

网易微专业之《前端开发工程师》

学习笔记

开始时间：2016.2.27

《产品前端架构》

技术选型

模块化

以下讨论都是基于 JavaScript 的模块组织（每个模块均以文件形式组织），而非工程的模块化。

语言的模块支持

- java: import
- C#: using
- CSS: @import
- javascript: None! (JavaScript 中并不存在模块组织在并不支持 ,于是产生了很多，模块系统。)

模块的职责：

- 封装实现
- 暴露接口
- 声明依赖

模块的使用：

反模式（**Anti-Pattern**）：没有应用任何模块系统

1. 封装性无
2. 接口结构不明显

1. 没有依赖声明
2. 使用全局状态

Anti-Pattern

math.js

```
function add ( a, b ){
    return a + b
}
function sub ( a, b ){
    return a - b
}
```

calculator.js

```
var action = "add";

function compute( a, b ){
    switch (action){
        case "add": return add(a, b)
        case "sub": return sub(a, b)
    }
}
```

字面量（Object Literal）

1. 结构性好
2. 访问控制

1. 同样没有依赖声明

Object Literal

math.js

```
var math = {
    add: function add ( a, b ){
        return a + b
    },
    sub: function mul ( a, b ){
        return a - b
    }
}
```

calculator.js

```
var calculator = {
    action: 'add',
    compute: function compute ( a, b ){
        switch (action){
            case "add": return math.add( a, b )
            case "sub": return math.sub( a, b )
        }
    }
}
```

IIFE(Immediately-invoked Function Expression): 自执行函数表达式

1. 访问控制
2. 无依赖声明

```
calculator-1.js

var calculator = (function() {
  var action = "add"

  return {
    compute: function(a, b) {
      switch (action) {
        case "add":
          return math.add(a, b)
        case "sub":
          return math.add(a, b)
      }
    }
  }
})();
```

1. 显示依赖声明
2. 仍然污染了全局变量
3. 必须手动进行依赖管理

```
calculator-2.js

var calculator = (function( m )
  var action = "add"
  function compute( a, b ){
    switch (action) {
      case "add":
        return m.add(a, b)
      case "sub":
        return m.add(a, b)
    }
  }
  return {
    compute: compute
  }
})( math )
```

命名空间（**Namespace**）:命名空间可以解决全局变量的污染的问题。

```
math.js

namespace("math", [], function(){

  function add(a, b){ return a + b }
  function sub(a, b){ return a - b }

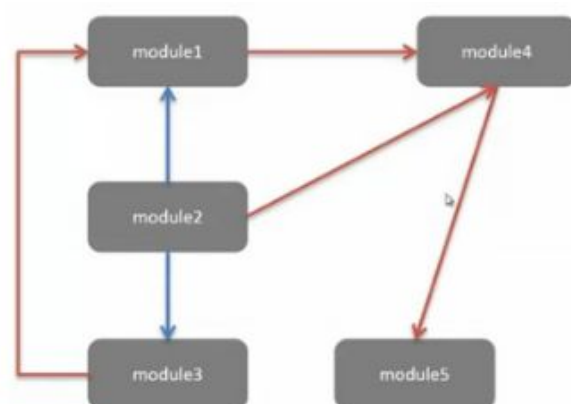
  return {
    add: add,
    sub: sub
  }
})

calculator.js

namespace("calculator", ["math"], function( m ){

  var action = "add"
  function compute( a, b ){
    return m[action](a, b)
  }
  return {
    compute: compute
  }
})
```

依赖管理（**dependency manage**）



```
<body>
  <script src='module5.js'></script>
  <script src='module4.js'></script>

  <script src='module1.js'></script>
  <script src='module3.js'></script>

  <script src='module2.js'></script>
</body>
```

模块系统

本笔记由西风潇潇编写，欢迎浏览博客访问更多内容：<http://www.xifengxx.com>

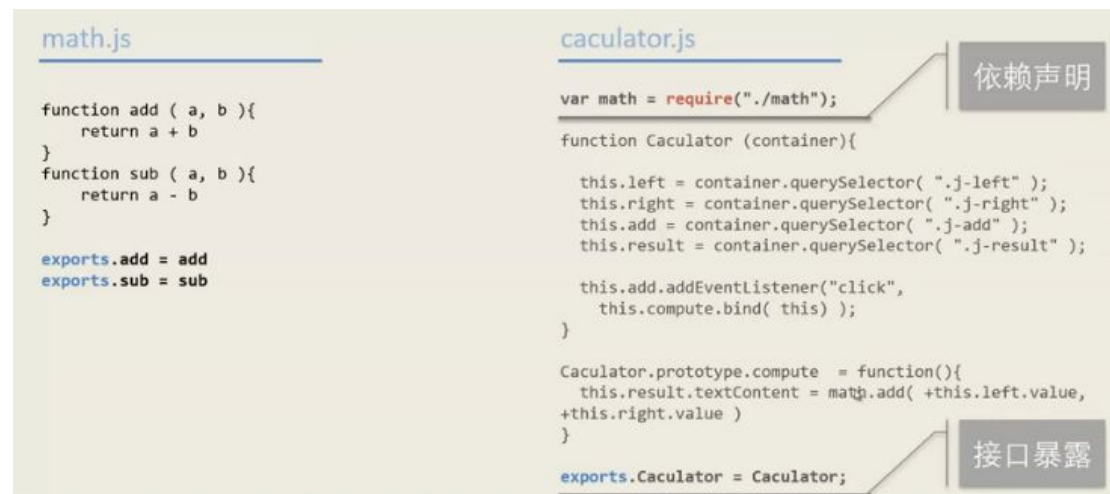
职责：

- 依赖管理（加载、分析、注入、初始化）
- 决定模块的写法。

常用的模块系统有：commonjs, AMD, ES6 module（语言级别的模块化）

Commonjs/module

它是一个模块规范，通常适用于非浏览器环境（NodeJS）



优点

- 依赖管理成熟可靠
- 社区活跃，规范接受度高
- 运行时支持，模块定义非常简单
- 文件级别的模块作用域隔离
- 可以处理循环依赖

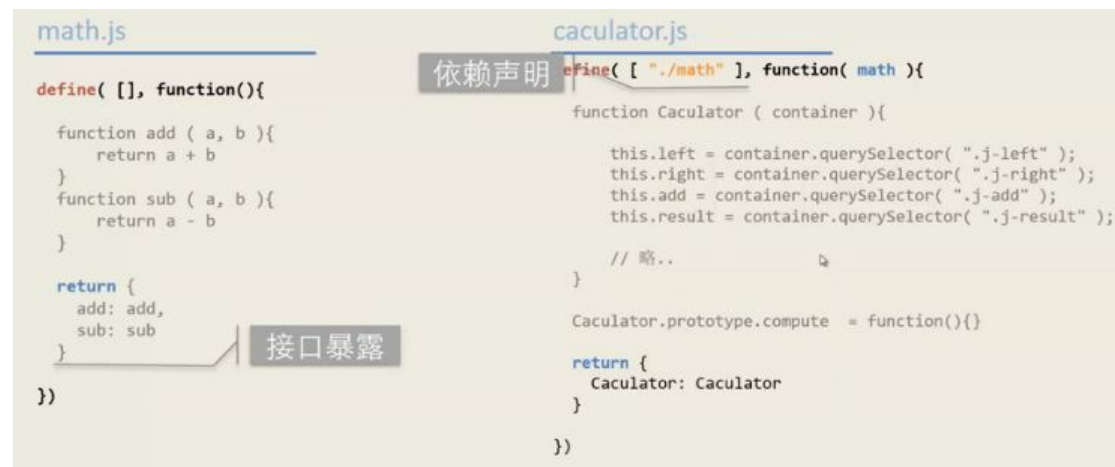
缺点

- 不是标准组织规范
- 同步的 `require`，没有考虑浏览器环境（可以使用

Browserify/webpack/component 来解决）

AMD(Asynchronous Module Definition):

适合：作用于异步的 **IO** 环境



Simplified CommonJS Wrapping

使用同样的 CommonJS 的依赖管理书写方法 ,之后在使用正则表达式来提取依赖列表。

```
define(function( require, exports ){  
    var math = require("./math");  
  
    function Caculator ( container ){  
  
        this.left = container.querySelector( ".j-left" );  
        this.right = container.querySelector( ".j-right" );  
        this.add = container.querySelector( ".j-add" );  
        this.result = container.querySelector( ".j-result" );  
  
        //略...  
    }  
  
    Caculator.prototype.compute = function(){/**/}  
  
    exports.Caculator = Caculator;  
  
})
```

```
> factory.toString()  
< "function ( require, exports ){  
    var math = require("./math");  
  
    function Caculator ( container ){  
  
        this.left = container.querySelector( ".j-left" );  
        this.right = container.querySelector( ".j-right" );  
        this.add = container.querySelector( ".j-add" );  
        this.result = container.querySelector( ".j-result" );  
  
        //略...  
    }  
  
    Caculator.prototype.compute = function(){/**/}  
  
    exports.Caculator = Caculator;  
}"  
> /require\[("[^"]*)"(\[""]*\[""]*)\]\.exec(factory.toString())[1]  
< "./math"  
> |
```

Loader Plugins

允许调用处理脚本外的其他资源（例如 HTML 与 CSS 文件），这样就可以形成一个完整的组件。

完整组件 = 结构 + 逻辑 + 样式



优点

- 依赖管理成熟可靠
- 社区活跃，规范接受度高
- 转为异步 IO 环境打造，适合浏览器环境
- 支持类似 Commonjs 的书写方式
- 通过插件 API 可支持加载非 js 资源
- 成熟的打包构建工具，并可结合插件使用

缺点

- 模块定义繁琐，需要额外嵌套
- 只是库级别的支持，需要引入额外的库
- 无法处理循环依赖
- 无法实现条件加载

ES6 module（未来的模块化标准，目前支持较少）



优点

- 是真正的规范，未来的模块标准
- 语言级别的关键字支持
- 适用于所有 JavaScript 运行时，包括浏览器
- 可处理循环依赖

缺点

- 规范未达到稳定级别
- 基本还没有浏览器支持
- 鲜有项目使用，即使有大量的 6to5 的 transpiler

Systemjs(动态模块加载器)

- 支持加载 AMD
- 支持加载 Commonjs
- 支持加载 ES6
- 支持加载 Transpiler 可支持任意资源

模块系统对比

- IIFE: 没有解决核心的依赖分析和依赖注入的问题。
- AMD:可以直接使用，库基本的支持。
- CommonJS:可以直接使用，在运行时的支持。
- ES6:语言本身的支持。
- 使用插件工具，可以将后三种模块管理系统打包成 IIFE,也可以进行相互转换。

问题：市面上这么多种模块系统，它们之间可以相互转换吗？

AMD、COMMONJS、CMD、UMD、ES6 Module、IIFE... 这么多的模块写法，一旦你选择了一种模块写法，那它在另一个系统中就可能无法运行了。值得庆幸的是，现在越来越多的工具可以帮助我们将 js 从一种模块写法转换为另一种写法，你能帮助同学们列举出一个或多个转换工具吗？

browserify：实现浏览器读取 CommonJS 模块

webpack：同时支持 CommonJS 和 AMD 形式的模块，对于不支持的模块格式，还可以对模块进行 shimming

uRequire：支持多种格式之间的互换，但暂时不包括 ES6

systemJS：几乎支持任意资源

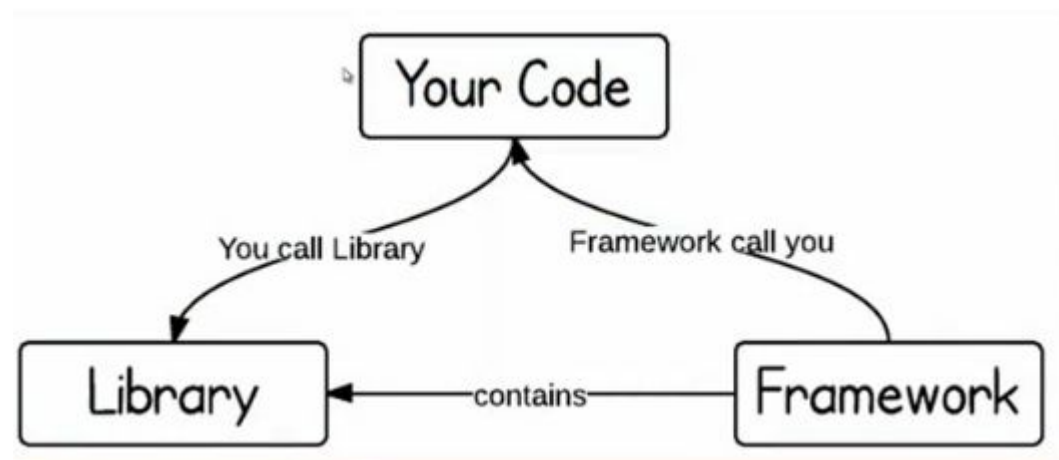
框架（基于 JavaScript 的框架）

库(Library)

- 针对特定问题的解答
- 不控制应用程序
- 被动的被调用

框架 (Framework)

- Inverse of control:控制反转，
- 决定应用程序生命周期
- 一般会集成大量的库



一个库是否是框架，取决于你从什么角度去看待它。

解决方案

- DOM
- Communication：通信
- Utility：工具库
- Templating：模板技术
- Component：组件
- Routing：路由（单页系统中尤其重要）
- Architecture：架构

为什么使用外部解决方案？

- 开发效率
- 可靠性：浏览器兼容性/测试覆盖
- 更好的配套：文档/DEMO/工具
- 设计的更好
- 专业性

什么时候不去使用外部解决方案？

- 问题过于简单
- 备选框架质量与可控性无法保证
- 无法满足当前业务需求
- 团队中已有相关积累

实际项目中如何使用？

- 开发：基于一个外部模块系统，自由组合。
- 半开放：基于一个定制过的模块系统，内容-外部的解决方案共存。
- 大教堂：深度定制的模块系统，很少需要引入外部模块。

DOM

与其相关的操作有：*Selector(选择器)*、*Manipulation(DOM操作)*、*Event(DOM)*、*Animation*。

DOM 操作的职责：

- 提供便利的 DOM 查询、操作、移动等操作
- 提供事件绑定及事件代理等支持
- 提供浏览器特性检测、UserAgent 侦测
- 提供节点属性、样式、类名的操作
- 所有以上操作实现目标平台的跨浏览器支持

常用的 DOM 库有

- **jQuery**：使用链式接口

- **zepto.js**
- **Mootools**：使用**原生 DOM 对象**，通过直接扩展了 DOM 原生对象，严格遵守 Command-Query 命令查询规范。

jQuery	zepto.js	
<pre>\$("button.j-submit") .addClass("disable") .attr("title", "Waiting...") .html("Waiting...") .on("click", showWarning) .appendTo('.j-form')</pre>	<pre>\$("button.j-submit") .addClass("disable") .attr("title", "Waiting...") .html("Waiting...") .on("click", showWarning) .appendTo('.j-form')</pre>	
<div> A COMPACT JAVASCRIPT FRAMEWORK</div> <pre>31 32 \$("button.j-submit") 33 34 .addClass("disable") 35 36 .setAttribute("title", "Waiting...") 37 38 .set("html", "Waiting...") 39 40 .addEvent("click", showWarning) 41 42 .inject('.j-form')</pre>		
<div> A COMPACT JAVASCRIPT FRAMEWORK</div> <div>简介<ul style="list-style-type: none">★: 730Size : 96K兼容性: IE6+优点<ul style="list-style-type: none">概念清晰，没有包装对象接口设计优秀源码清晰易懂不局限于Dom和Ajax缺点<ul style="list-style-type: none">扩展原生对象（致命）社区衰弱</div>	<div> write less, do more.</div> <div>简介<ul style="list-style-type: none">★: 33728Size : 94K兼容性: IE6+优点<ul style="list-style-type: none">社区强大, 普及率高包装对象, 不污染原生基本上专注于Dom缺点<ul style="list-style-type: none">包装对象, 容易混淆接口两义性社区水平层次不齐, 容易踩坑</div>	<div> 云课堂</div> <div>简介<ul style="list-style-type: none">★: 8236Size : 25K兼容性: IE10+优点<ul style="list-style-type: none">小, 启动快接口与jQuery兼容提供了简单的手势缺点<ul style="list-style-type: none">与jQuery不能做到100%对应支持浏览器少, 功能较弱</div>

- mootools: 最好的源码阅读学习的资源

- jQuery:最稳妥的方案
- zepto.JS:移动端的备选品

DOM 专业领域

<h3>手势</h3> <ul style="list-style-type: none">● Hammer.js● ★: 10029● Size : 12K● Issue: 444/600● Desc: 常见手势封装, 包括 tap, hold, transform, swipe 等等, 并支持自定义扩展	<h3>局部滚动</h3> <ul style="list-style-type: none">● iscroll.js● ★: 5200● issue: 380/640● Size : 13K● Desc: 移动端position:fix + overflow: scroll的救星
<h3>高级动画</h3> <ul style="list-style-type: none">● Velocity.js● ★: 10029● Size : 12K● Issues: 400/411● Desc: 复杂动画序列实现, 不仅限于dom	<h3>视频播放</h3> <ul style="list-style-type: none">● video.js● ★: 8300● Size : 101K● Issue: 1160 / 1290● Desc: 类似原生video标签的使用方式, 对低级浏览器回退到flash播放

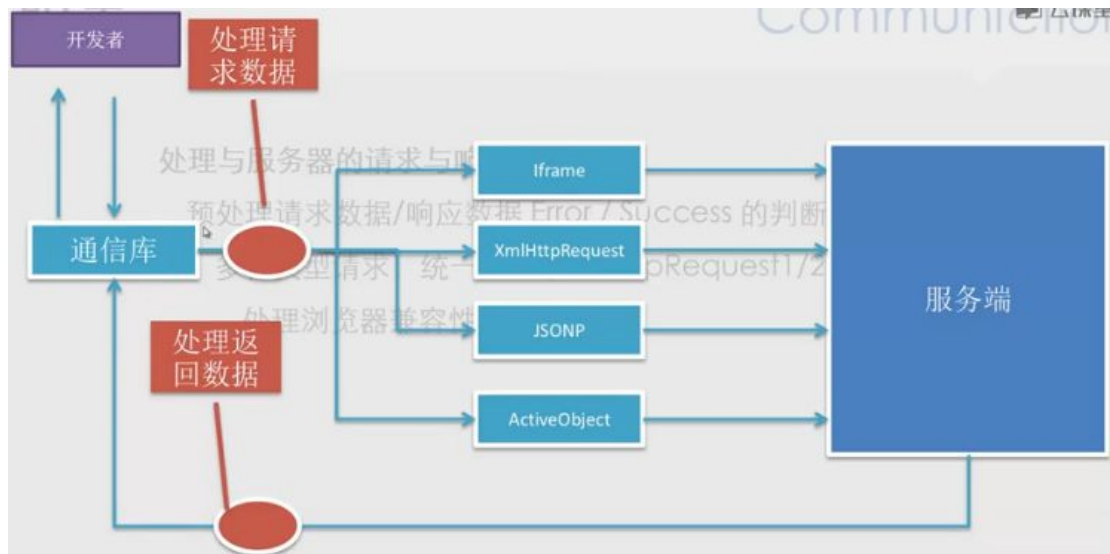
选取标准: Issue 总量与解决率比 Star 更关键

Communication(通信)

与其相关的有 *XMLHttpRequest*、*Form*、*JSONP*、*Socket*。

它的主要职责则：

- 处理与服务器的请求与相应
- 预处理请求数据/响应数据 Error/Success 的判断封装
- 多种类型请求，统一接口（XMLHttpRequest1/2、JSONP、iFrame）
- 处理浏览器兼容性



Communication 库:

Request:

优点: JSONP 支持; 稳定/IE6 + support; CORS 跨域; Promise/A 支持。

qwest:

优点: 更小的代码量; 支持 XMLHttpRequest2; CORS 跨域; 支持高级数据类型, 如 ArrayBuffer, Blob 和 FormData.

Socket.io (针对实时性要求高的需求)

优点: 实时性; 支持二进制数据流; 智能自动的回退支持 (非二进制数据流); 多种后端语言支持。

Utility (Lang): 函数工具包

与其相关的有 *函数增强 & Shim* (保证实现与规范一致)、*Flow Control*。

它的主要职责:

- 提供 JavaScript 原生不提供的功能
- 方法门面包装, 使其更易于使用

- 异步队列/流程控制等等

Utility 库：

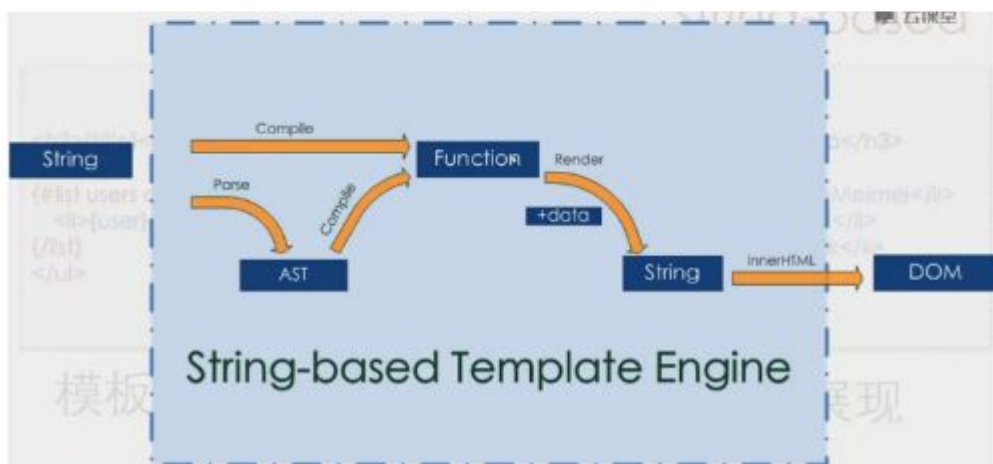
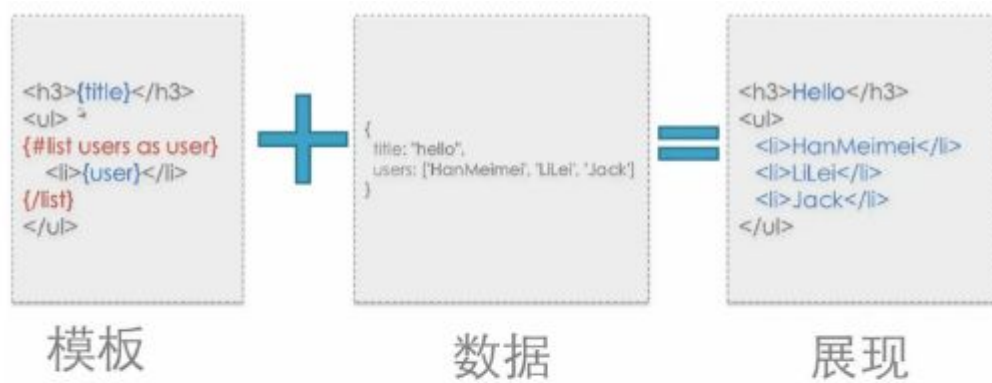
Shim	Extension
<ul style="list-style-type: none">● es5-shim (部分支持)<ul style="list-style-type: none">- Github: es-shims/es5-shim- ★ : 3500- Size: 53K● es6-shim (部分支持)<ul style="list-style-type: none">- Github: paulmillr/es6-shim- ★: 1000- Size: 38K	<ul style="list-style-type: none">● underscore<ul style="list-style-type: none">- ★ : 13500- Size: 16.5K- 兼容: IE6+● Lodash<ul style="list-style-type: none">- ★: 8400- Size: 50K- 兼容: IE6+- lodash是underscore的高性能版本，方法大部分都是runtime编译出来的。

Templating:模板技术

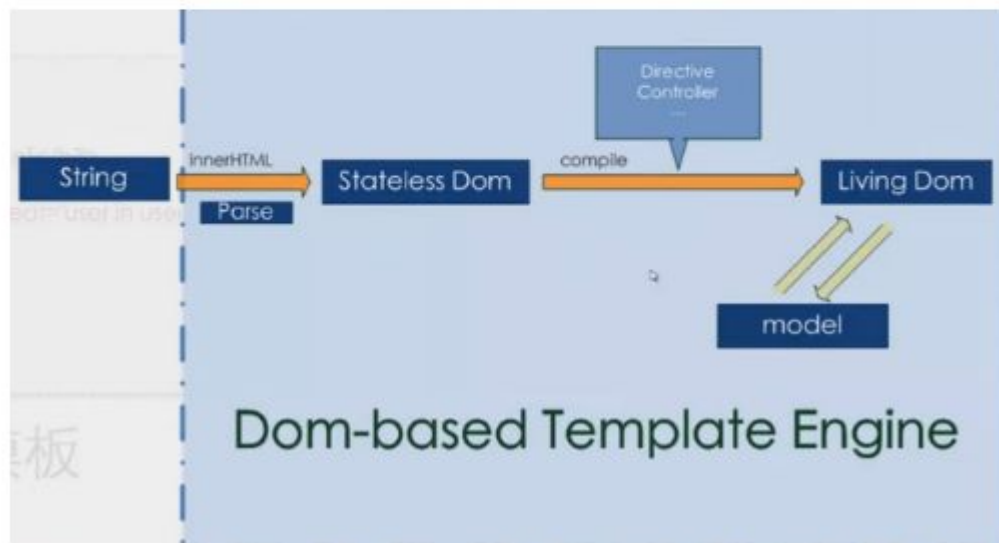
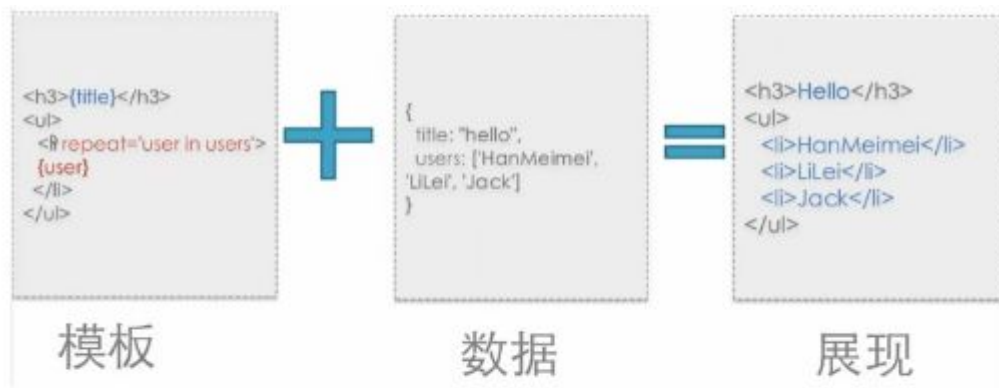
与其相关的有 *String-based*、*DOM-based*、*Living Template*。

String-based:基于字符串的模板

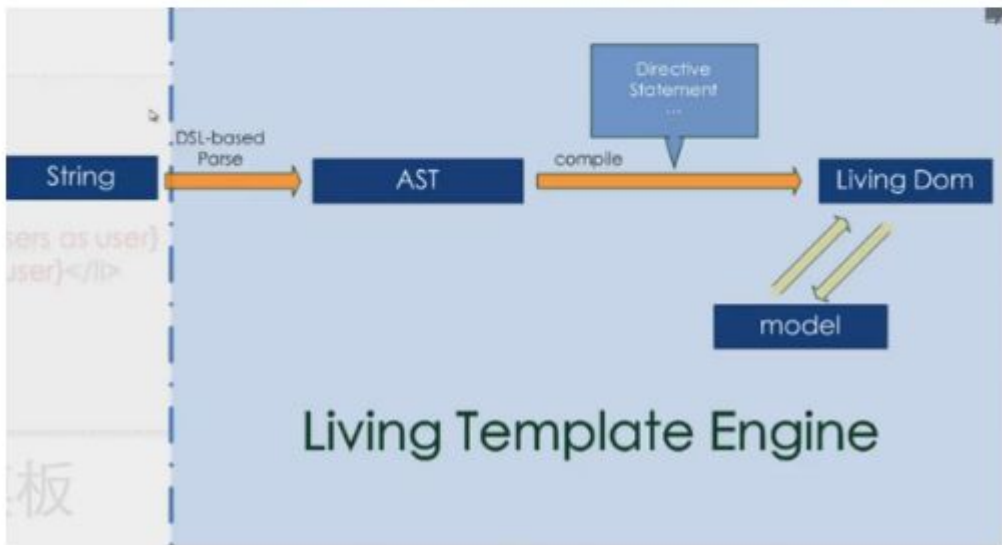
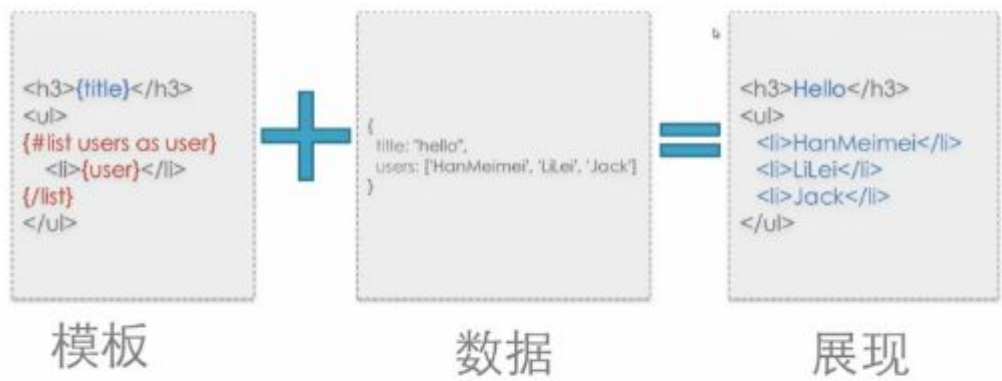
本笔记由西风潇潇编写，欢迎浏览博客访问更多内容：<http://www.xifengxx.com>



DOM-based: 基于 DOM 的模板



Living-Template:活动模板技术



String-based	Dom-based	Living-template
<p><u>总览</u></p> <ul style="list-style-type: none">初始化时间: ★★★动态更新: ✖dom无关: ★★★语法: ★★★学习成本: ★SVG支持: ✖安全性: ★ <p><u>解决方案</u></p> <ul style="list-style-type: none">dustjshogan (mustache实现之一)dot.js	<p><u>总览</u></p> <ul style="list-style-type: none">初始化时间: ★动态更新: ★★★dom无关: ✖语法: ★学习成本: ★★★SVG支持: ★★安全性: ★ <p><u>解决方案</u></p> <ul style="list-style-type: none">AngularjsVuejsKnockout	<p><u>总览</u></p> <ul style="list-style-type: none">初始化时间: ★★动态更新: ★★★dom无关: ★★语法: ★★学习成本: ★★SVG支持: ★★安全性: ★★★ <p><u>解决方案</u></p> <ul style="list-style-type: none">RegularjsRactivejshtmlbar

Component:组件

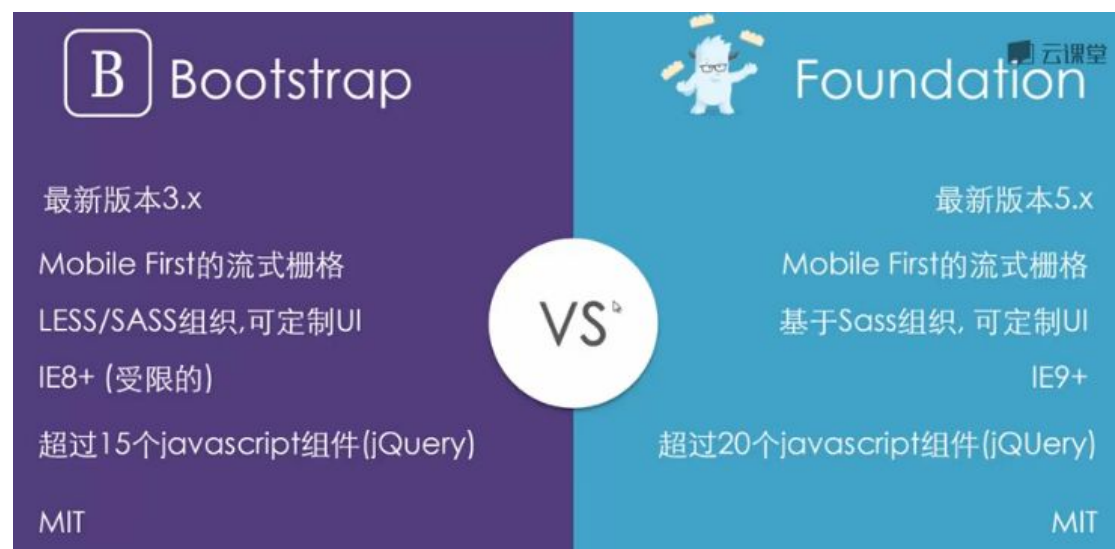
本笔记由西风潇潇编写，欢迎浏览博客访问更多内容：<http://www.xifengxx.com>

与其相关的有 *Modal*、*Slider*、*DatePicker*、*Tabs*、*Editor* (其为产品开发中最耗时也是最必要的一部分) 。

它的主要职责：

- 提供基础组件 CSS 支持
- 提供常用组件，如 Slider,Modal
- 提供声明式的调用方式 (Optional)

Component:组件库:



非 jQuery 版本的 Bootstrap

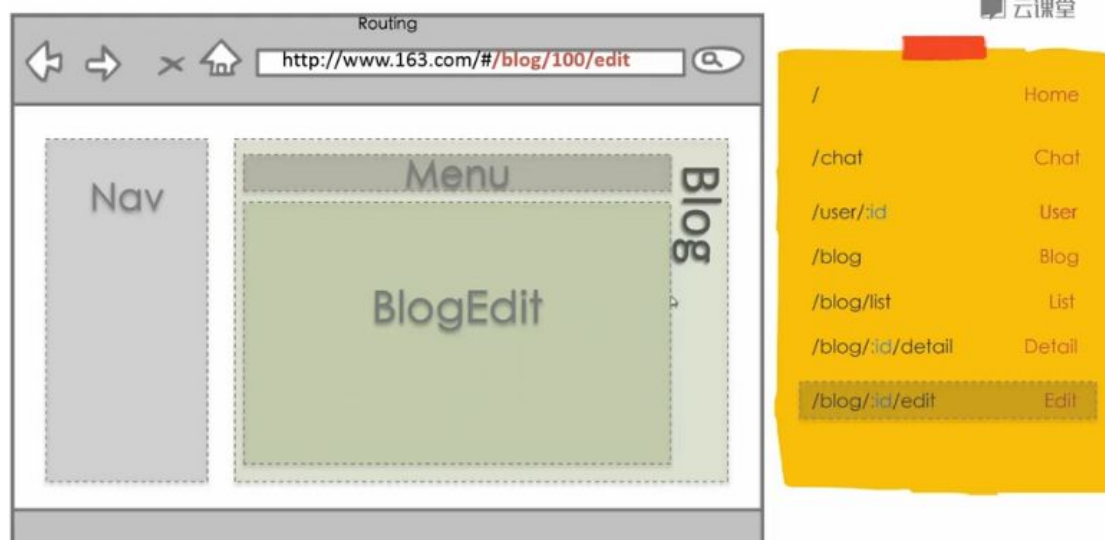
- Knockout-Bootstrap
- AngularUI Bootstrap
- React Bootstrap

Router:路由

与其相关的有 *Client Side*、*Server Side*。

它的主要职责：

- 监听 URL 变化，并通知注册的模块
- 通过 JavaScript 进行主动跳转
- 历史管理
- 对目标浏览器的兼容性支持



routing 库:

<u>page.js</u> <ul style="list-style-type: none">★: 2029Size : 6.2K兼容: IE8+Desc: 类似Express.Router的路由规则的前端路由库	<u>crossroad.js</u> <ul style="list-style-type: none">★: 970Size : 7.5KLast update: 2 yearDesc: 老牌Routing库, API定义较为繁琐
<u>Director.js</u> <ul style="list-style-type: none">★: 2443Size : 10K兼容: IE6+Desc: 可以前后端使用一套规则来定义路由.	<u>Stateman</u> <ul style="list-style-type: none">★: 105Size : 10K兼容: IE6+Desc: 用于处理深层复杂路由的独立路由库

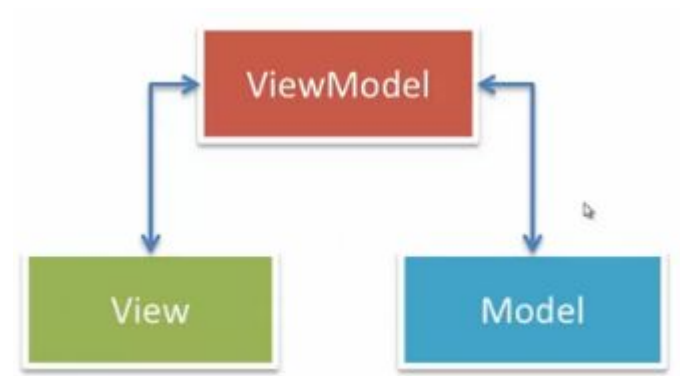
Architecture (解耦) : 架构

与其相关的有 *MVC*、*MVVC*、*MV**, 解耦又可以通过很多方式来实现 (例如事件、分层)。

它的主要职责

- 提供一种范式帮助 (强制) 开发者进行模块解耦
- 视图与模型分离
- 更容易进行单元测试
- 更容易实现应用程序扩展

以 MVVM 为例 :



Model :

数据实体，比如 Car、Person 等，它们用于记录应用程序的数据

View :

展示友好的界面，它是数据的定制反映，它包含样式结构定义以及 VM 享有的声明式数据、数据绑定

ViewModel :

其为 View 与 Model 的粘合剂，它通过绑定、事件与 View 交互，并可以调用 Service 处理数据持久化，当然也能通过数据绑定将 Model 的变动更新到 View 中。

MV* != SPA(单页系统)

Routing 是 MV*系统的可定位状态的信息来源。

TodoMVC : <http://todomvc.com/>

本笔记由西风潇潇编写，欢迎浏览博客访问更多内容：<http://www.xifengxx.com>

参考网站：

1. <https://www.javascripting.com/>
2. <https://www.javascriptoo.com/>
3. <http://microjs.com/>
4. 一个对前端模板技术的全面总结：<http://www.tuicool.com/articles/qMJ77r>

开发实践

系统设计

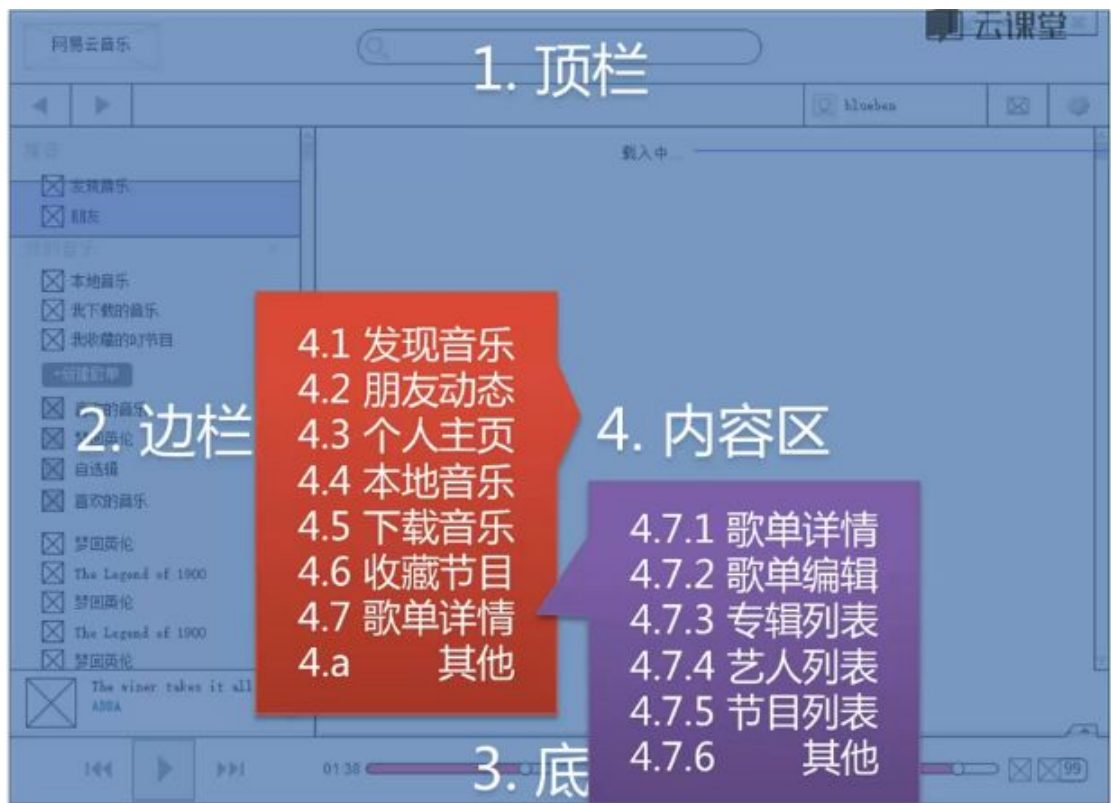
综合运用课程案例---**网易云音乐**，并且主要关注前端工程师的工作职责，不包含其他工程师的职责规范。

交互文档说明

通过交互文案来了解用户行为与异常提示。

系统分解：

- 注册登录密码
- 系统主框架：
 - 顶栏：
 - ◆ 搜索
 - ◆ 账号
 - ◆ 消息
 - ◆ 设置
 - 边栏
 - ◆ 歌单操作
 - ◆ 其他
 - 底栏
 - ◆ 播放器
 - ◆ 播放列表
 - ◆ 歌曲详情
 - 内容区
 - ◆ 发现音乐
 - ◆ 朋友动态
 - ◆ 个人主页
 - ◆ 本地音乐
 - ◆



系统分解必须对照交互稿做到百分之百的对应，不能漏掉任何一个模块。 后续的开发与评估都需根据此分解进行。

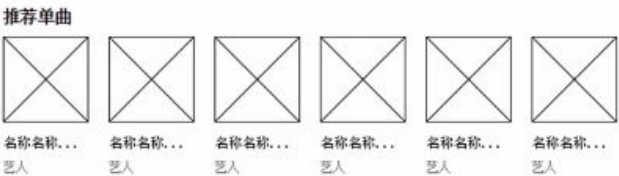
接口设计

分析模块交互设计，设计前后端交互接口，并定义数据类型、模板资源、异步接口、以及页面摘要，并和其他端角色配合，完成不同的页面逻辑。

以下以“发现音乐-推荐”模块为例来说明：

这个模块包含一系列列表：

- 同步** 推荐的Banner列表
- 同步** 推荐的歌单列表
- 同步** 推荐的榜单列表
- 异步** 推荐的节目列表
- 异步** 推荐的新碟列表
- 异步** 推荐的单曲列表



针对此模块定义内容规范：

- 发现音乐 - 推荐
 - 数据类型
 - 模板资源
 - 异步接口
 - 页面摘要

Bookmarks

- 数据模型
- 模板资源
- 路由
- 异步接口
- 开发规范

推荐

模板文件: /template/discover/rec.ftl

模板描述: 发现音乐模块之推荐页面默认模板, 用于展示各种推荐列表

填充数据:

名称	类型	描述
recBanners	Array <Banner>	推荐的Banner列表
recPlayLists	Array <PlayList>	推荐的歌单列表
recTopLists	Array <PlayList>	推荐的榜单列表

异步接口

获取推荐的节目列表

请求方式: GET

请求地址: /api/getRecProgramList

输入参数:

名称	类型	描述
offset	Number	请求数据偏移量
limit	Number	请求列表数量

输出结果:

这些模板规范，需有前后端人员共同确认后，才能进行后续开发。

工程构建

项目结构

- 后端模板
- 前端实现

后端模板

template

discover

rec

macro.ftl

mock.ftl

rec.ftl

error

disconnect.ftl

macro.ftl

后端模板

webapp

res

src

css

javascript

cache

mock

album.js

playlist.js

program.js

track.js

user.js

lib

page

discover

rec.js

widget

pages

本笔记由西风潇潇编写，欢迎浏览博客访问更多内容：<http://www.xifengxx.com>

初始代码：（使用工具构建）

-rec.ftl

```
<@compress>
<!DOCTYPE html>
<html>
  <#include "../mock.ftl"/>
  <#include "../../macro.ftl"/>
  <#include "../macro.ftl"/>
  <head>
    <title>发现音乐 - 推荐</title>
    <meta charset="utf-8"/>
    <@css/>
    <!-- @STYLE -->
    <link href="${csp}page/discover/rec.css" rel="stylesheet" type="text/css">
  </head>
  <body>

    <!-- Page Content Here -->

    <!-- @SCRIPT -->
    <script src="${jslib}define.js?${jscnf}"></script>
    <script src="${jspro}page/discover/rec.js"></script>
  </body>
</html>
</@compress>
```

模拟数据

推荐

模板文件: /template/discover/rec/rec.xml

模板描述: 发现音乐模块之推荐页面默认模板, 用于展示

预填数据:

名称	类型	描述
recBanners	Array <Banner>	推荐的Banner列表
recPlayLists	Array <PlayList>	推荐的歌单列表
recTrackLists	Array <Track>	推荐的单曲列表

template

discover

rec

macro.xml

mock.xml

rec.xml

```
<!-- 推荐的Banner列表测试数据 -->
<!-- 推荐的歌单列表测试数据 -->
<!-- 推荐的单曲列表测试数据 -->
<#assign recTrackLists = [
{
  "id": 400,
  "name": "歌曲名称1",
  "duration": 254878948,
  "album": {
    "id": 1,
    "name": "专辑1",
    "playCount": 100,
    "coverImgUrl": "/test/1.png"
  },
  "artists": [
    { "id": 120, "name": "歌手2" },
    { "id": 130, "name": "歌手3" }
  ]
},
]
]/>
```

请求方式: GET
请求地址: /api/getRecProgramList
输入参数:

名称	类型	描述
offset	Number	请求数据偏移量
limit	Number	请求列表数量

输出结果:

名称	类型	描述
code	Number	结果标识
result	Array <Program>	节目列表

webapp

res

src

css

javascript

cache

mock

rec

playlist.json

program.json

track.json

album.js

playlist.js

program.js

track.js

user.js

lib

page

widget

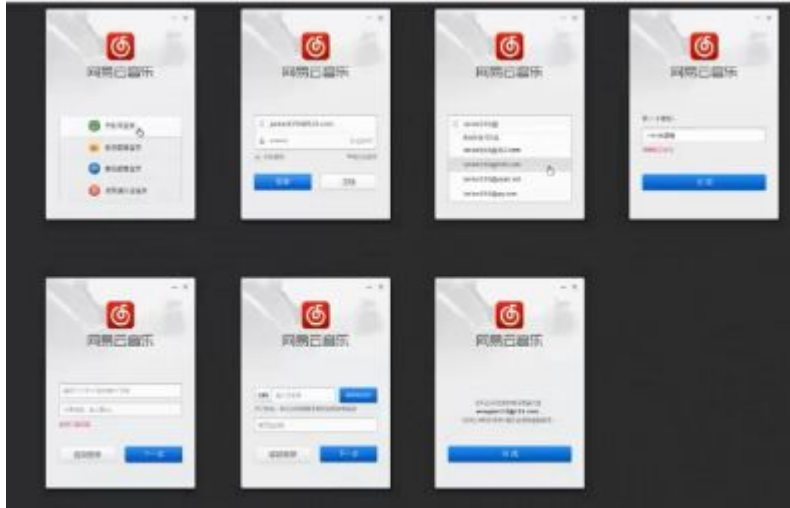
```
{
  "code": 200,
  "result": [
    {
      "id": 1,
      "name": "节目1",
      "playCount": 100,
      "coverImgUrl": "/test/1.png",
      "artist": {
        "id": 100,
        "name": "歌手1"
      }
    }, {
      "id": 2,
      "name": "节目2",
      "playCount": 120,
      "coverImgUrl": "/test/2.png",
      "artist": {
        "id": 100,
        "name": "歌手1"
      }
    }
  ]
}
```

系统实现

视觉说明：（以“网易云音乐”为例）

视觉稿定义了交互稿中的所有效果，如下图包含各个情况下用户界面的显示样式：

本笔记由西风潇潇编写，欢迎浏览博客访问更多内容：<http://www.xifengxx.com>



组件提取

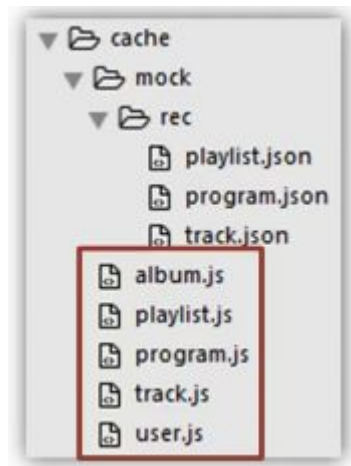
视觉稿之后则需要从中提取出通用组件，其中包括：

- 通用元件（Logo，提示，输入框，图标，按钮等）
- 通用列表
- 复合组件（比如评论控件）
- 浮层弹窗

下图为按钮原件：

逻辑实现

- 数据层实现：节目数据（program.js）
- 页面模板实现：页面模板（rec.ftl）
- 控制层实现：页面入口（rec.js）



测试发布

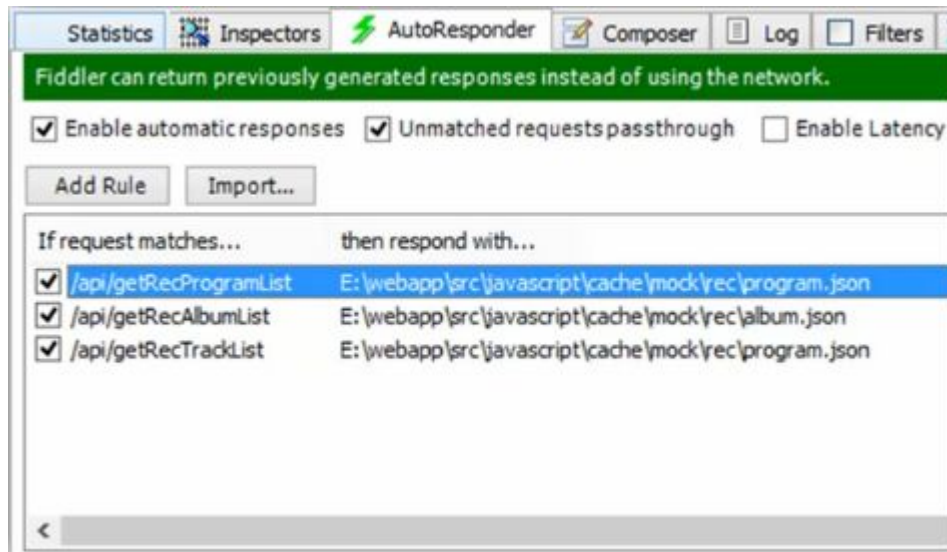
本地测试

配置模拟数据

1.使用同步模拟数据

```
<@compress>
<!DOCTYPE html>
<html>
  <#include "../mock.ftl"/>
  <#include "../../macro.ftl"/>
  <#include "../macro.ftl"/>
  <head>
    *****
```

2.使用异步模拟数据：采用第三方代理软件：

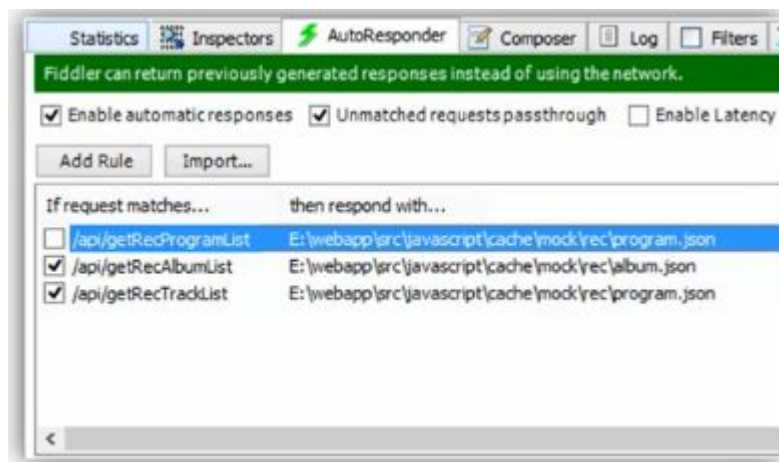


对接联调

- 去除同步模拟数据

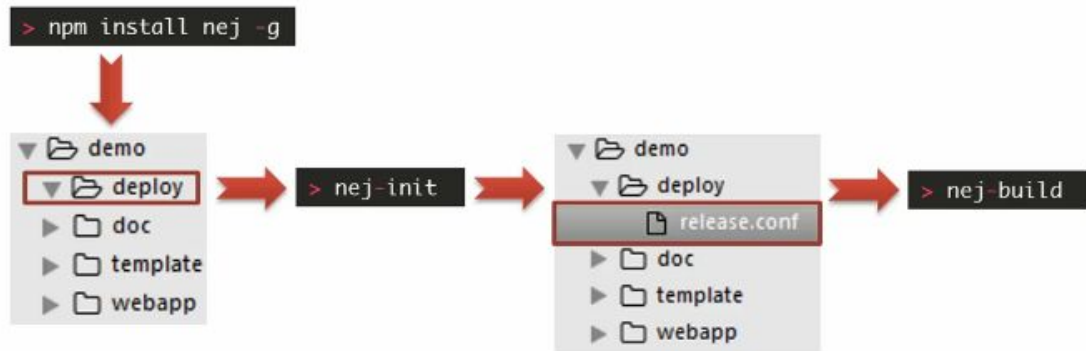
```
<@compress>
<!DOCTYPE html>
<html>
  <#include "../mock.ftl"/>
  <#include "../../macro.ftl"/>
  <#include "../macro.ftl"/>
  <head>
    .....
```

- 去除异步数据:



发布上线

打包发布：选用 nej 构建工具



打包配置:

- 输入输出

```
# 路径相关配置 # WEB根路径，必须配置，
# 如果是相对路径则相对于当前配置文件路径(即.conf文件所在目录)
DIR_WEBROOT = ../webapp/
# 项目服务器端模板文件根路径
# 服务器端模板文件确保页面所需的CSS、JS文件的引用均出现在模板文件中
DIR_SOURCE_TP = ../template/
# 项目服务器端模板输出路径，默认为DIR_OUTPUT配置信息
DIR_OUTPUT_TP = ../views/
```

优化配置:

- 优化图片

```
# 优化图片输出
# 图片优化开关，打开此开关则DIR_STATIC下的图片会做优化压缩，替换原文件
OPT_IMAGE_FLAG = true
# 图片输出质量 1-100
OPT_IMAGE_QUALITY = 100
```

- CDN 配置
- 代码压缩
- 代码合并

讨论区问答:

前端架构师在一个项目中具体是做什么的？

1. 参与需求评审、交互评审、视觉评审，给出专业意见和建议（如：技术可行性、难点、方案等）。
2. 结合项目计划，根据技术实现要求（如兼容性要求、平台要求等），评估人力和时间，预估风险。
3. 指导或负责制定开发规范（如：结构、编码、文档等）。
4. 指导或负责制定流程规范（如：协作、任务、接口、测试、联调、发布等）。
5. 协助实施开发规范和流程规范，并验证规范执行。

6. 为项目进行技术选型（包括框架、库、工具等），并负责培训、指导和问题解决。

7. 指导或负责系统设计（包括系统入口设计和异步接口设计等）。

8. 指导或负责项目构建和环境搭建。

9. 指导或负责发布部署和优化

10. 帮助技术升级，推广新技术新方案新工具。