚未开发完成或者不可连接，前端不可避免的要使用模拟的数据维持前端的完整性运行。

而自研 Mock 服务器便是为此而生。

开发环境下，可把 API 请求发送到 Mock 服务器，完成最接近真实环境的模拟数据请求。

如何使用

Mock 服务器基于 Node.js 开发，可独立使用。先将代码包复制到自己电脑，解压 node_modules.zip 到当前文件夹。

你只需要在 mock-data 文件夹内添加任意数量的 js 获取 json 文件即可；然后在命令行中敲入指令：gulp 即可开启模拟服务器！

mock-data 内的 js 和 json 示例如下：

模块化开发，React，Webpack，Ant Design 等技术在开发平台新前端的实例应用

得益于之前版本也是采取前后端分离架构，技术切换成本非常低，可扩展性很大，开发平台新前端已应用上述所有技术。

具体见项目讲解。

开发环境及生产环境下的测试与调试方案

工欲善其事必先利其器。好的工具能让我们的工作效率得到不少提高。下面介绍几种 React 前端开发生态下的测试调试利器。

开发调试工具

webpack-dev-server

启动前端独立应用服务器，支持反向代理，方便测试环境下的连接模拟数据服务器调试，以及和后台真实数据的对接。

脚手架已经包含此功能。

source-map 源代码映射

由于开发环境下的源代码需要编译后发送到浏览器运行，所以会遇到代码调试的困难。而 source-map 可以创建源代码映射，让浏览器可以链接到源代码。

脚手架已经包含此功能。

react-developer-tools 谷歌浏览器插件

安装在谷歌浏览器，可方便调试。

安装包已提供，请自行安装。

JetBrains IDE Support

JetBrains 旗下的 IDE 使用的谷歌开发插件，可 IDE 实时前端调试，响应式开发，无需刷新。
安装包已提供，请自行安装。

vs code

插件丰富，扩展性高，性能高，最流行的前端开发软件之一。

webstorm

最智能的前端开发 IDE 之一，功能强大，但是较吃内存。

whistle

Node.js 环境下运行，前端网络抓包调试利器，可用于线上系统的前端代码调试。
安装包已提供，请自行安装。

Proxy-SwitchyOmega-Chromium-2.5.15

浏览器网络代理插件，配合 whistle 进行线上环境的前端调试。