# 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

math.js

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

1. 没有依赖声明
2. 使用全局状态

Anti-P

caculator.js

```
var action = "add";

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

)bject L

math.js

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

caculator.js

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

caculator-1.js

```
var caculator = (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

caculator-2.js

```
var caculator = (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**）:命运空间可以解决全局变量的污染的问题。

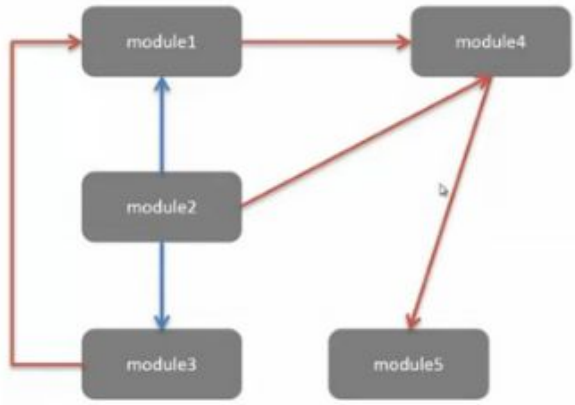
math.js

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

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

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

caculator.js

```
namespace("caculator", ["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

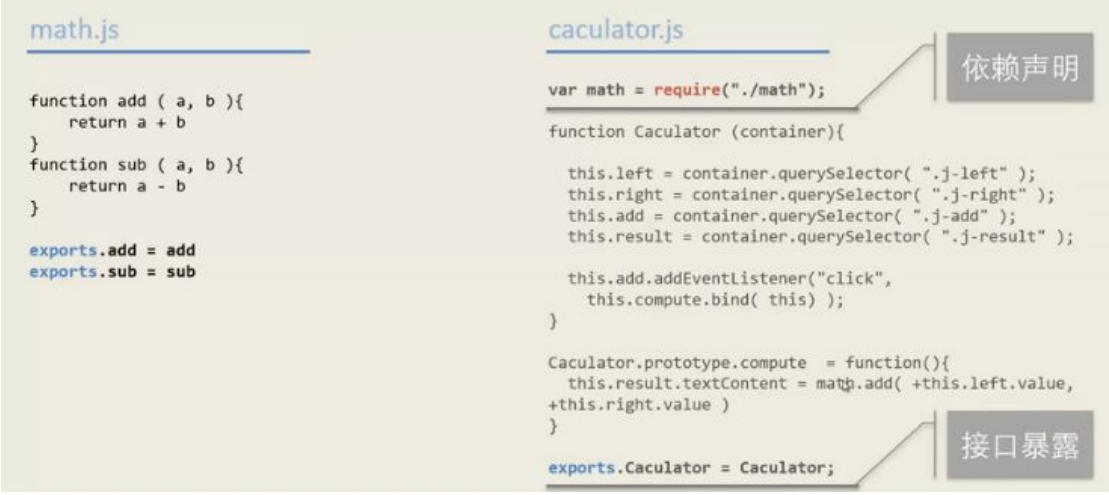
**职责：**

- 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**

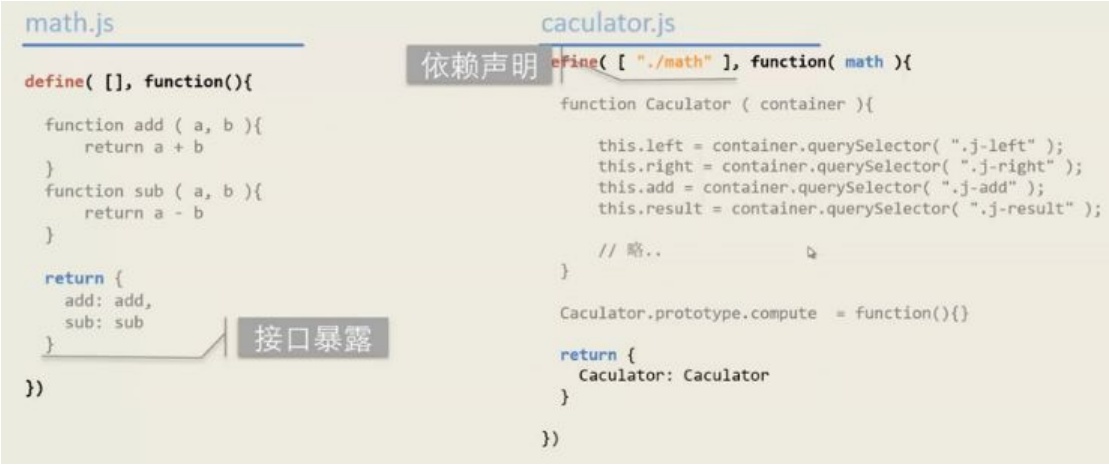
- 依赖管理成熟可靠
- 社区活跃，规范接受度高
- 运行时支持，模块定义非常简单
- 文件级别的模块作用域隔离
- 可以处理循环依赖

**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

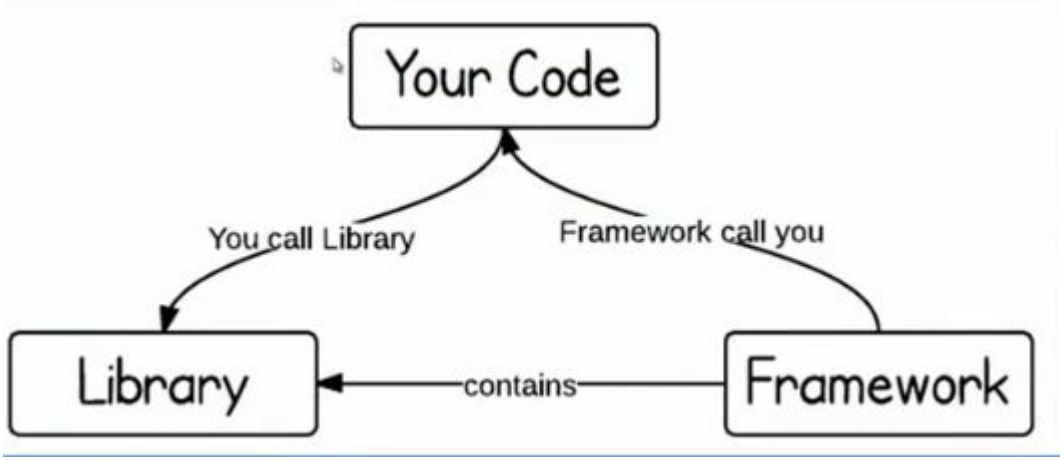
# 框架（基于 JavaScript 的框架）

## 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：

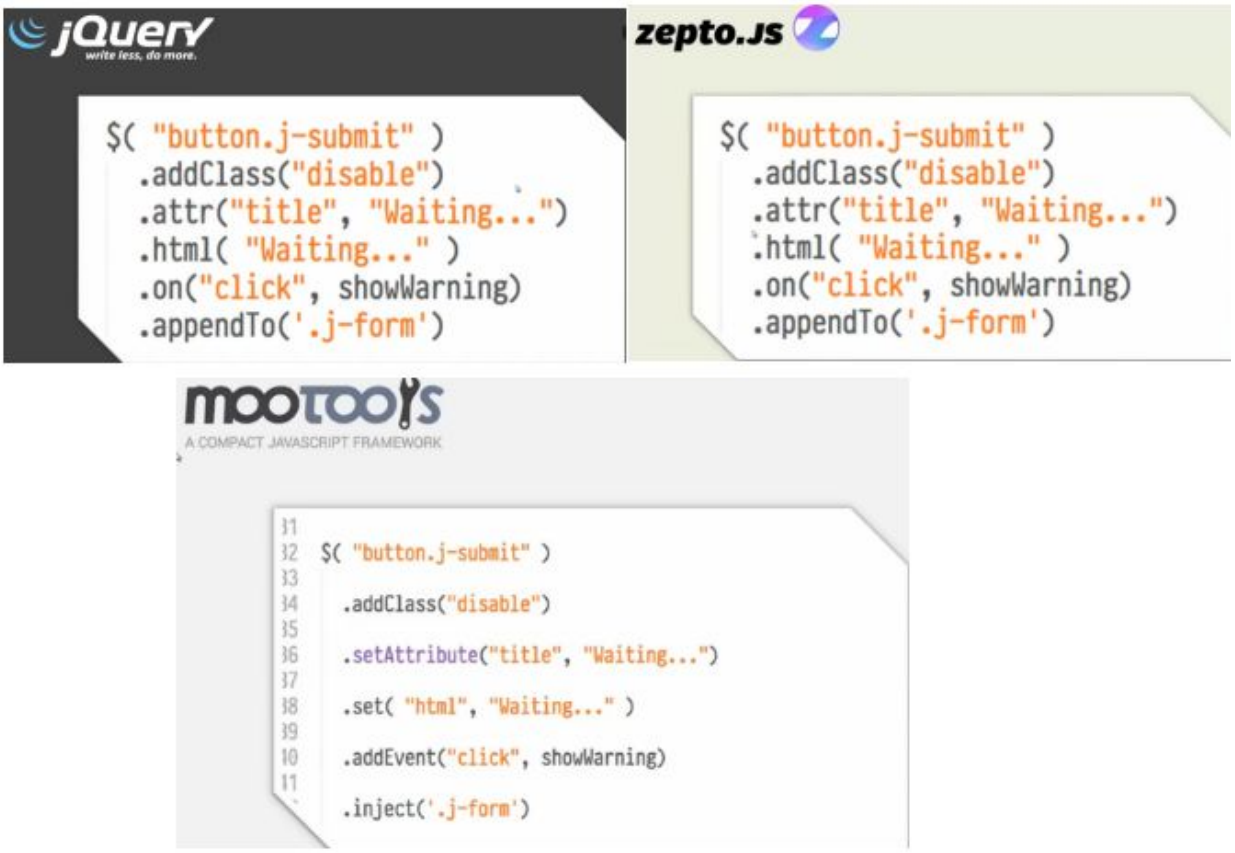
- 提供便利的 DOM 查询、操作、移动等操作
- 提供事件绑定及事件代理等支持
- 提供浏览器特性检测、 UserAgent 侦测
- 提供节点属性、样式、类名的操作
- 所有以上操作实现目标平台的跨浏览器支持

#### 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。





- 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**

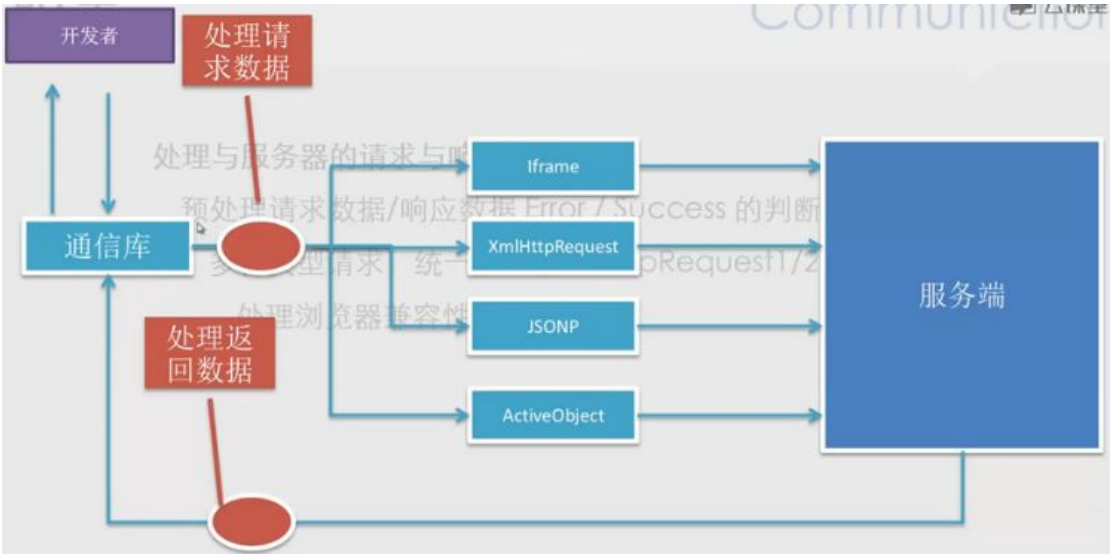

选取标准：Issue 总量与解决率比 Star 更关键

## 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：**

**Reqwest:**

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

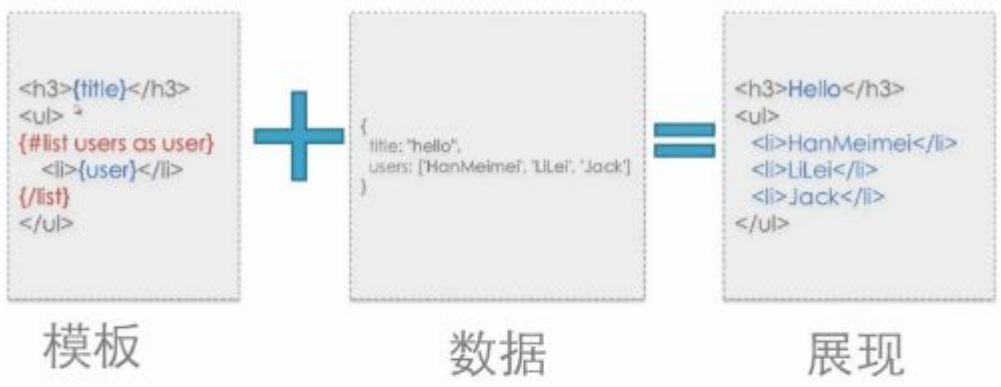
- 异步列队/流程控制等等

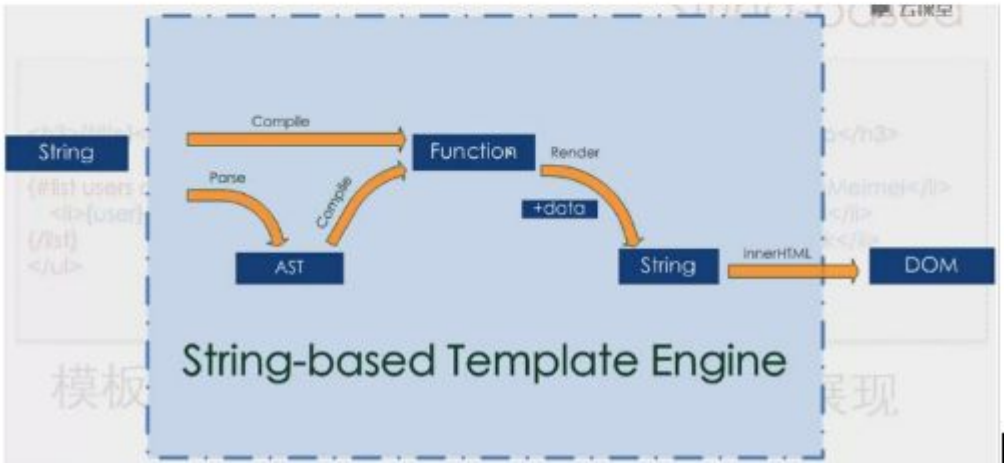
**Utility 库：**


## Templating:模板技术

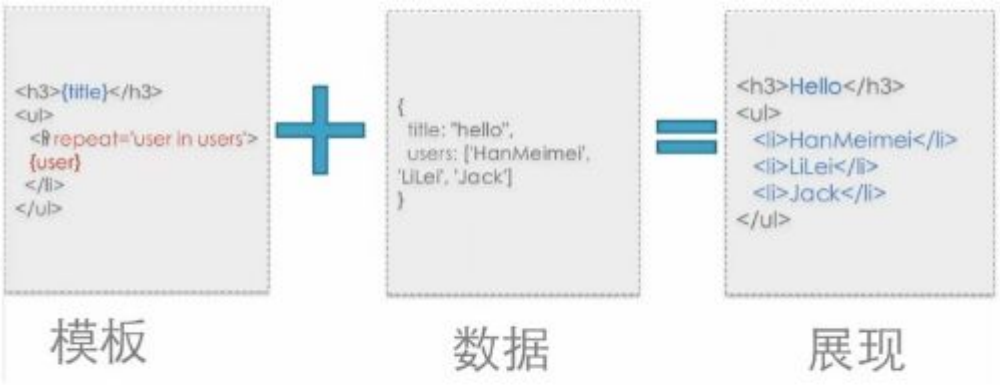
与其相关的有 *String-based*、*DOM-based*、*Living Template*。

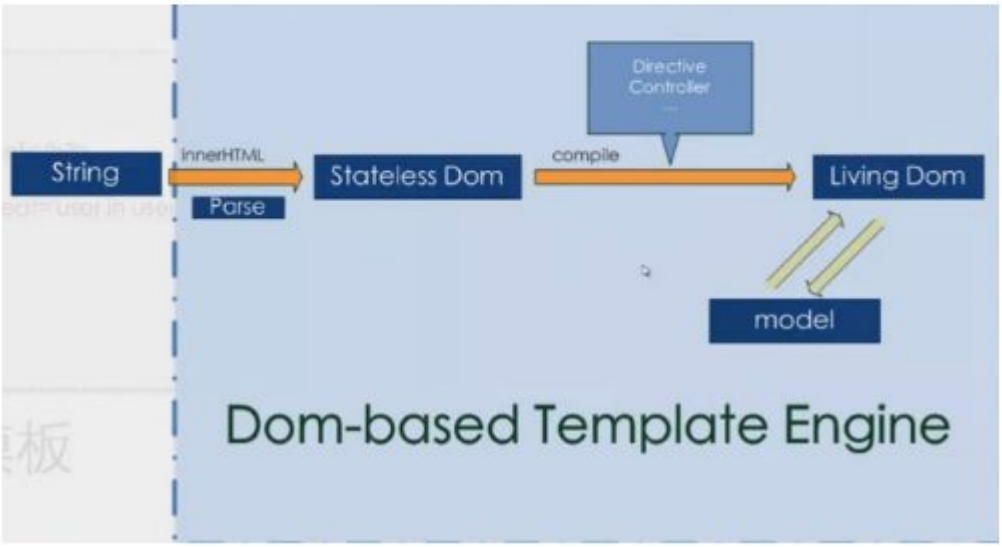
### String-based:基于字符串的模板



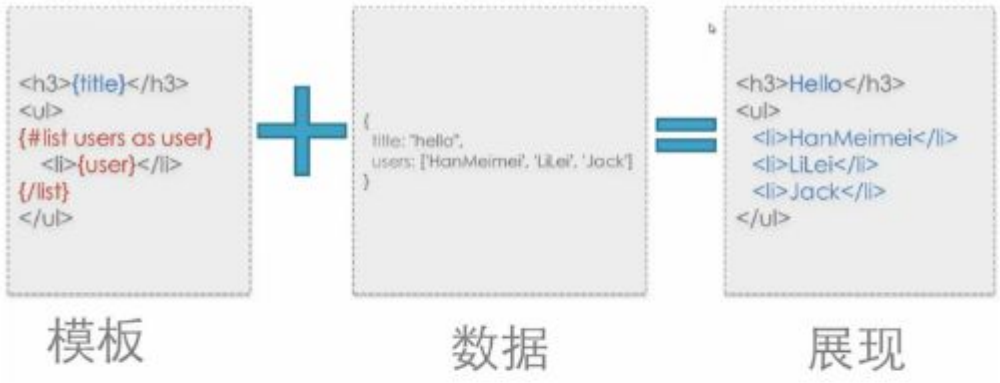



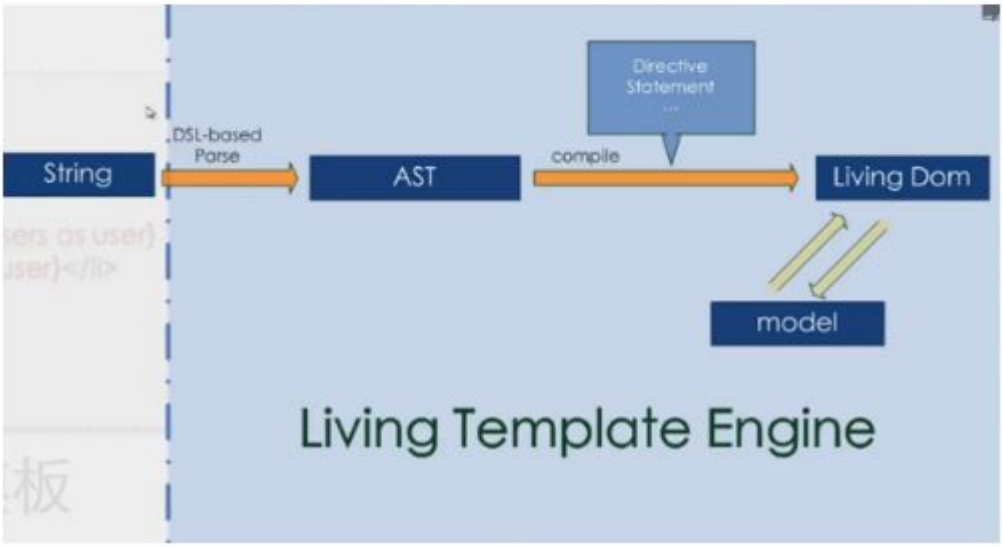

**DOM-based: 基于 DOM 的模板**

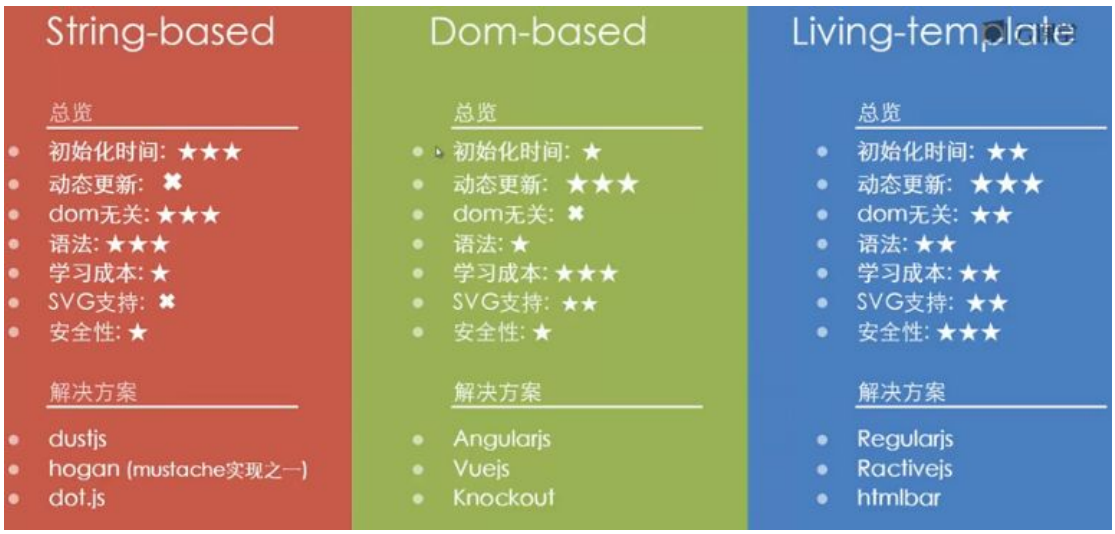



**Living-Template:活动模板技术**



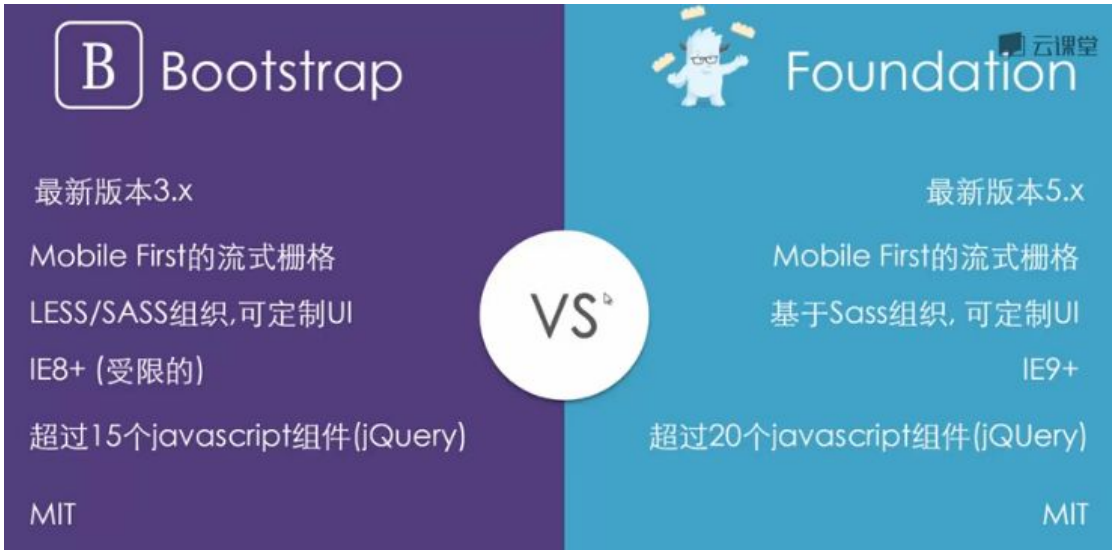

## Component:组件

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**

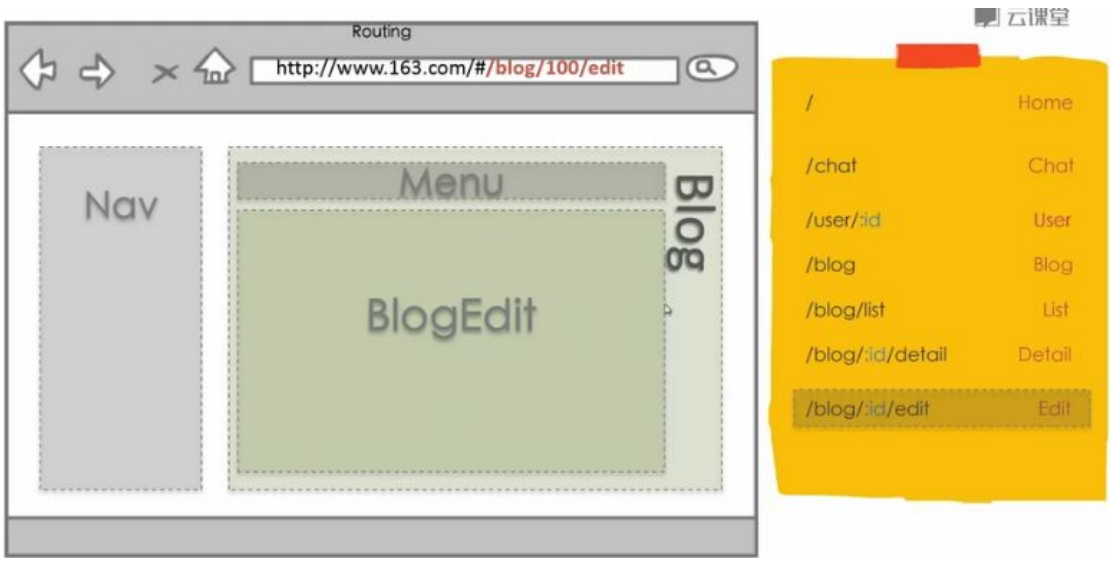
- Knockout-Bootstrap
- AngularUI Bootstrap
- React Bootstrap

## 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：**

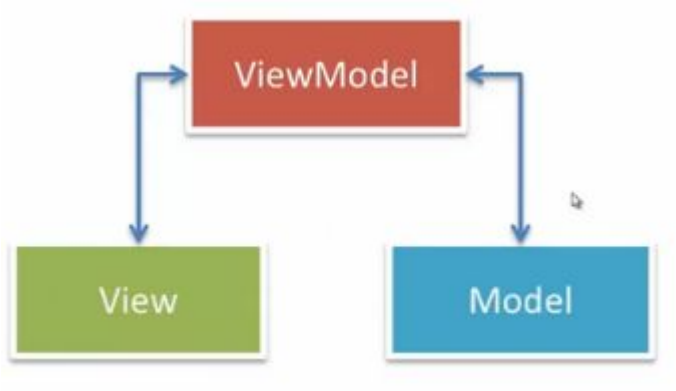

## Architecture（解耦）：架构

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/

## 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

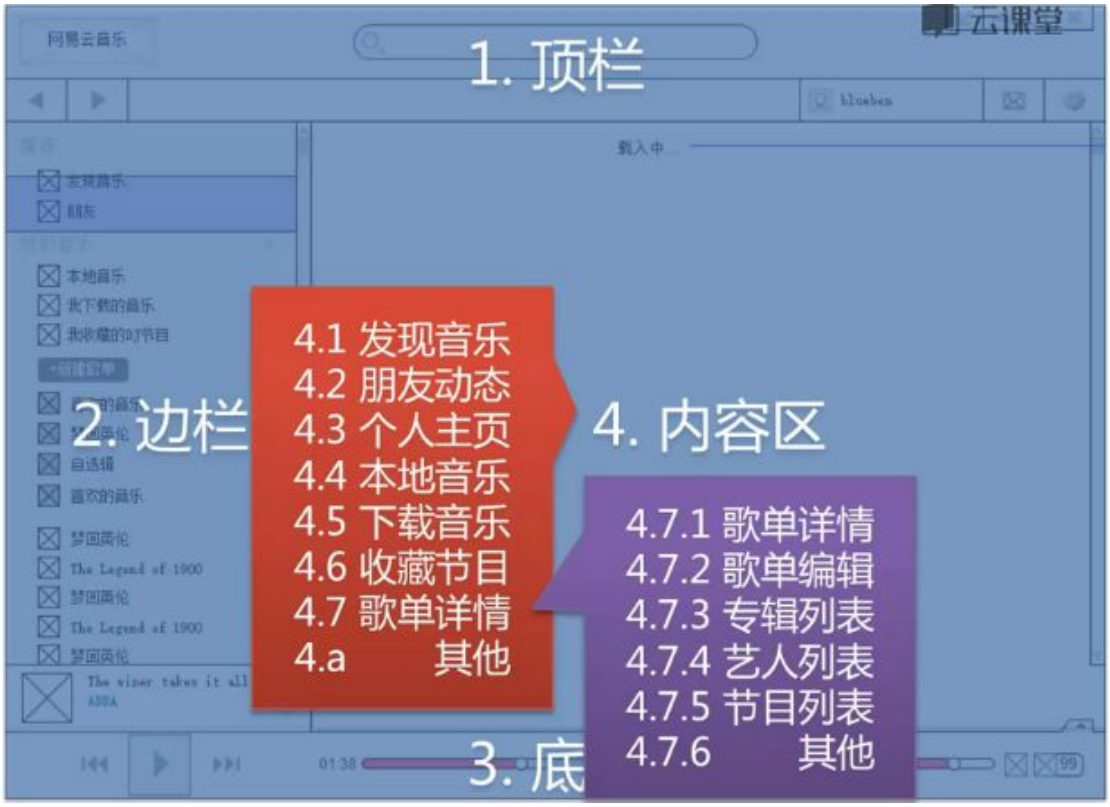
综合运用课程案例---**网易云音乐**， 并且主要关注前端工程师的工作职责，不包含其他工程师的职责规范。

### 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
    - ……



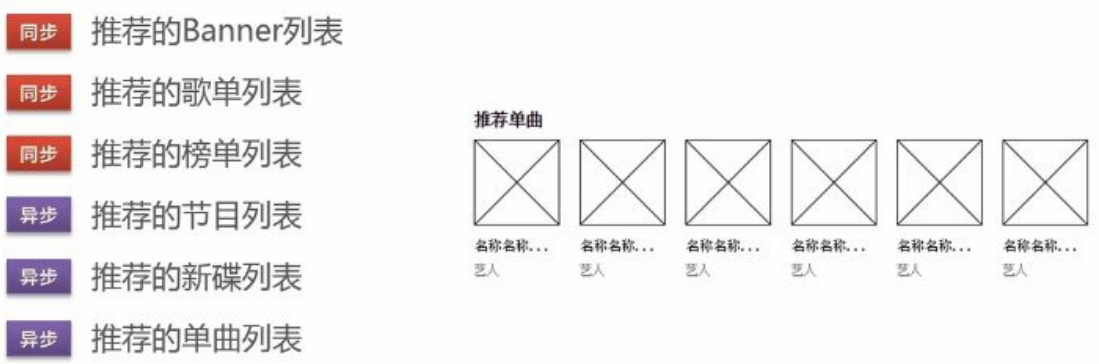

系统分解必须对照交互稿做到百分之百的对应，不能漏掉任何一个模块。 后续的开发与评估都需根据此分解进行。

## 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：**



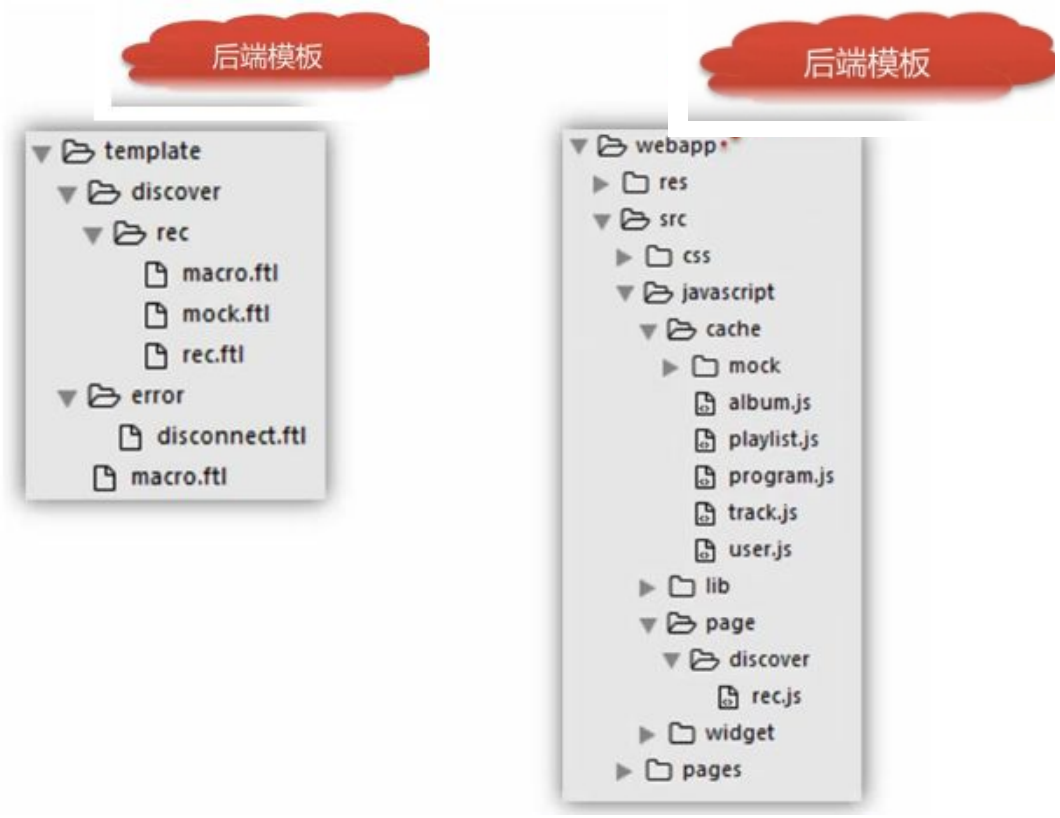
**针对此模块定义内容规范：**


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

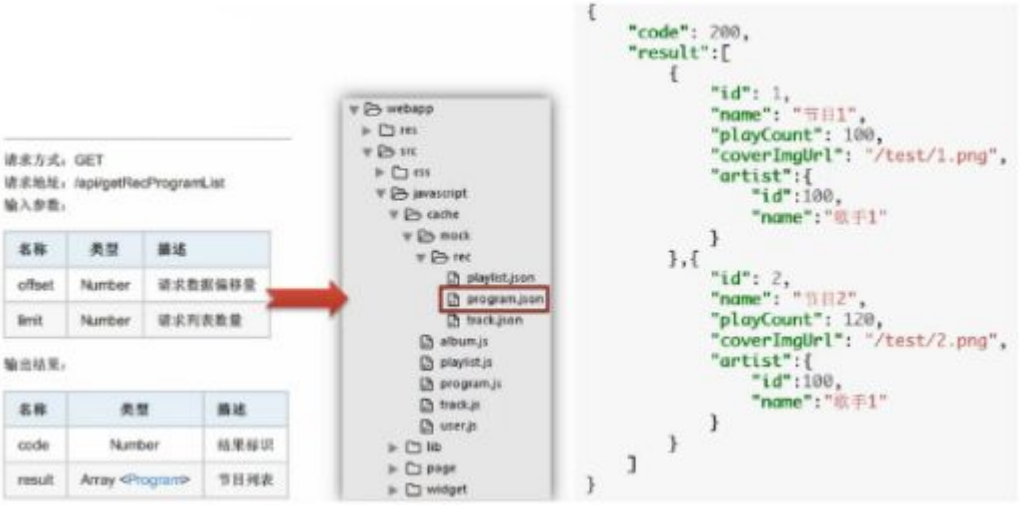


**初始代码：（使用工具构建）**

-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="${cspro}page/discover/rec.css" rel="stylesheet" type="text/css">
  </head>
  <body>

    <#-- Paget Content Here -->

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

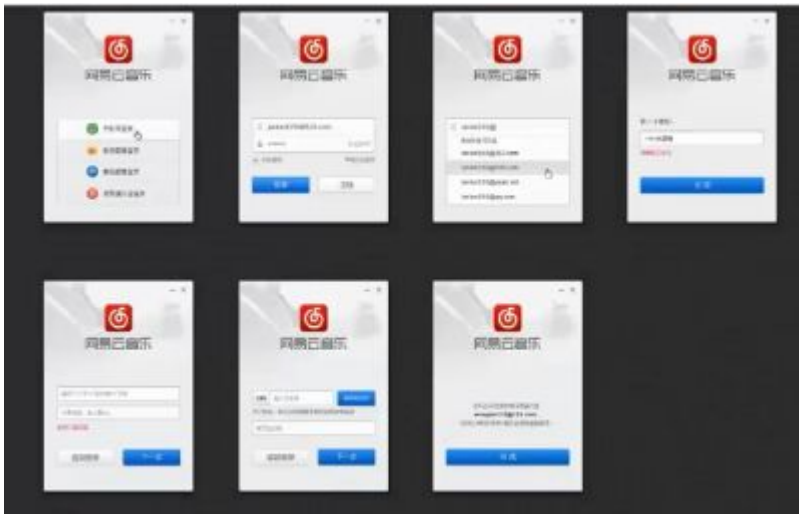
**模拟数据**


## 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：



## 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：


## 页面制作：

以“网易云音乐-发现音乐-推荐”模板为例,其包括：

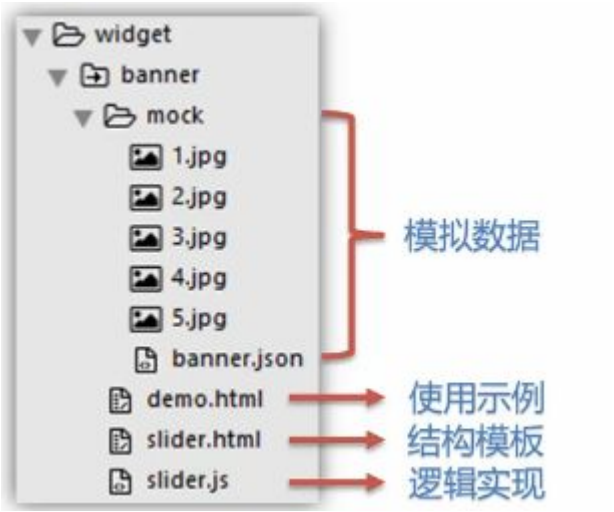
- Banner
- 推荐歌单
- 最新专辑等

根据视觉效果来设计静态页面。

## 组件封装

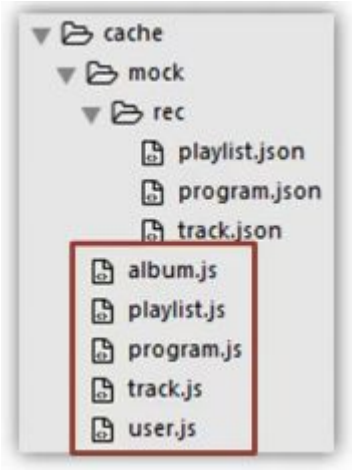
BannerSlider

- 模板（slider.html）
- 示例（demo.html）



### 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

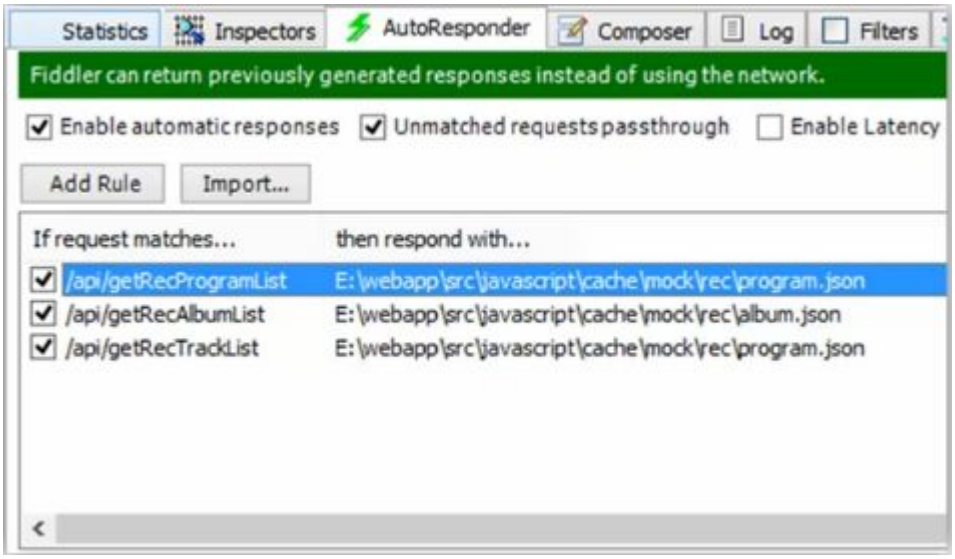
### 本地测试

#### 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：

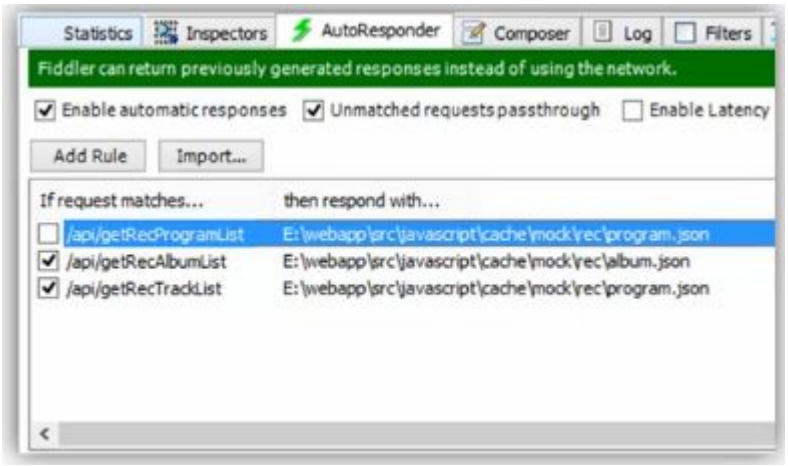



## 对接联调

- 去除同步模拟数据

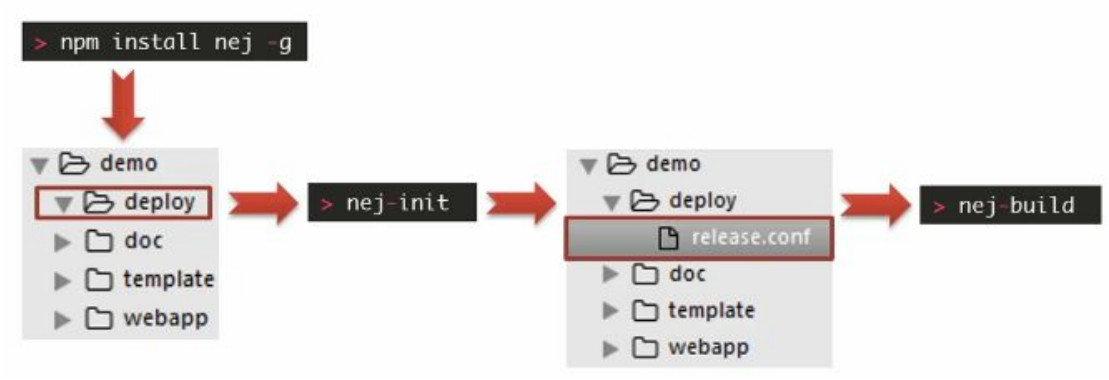
```
<@compress>
<!DOCTYPE html>
<html>
  <#include "./mock.ftl"/>
  <#include "../../macro.ftl"/>
  <#include "./macro.ftl"/>
  <head>
  ......
```

- 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。