



金蝶KIS轻应用前端开发培训

——基于**grunt**的模块化和片段化的轻应用开发



micty_deng (邓良太)

KIS云平台部

2016.01.15

关于我

micty_deng (邓良太)

KIS云产品部

主导开发过大型JS
框架

Web前端开发工
程师(JavaScript)

技术

曾于腾讯公司QQ秀产
品部工作过两年

熟悉 C#.NET 开发

5 年的 JS
开发经验

熟悉HTML

K歌

户外

探
险

单身屌丝

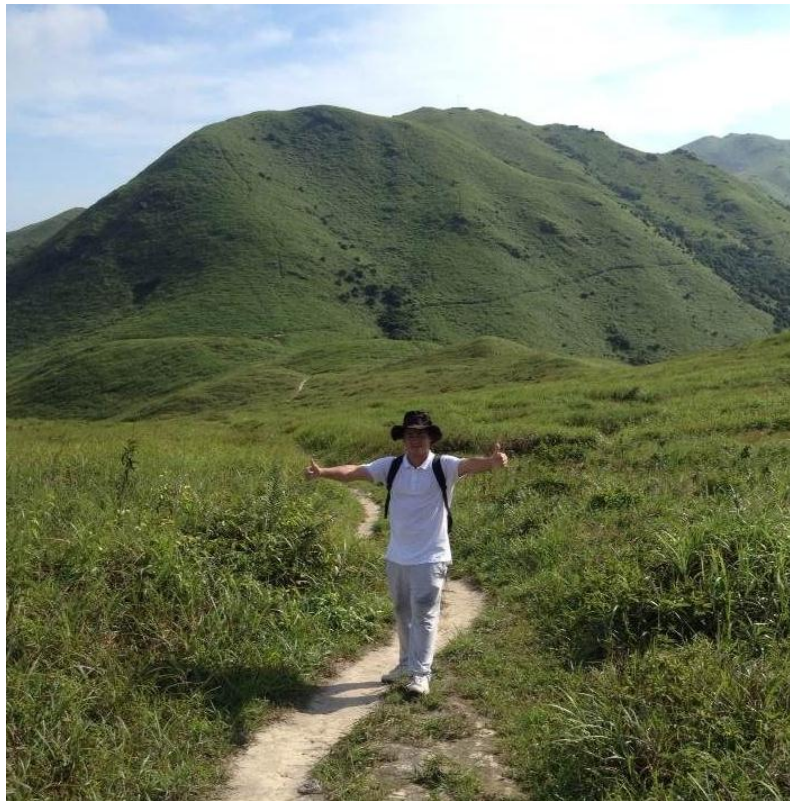
熟悉CSS

骑
行

80后

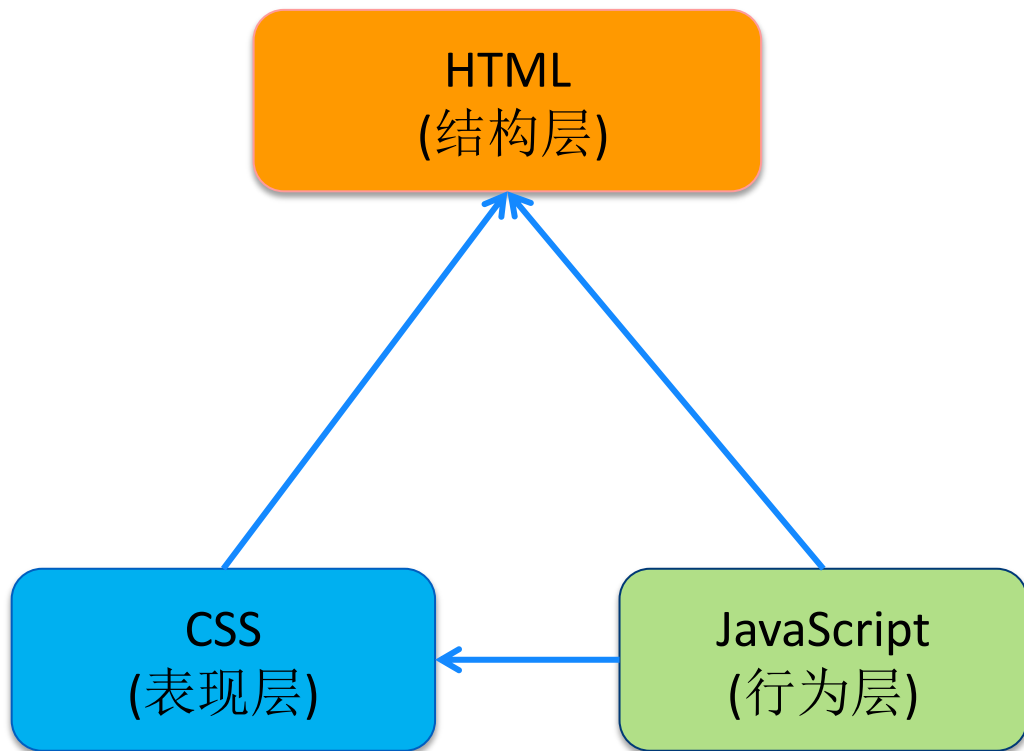
摄影

广东人



- ☁ 轻应用的开发流程和模式特点
- ☁ 基于 grunt 的实时编译型的片段化开发
- ☁ 模块的组织、依赖和管理
- ☁ 事件驱动的通信模式
- ☁ 树形结构的中介者模式
- ☁ 本地代理模拟服务器响应
- ☁ Web 前端的模板填充
- ☁ 把HTML模板从代码中剥离出来
- ☁ 基于grunt的自动化构建和发布



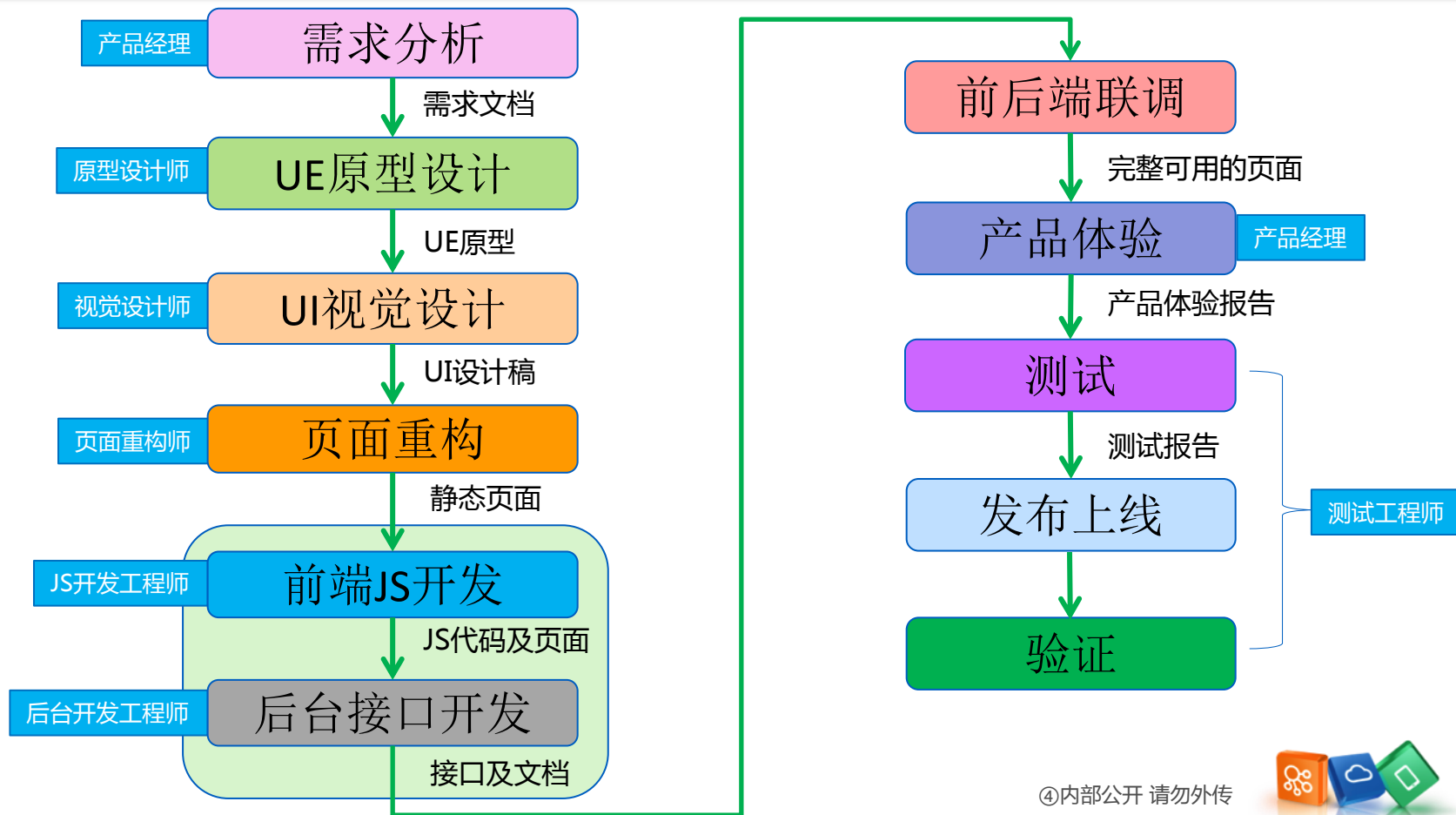


三层结构





开发流程和分工模式



☁ 页面重构师

- 对视觉设计图进行分析，用 PhotoShop 等工具进行切图，用 HTML+CSS 写出静态页面，精确还原设计图。

☁ JS开发工程师

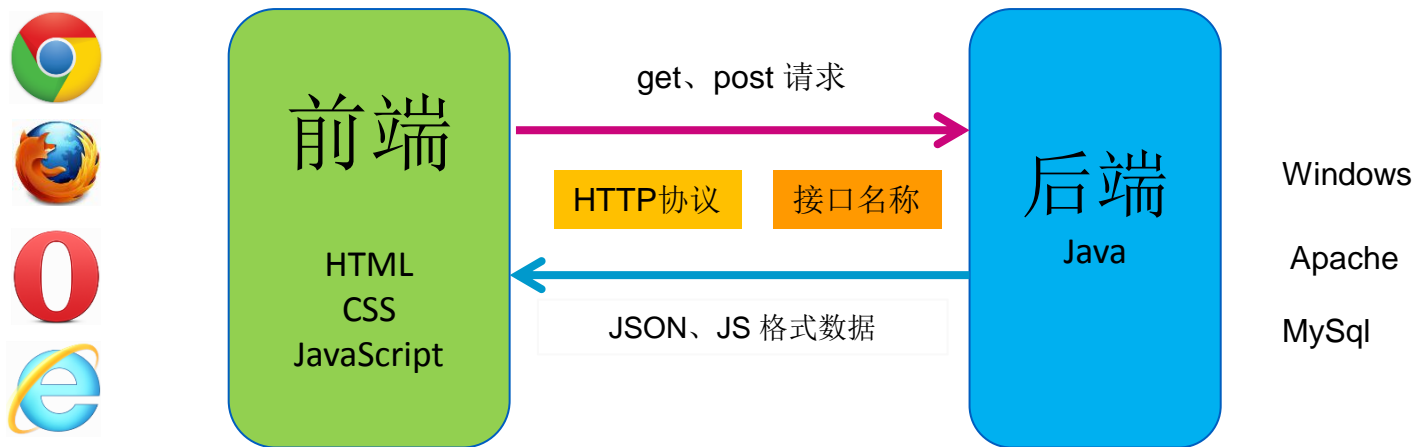
- 用 JavaScript 进行对重构页面进行编程，响应用户交互，与服务器后台接口进行通讯，对页面进行数据填充与展现等。

Web前端开发工程师



前后端完全分离的开发模式

把后台数据与逻辑和前端展示完全分离开来，
两者通过接口和数据进行沟通



专注于 UI 层面、体验层面的开发不必了解后台的技术架构，包括使用的后台使用的语言、数据库、服务器逻辑结构等

专注于数据、业务逻辑的开发不必了解前端的各种复杂知识，包括UI设计、HTML、CSS、JS、浏览器兼容性、性能优化等



☁ **核心：解耦！让更合适的人做更合适的事**

☁ **前端变得越来越复杂、越来越重要，需要专业的人员去更专注地处理**

☁ **一个 Web 团队期望的工作方式**

① 让前端专注于 UI 层面、体验层面的开发

不必了解后台的技术架构，包括使用的后台使用的语言、数据库、服务器逻辑结构等

② 让后端专注于数据、业务逻辑的开发

不必了解前端的各种复杂知识，包括UI设计、HTML、CSS、JS、浏览器兼容性、性能优化等

③ 第1点和第2点能并行开发、独立开发

④ 将第1点和第2点的工作无缝衔接起来进行联调



- ☞ HTML5、CSS3
- ☞ W3C 标准化，浏览器兼容性
- ☞ JavaScript 严格模式
- ☞ 手机终端、移动终端的 APP



One Web **W3C** for All





KIS轻应用，是指基于KIS平台的Web App，是用移动端Web方式实现Native App功能的技术，实质仍为Web开发，但有自己的特点和优点。

Web App的功能和UI展现上接近于Native App，但技术上是Web，是Web和App的综合体。

小应用大舞台



跨平台

使用 W3C 标准的 **HTML** 语言开发，能够轻松实现跨平台，移动应用开发者不再需要考虑复杂的底层适配和跨平台开发语言的问题。

开发和学习成本低

基于当下开始普及流行的 HTML5 和 CSS3 技术，Web App 在投入上会大大的低于传统的 Native App。它可以实现很多原本 Native App 才可以实现的功能，比如 LBS 的功能、本地数据存储、音视频播放的功能，甚至还有调用照相机和结合 GPU 的硬件加速功能。

开发迭代周期短，升级简单

Native App 的迭代周期比较长，用户需要频繁的重新下载与升级。而 Web App 则更新迭代周期短得多，也无需用户下载即可实时升级。



- ☁ 屏幕小，可展现和交互的空间小
- ☁ UI元素种类相对统一，数量有限
- ☁ 视图足够简单，布局比较统一
- ☁ 视图数量较多，视图间的跳转比较常见
- ☁ 视图间的跳转要响应快和动画过渡效果
- ☁ 滚动体验要尽可能接近原生App
- ☁ 响应要迅速，提示要友好
- ☁ 要支持各种手势

.....



优点

- ✓ 只有一个物理页面，有多个逻辑视图
- ✓ 一次加载，到处运行
- ✓ 有效避免页面跳转时的重新加载资源文件引起的性能问题
- ✓ 更方便统一管理视图的呈现、隐藏以及过渡动画效果
- ✓ 更接近原生APP的体验

缺点

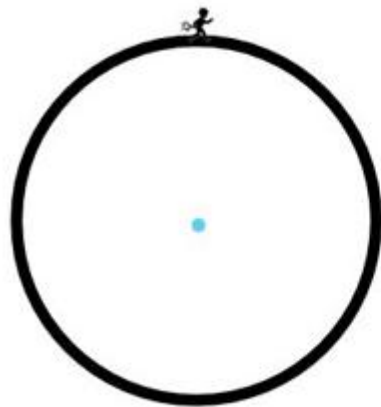
- 所有的逻辑视图都写在一个物理页面，导致 HTML 代码量暴涨
- 页面难以维护，在海量 HTML 代码中进行修改非常痛苦
- 文件容易冲突，不利于团队按功能模块进行分工协作
- 功能模块多，JS代码量大，模块的划分和依赖难以管理
- 不利于自动化工具的预处理，如生成 id，引入 css 和 js



核心：专注

让团队成员更专注的完成自己的功能模块，免受其它代码、文件和习惯的干扰

通过把功能模块相关的资源、代码、文件放到模块专有目录，可以更专注自己该关心的



世界很大
我们只专注这一点





GRUNT

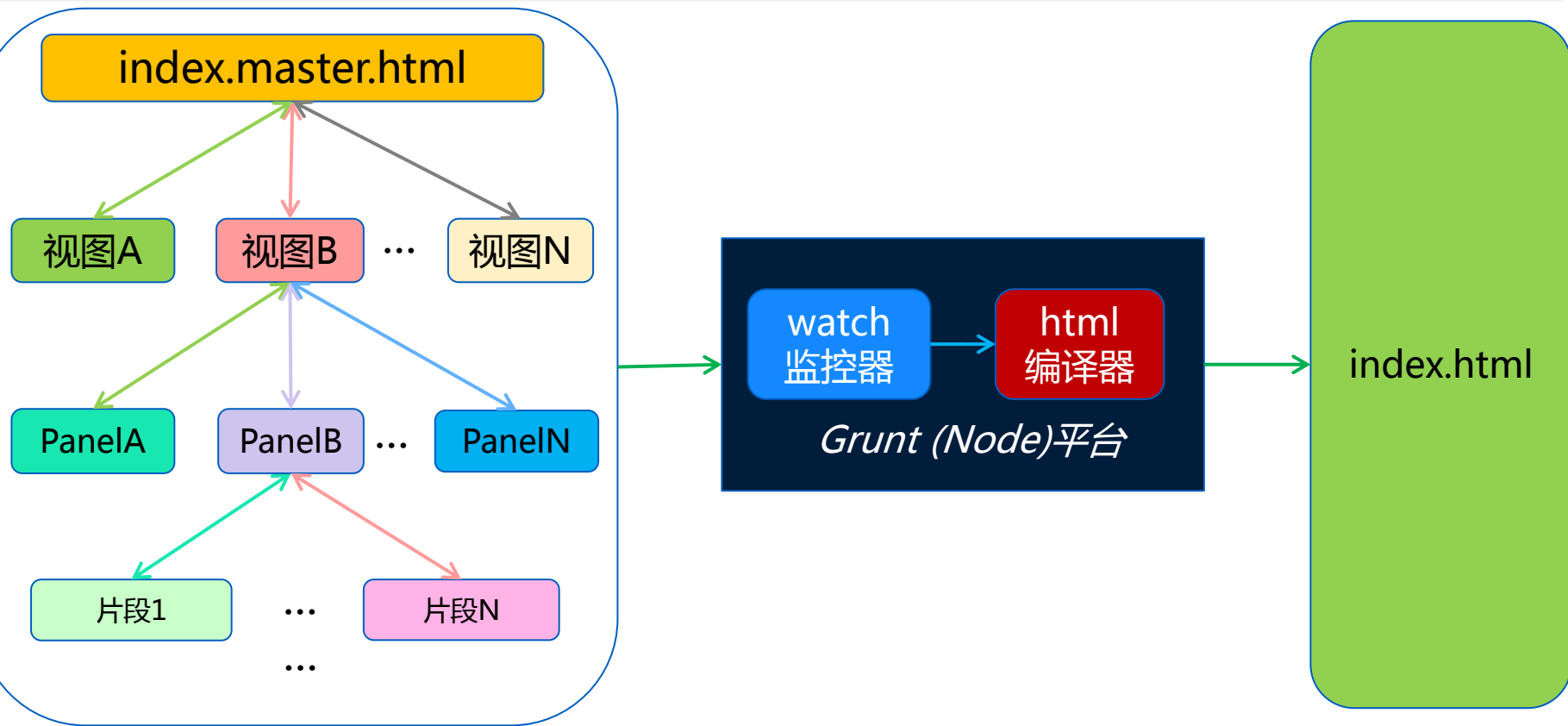
JavaScript世界的构建工具

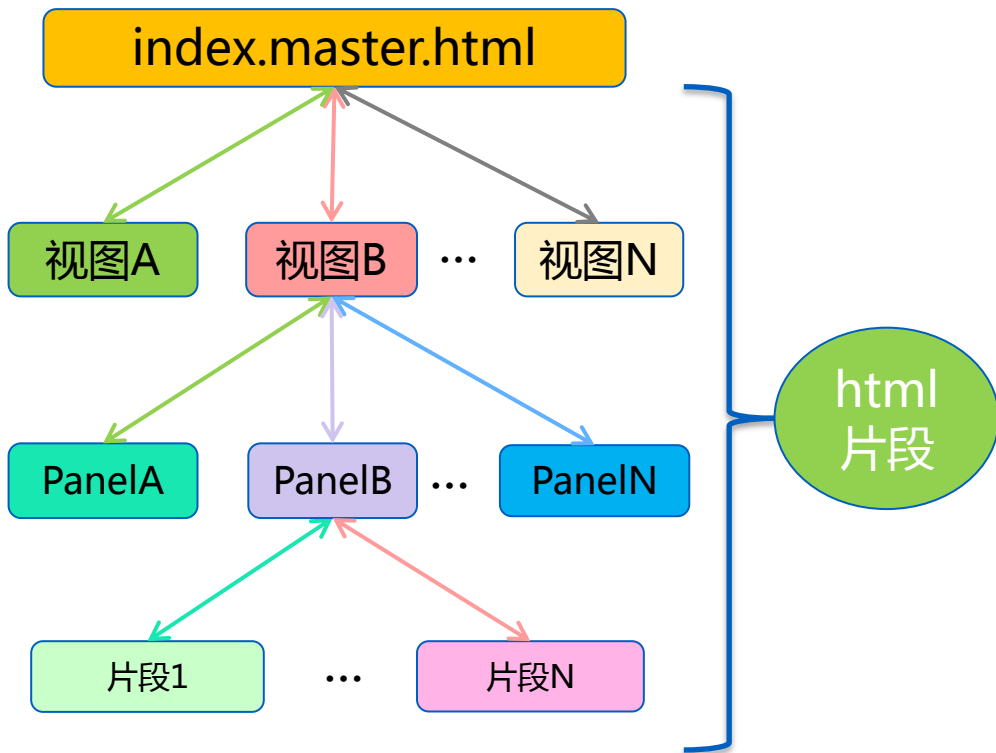
<http://www.gruntjs.net/>

为何要用构建工具？

一句话：**自动化**。对于需要反复重复的任务，例如合并、压缩（minification）、编译、单元测试、linting等，自动化工具可以减轻你的劳动，简化你的工作。当你正确配置好了任务，任务运行器就会自动帮你或你的小组完成大部分无聊的工作。







- 支持递归引用，html片段里可以继续引用其它子片段
- 保持原有的缩进格式
- 支持自动插入模块名相关的特征
- 实时编译，保存、刷新即生效
- 支持短名称引入子片段，有利于自适应后期文件名和目录的变更





Less 将 CSS 赋予了动态语言的特性，如嵌套、变量、继承、运算、函数。Less 既可以在客户端上运行 (支持IE 6+, Webkit, Firefox)，也可以借助 NodeJs或者 Rhino 在服务端运行。

作为一门标记性语言，CSS 的语法相对简单，对使用者的要求较低，但同时也带来一些问题：CSS 需要书写大量看似没有逻辑的代码，不方便维护及扩展，不利于复用，尤其对于非前端开发工程师来讲，往往会因为缺少 CSS 编写经验而很难写出组织良好且易于维护的 CSS 代码，造成这些困难的很大原因源于 CSS 是一门非程序式语言，没有变量、函数、作用域等概念。Less 为 Web 开发者带来了福音，它在 CSS 的语法基础之上，引入了变量，Mixin（混入），运算以及函数等功能，大大简化了 CSS 的编写，并且降低了 CSS 的维护成本，就像它的名称所说的那样，Less 可以让我们用更少的代码做更多的事情。

本质上，Less 包含一套自定义的语法及一个解析器，用户根据这些语法定义自己的样式规则，这些规则最终会通过解析器，编译生成对应的 CSS 文件。Less 并没有裁剪 CSS 原有的特性，更不是用来取代 CSS 的，而是在现有 CSS 语法的基础上，为 CSS 加入程序式语言的特性

<http://lesscss.org/>

<http://www.lesscss.net/>

<http://www.bootcss.com/p/lesscss/>





结构化（嵌套）

```
//less
#header {
  h1 {
    font-size: 26px;
    font-weight: bold;
  }
  p {
    font-size: 12px;
    a {
      text-decoration: none;
      &:hover {
        border-width: 1px;
      }
    }
  }
}

/*生成的 CSS */
#header h1 {
  font-size: 26px;
  font-weight: bold;
}

#header p {
  font-size: 12px;
}

#header p a {
  text-decoration: none;
}

#header p a:hover {
  border-width: 1px;
}
```

我们可以在一个选择器中嵌套另一个选择器来实现继承，这样很大程度减少了代码量，并且代码看起来更加的清晰。



高级的CSS写法：Less




变量

```
//less
@color: #4D926F;

#header {
  color: @color;
}

h2 {
  color: @color;
}
```



```
/* 生成的 CSS */

#header {
  color: #4D926F;
}

h2 {
  color: #4D926F;
}
```

变量允许我们单独定义一系列通用的样式，然后在需要的时候去调用。所以在做全局样式调整的时候我们可能只需要修改几行代码就可以了。



混入



```
//less

.rounded-corners (@radius: 5px) {
  border-radius: @radius;
  -webkit-border-radius: @radius;
  -moz-border-radius: @radius;
}

#header {
  .rounded-corners;
}

#footer {
  .rounded-corners(10px);
}
```



```
/* 生成的 CSS */

#header {
  border-radius: 5px;
  -webkit-border-radius: 5px;
  -moz-border-radius: 5px;
}

#footer {
  border-radius: 10px;
  -webkit-border-radius: 10px;
  -moz-border-radius: 10px;
}
```

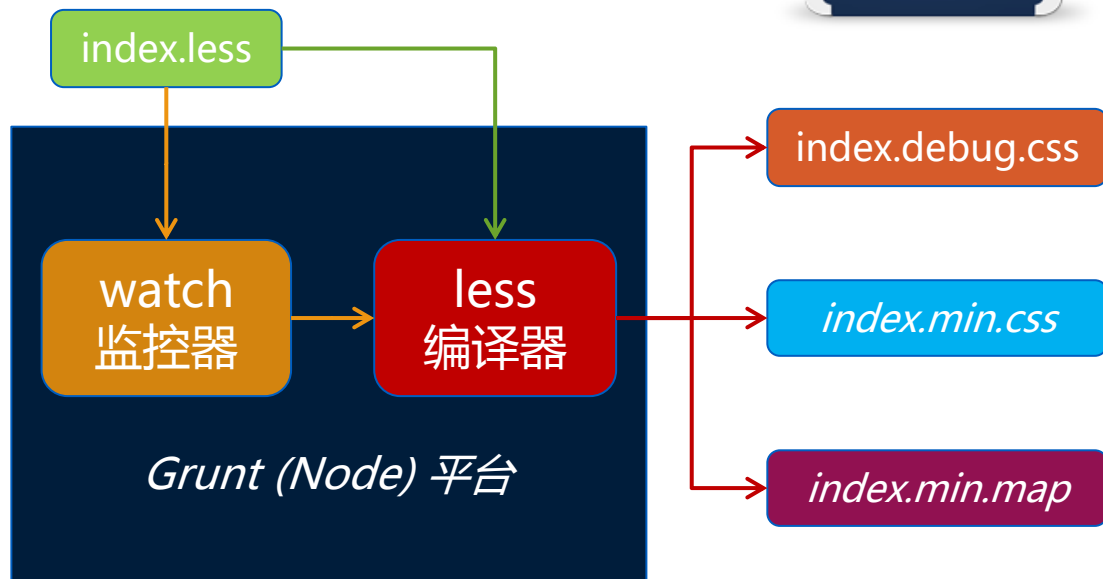
混入可以将一个定义好的 class A 轻松的引入到另一个 class B 中，从而简单实现 class B 继承 class A 中的所有属性。我们还可以带参数地调用，就像使用函数一样。



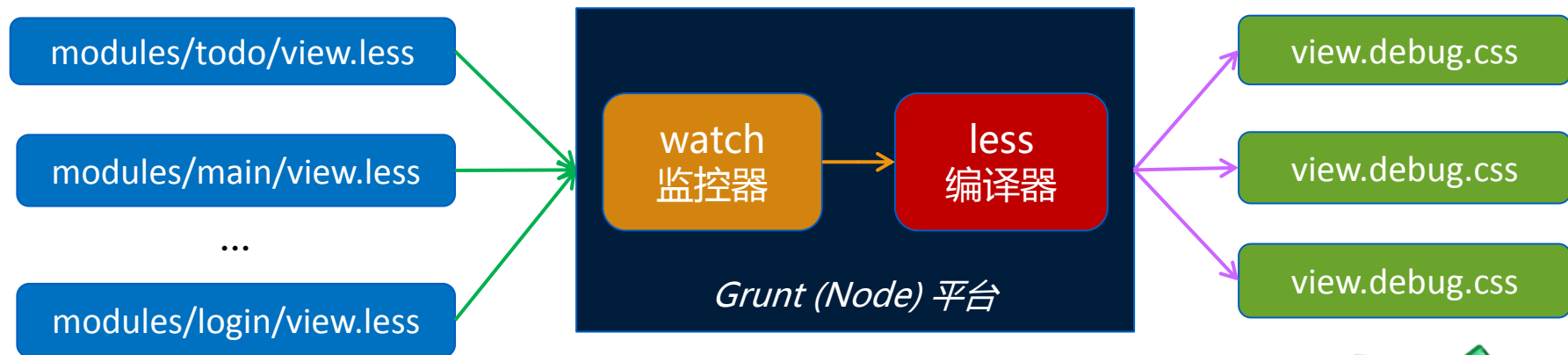


这样做的好处

1. 自动化编译，避免手动操作。
2. 即时生效，修改-保存-刷新即可看到效果。
3. 高效开发，让人感觉浏览器直接原生支持 less 一样。
4. 方便后期对页面做进一步自动化处理。



1. 每个功能模块目录下都可以拥有自己的一个或多个 less 文件。
2. 所有的 less 文件都会给编译成 css 文件，存到 `htdocs/style/css/` 目录。
3. 团队成员互相不知道或不关心对方的 less 文件名。
4. 每个成员都有权且应该按照最合适的方式对 less 文件命名。
5. less 文件名的更改应该能自动应用到页面中，而不需要再手动更改。



用 grunt 自动重命名

路径转名称

只需要把 less 文件所在的完整的路径转成文件名即可。

modules/**todo**/view.less

modules.**todo**.view.debug.css

modules/**main**/view.less

modules.**main**.view.debug.css

modules/**login**/view.less

modules.**login**.view.debug.css



用 grunt 自动引入 css 文件

less 文件会生成对应的 css 文件到 /style/css/ 目录，并且具有唯一名称。可用 grunt 自动搜索对应的 less 文件，然后转换成 css 文件名列表，再插入到页面中，避免人工手动去操作，提高了开发者效率。

```
<!--grunt.css.begin-->
<script>
  [
    'style/less/index.less',
    'modules/**/*.less',
  ]
</script>
<!--grunt.css.end-->
```



```
<!--grunt.css.begin-->
<link href="style/css/style.less.index.debug.css?c9" rel="stylesheet" />
<link href="style/css/modules.card.comment.comment-list.debug.css?c9" />
<link href="style/css/modules.card.detail.card-detail.debug.css?c9" rel="stylesheet" />
<link href="style/css/modules.card.list.card-list.debug.css?c9" rel="stylesheet" />
<link href="style/css/modules.card.publish.card-publish.debug.css?c9" />
<link href="style/css/modules.card.select.card-select.debug.css?c9" rel="stylesheet" />
...
<link href="style/css/modules.todo.business-opportunity.business-
<link href="style/css/modules.todo.less.todo-allocation.debug.cs
<link href="style/css/modules.todo.less.todo-follow.debug.css?c9"
<link href="style/css/modules.todo.less.todo-order.debug.css?c9"
<link href="style/css/modules.todo.less.todo.debug.css?c9" rel="stylesheet" />
<link href="style/css/modules.todo.list.saler-list.debug.css?c9"
<!--grunt.css.end-->
```

这样做的好处

- ↪ 解放开发者手动管理 css 文件，避免手动引入/移除css文件，节省开发者时间。
- ↪ 屏蔽 css 细节，就如同直接支持 less 一样。
- ↪ 支持自动插入随机数，确保每次都应用到最新的更改。
- ↪ 添加和删除 less 文件，会映射到对应的 css 文件中，开发者只需要管理好 less 文件即可。
- ↪ 母版页更简洁，对所引入的 css 文件更清晰可见。
- ↪ 避免对母版页的修改，减少文件冲突。



用 grunt 自动引入 js 文件

用 CMD 模式去定义的 js 模块文件，它们在页面中的引入顺序是无关的。可以用 grunt 自动搜索对应的 js 文件，然后插入到页面中，避免人工手动去操作，提高了开发者效率。

```
<!--grunt.js.begin-->
<script>
[
  'lib/**/*.js',
  'modules/**/*.js',
  'index.js',
]
</script>
<!--grunt.js.end-->
```



```
<!--grunt.js.begin-->
<script src="lib/API.js?5c"></script>
<script src="lib/CloudAPI.js?5c"></script>
<script src="lib/CloudHome.js?5c"></script>
<script src="lib/CloudMessage.js?5c"></script>
<script src="lib/ImageReader.js?5c"></script>
...
<script src="modules/todo/js/ToDo.js?5c"></script>
<script src="modules/todo/list/SalerList/API.js?5c"></script>
<script src="modules/todo/list/SalerList/filter/Filter.js?5c"></script>
<script src="modules/todo/list/SalerList/Message.js?5c"></script>
<script src="modules/todo/list/SalerList.js?5c"></script>
<script src="index.js?5c"></script>
<!--grunt.js.end-->
```



这样做的好处

- ☁ 解放开发者手动管理 js 文件，避免手动引入/移除 js 文件，节省开发者时间。
- ☁ 支持自动插入随机数，确保每次都应用到最新的更改。
- ☁ 添加和删除 js 文件，会映射到母版页中，开发者只需要管理好自己的模块 js 文件即可。
- ☁ 母版页更简洁，对所引入的 js 文件更清晰可见
- ☁ 避免对母版页的修改，减少文件冲突。



在前端开发领域，一个模块，可以是JS 模块，也可以是 CSS 模块，或是 Template 等模块。在这里，我们专注于 JS 模块。

1. 模块是一段 JavaScript 代码，具有统一的基本书写格式。
2. 模块之间通过基本交互规则，能彼此引用，协同工作。

把上面两点中提及的基本书写格式和基本交互规则描述清楚，就能构建出一个模块系统。



所有 JavaScript 模块都遵循 CMD (Common Module Definition) 模块定义规范。

- ☞ 在 CMD 规范中，一个模块就是一个文件
- ☞ 模块定义通过 *define* 函数实现
- ☞ 模块加载通过 *require* 函数实现
- ☞ 模块定义里通过 *module.exports* 或 *return* 导出模块
- ☞ 模块加载用同步的方式 *var module = require('...');*
- ☞ 延迟和按需构造模块，提高性能
- ☞ 模块功能单一，提高可维护性

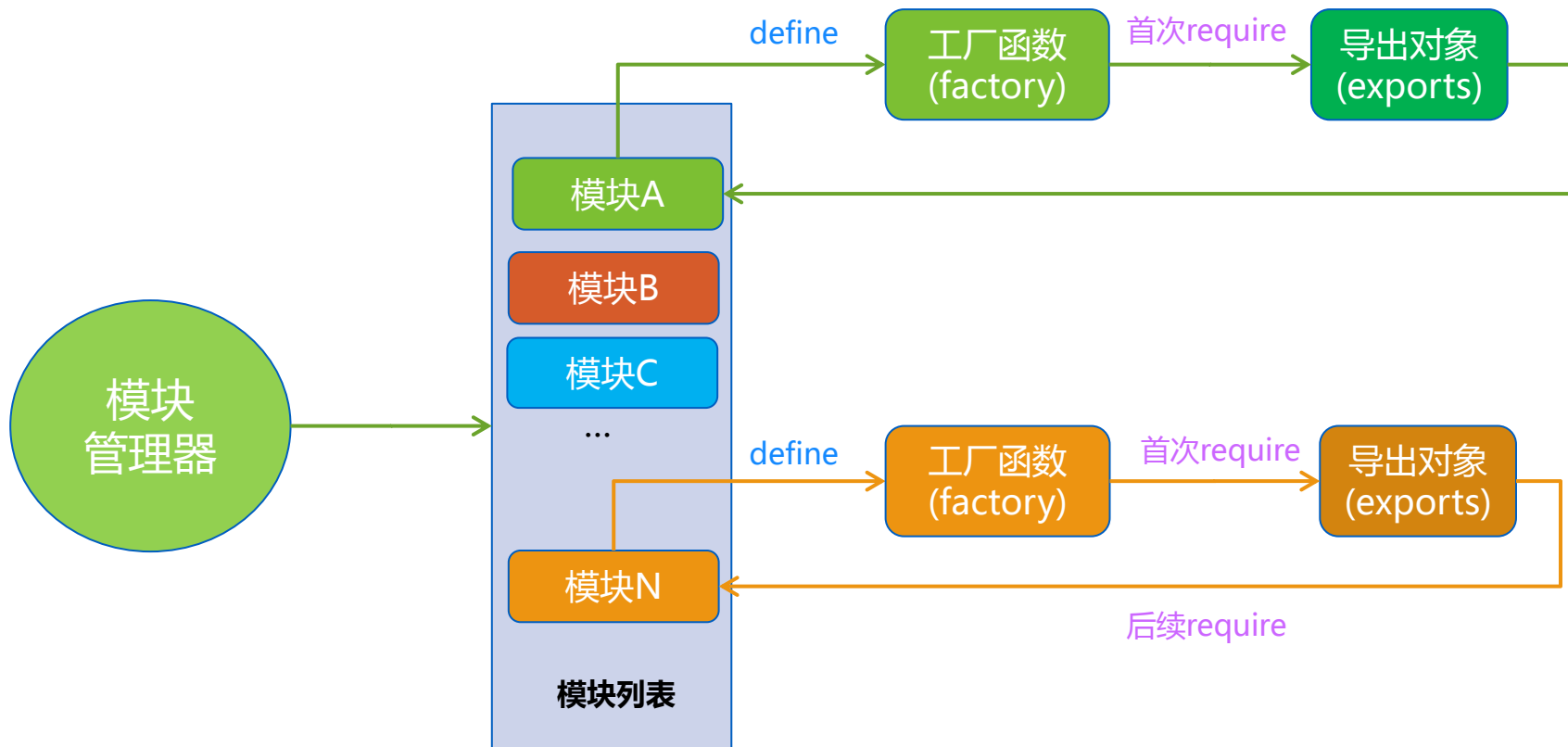
<https://github.com/seajs/seajs/issues/242>
<https://github.com/seajs/seajs/issues/547>
<http://seajs.org/docs/#docs>



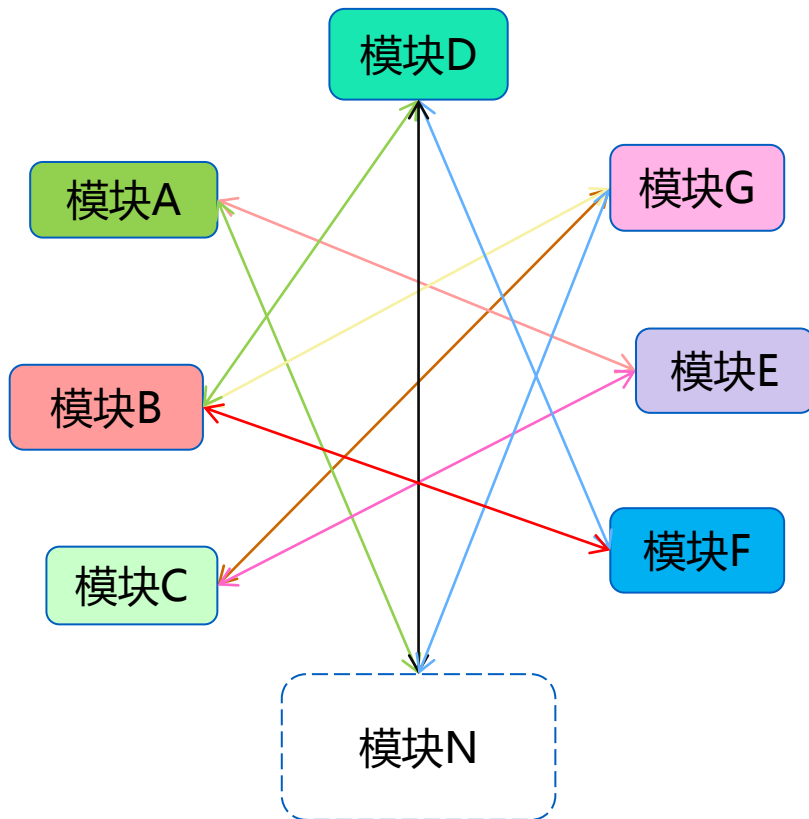
☁ 模块定义 define

☁ 模块加载 require

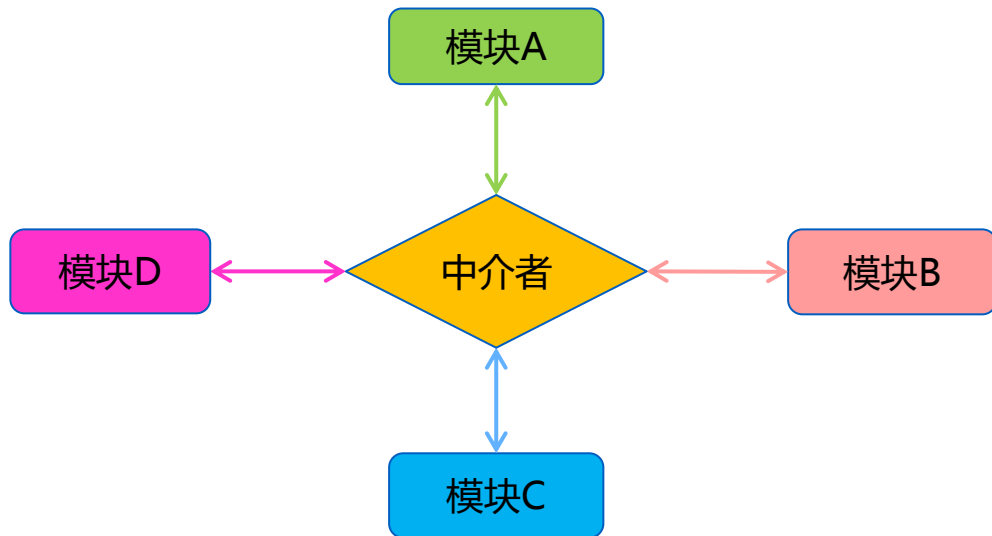




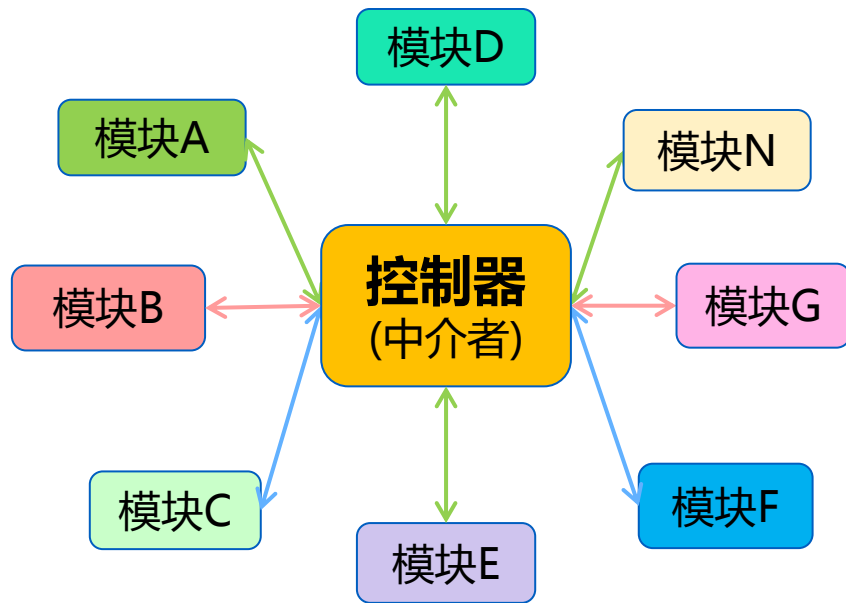
- 应用程序，无论其大小，都由一些单个的对象（模块）所组成。
- 随着应用程序的增长，将添加越来越多的对象，在代码重构期间，对象可能会被删除或重新整理。
- 所有这些对象需要一种方式来实现相互通信，要求不能降低可维护性，也不能因为改变应用程序的某部分而破坏其余部分。
- 当对象互相知道太多信息并直接通信（调用对方的方法/改变属性）时，会导致不良的 **紧耦合** 问题。
- 当对象紧密耦合时，很难做到在改变单个对象的同时而不影响其他多个对象，同时很难评估需要的时间和风险。



- 中介者模式促成松耦合，提高了可维护性。
- 对象之间无法互相看到对方的存在，就像对方是完全透明的，更不能直接通信。
- 一个对象仅能看到自己内部的成员和状态，当状态发生改变后，如果想传达给外界，则通过消息（事件）机制通知中介者，由中介者去调用其他对象的成员方法。
- 消息的触发，只能由对象自身去完成，外界只能监听该消息。



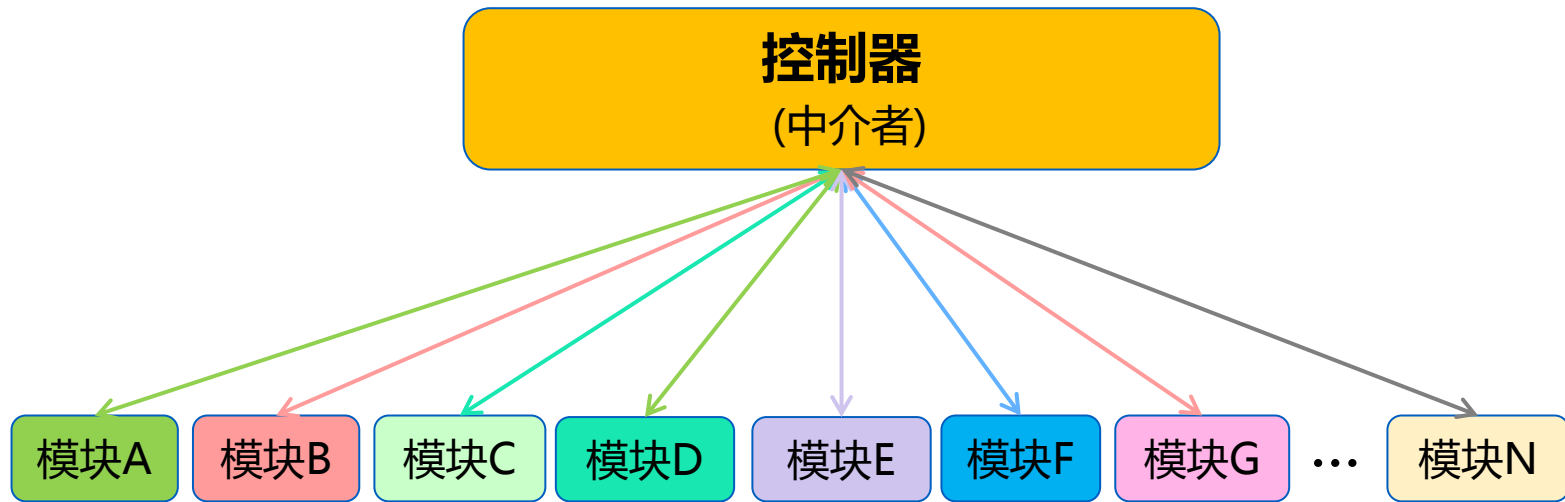
- 模块与模块之间互相独立，互不依赖和引用
- 模块内部维护自身的状态的正确性
- 模块内部的事件只能由模块自身触发
- 模块暴露接口给外界(控制器)监听模块事件
- 模块暴露必要接口给外界(控制器)调用
- 所有模块对控制器可见
- 在控制器里绑定模块与模块之间的事件监听



中介者模型



本质上是一树形结构，每个节点（模块）对直接父节点（控制器）可见，兄弟节点之间互相看不见对方的存在。同时也是一种上下级的关系，下属只对直接上级可见，同级之间互相不可见。

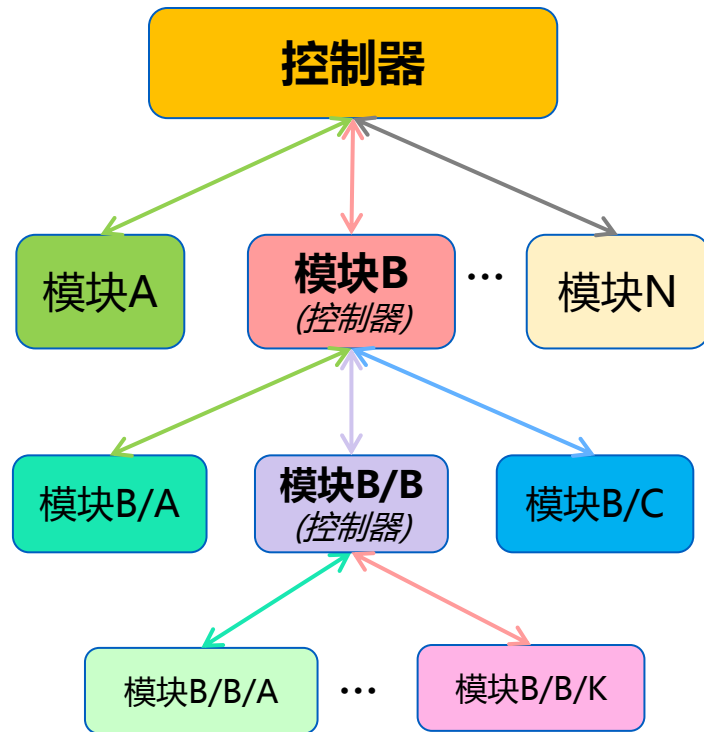


树形结构



中介者模式下的模块划分

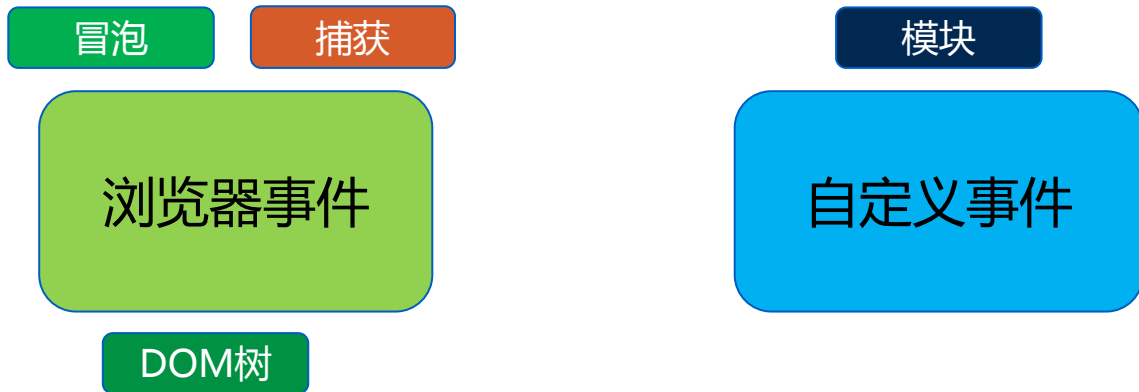
- ☞ 模块要足够简单，功能要单一。
- ☞ 一个模块单独成一个文件，文件内只能有模块定义(define)调用，而不能有其它逻辑。
- ☞ 兄弟模块之间互不可见，互不引用和依赖。
- ☞ 父模块充当直接子模块的控制器(中介者)，不属于自己的直接子模块不可见。
- ☞ 模块的命名采用目录层级形式，如A/B/C，且在父模块所在的目录建立同名的目录名，直接子模块放入其内。
- ☞ 当一个模块开始变得复杂时，可采用递归和分而治之的方式进一步划分成控制器和直接子模块。



树形结构



1. 该模式也叫 **观察者模式** 或 **订阅/发布模式**。
2. 设计这种模式背后的主要动机是促进松散耦合。
3. 在这种模式中，并不是一个对象调用另一对象的方法，而是一个对象订阅另一个对象的特定活动并在该活动发生改变后获得通知。
4. 订阅者也称之为观察者，而被观察的对象成为发布者。
5. 当发生了一个重要的事件时，发布者将通知（调用）所有订阅者并且以事件对象的形式传递消息。

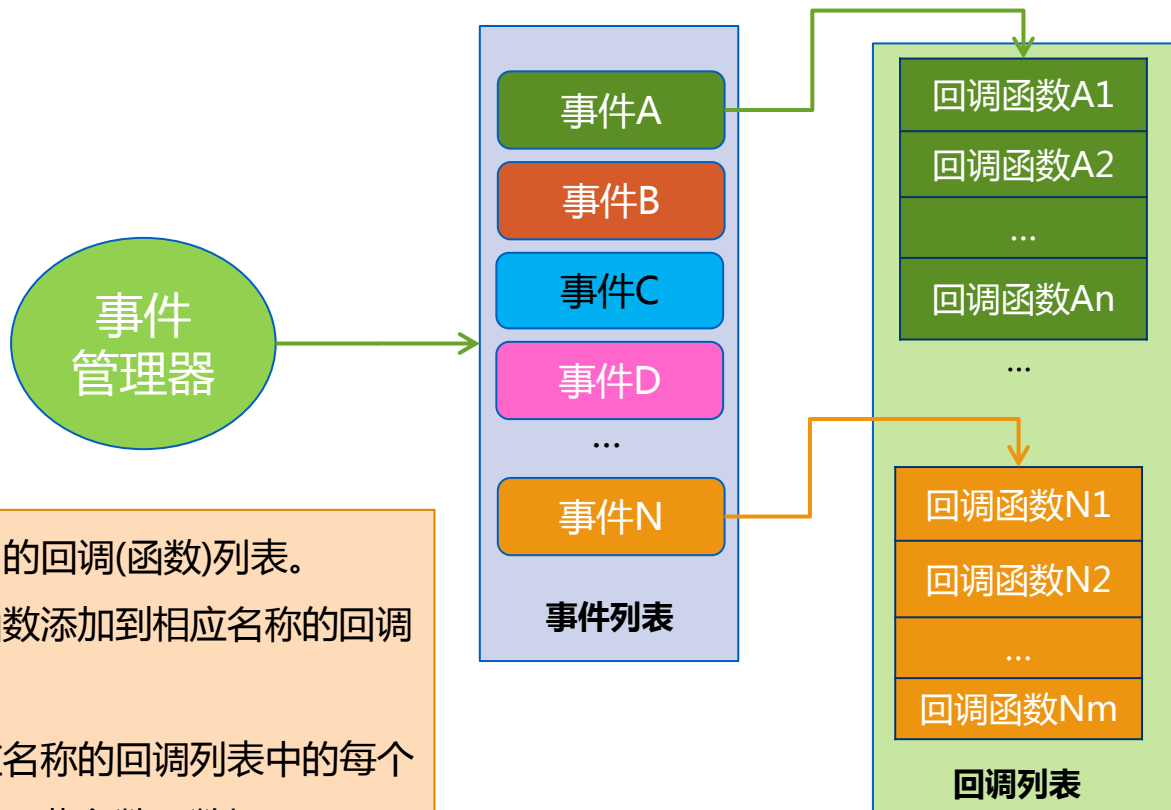


☁ 事件绑定

on (name, fn)

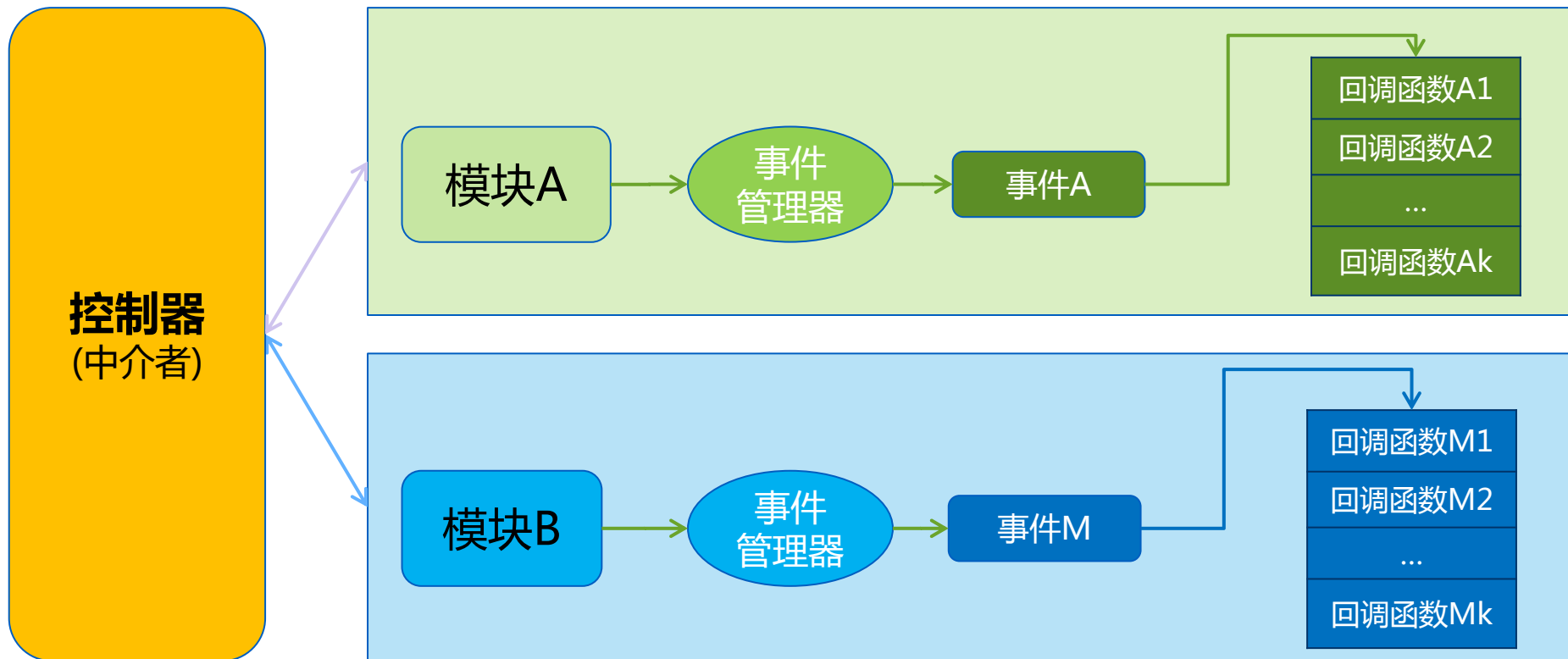
☁ 事件触发

fire(name, args)



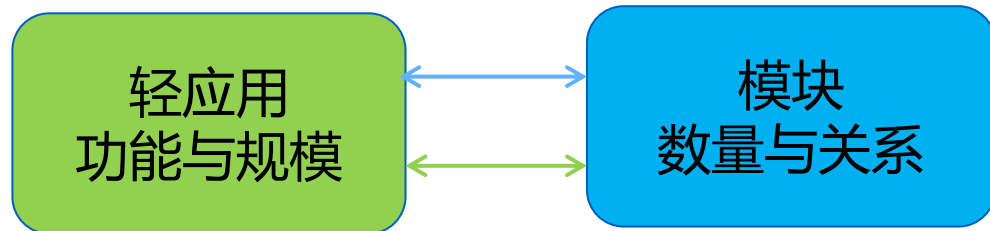
- 自定义事件，实质上是一个已命名的回调(函数)列表。
- 绑定事件，实质上是把一个回调函数添加到相应名称的回调（事件处理程序）列表中。
- 触发事件，实质上是逐一执行相应名称的回调列表中的每个函数（事件处理程序），并可传递一些参数（数据）。





轻应用里的模块特点

1. 模块数量庞大
2. 模块之间的依赖关系复杂
3. 模块文件存放的物理位置不能影响其逻辑关系
4. 模块与模块之间的逻辑关系不依赖于其物理位置



两者具有正相关的关系，即轻应用功能规模越大，模块的数量就会越多，模块之间的关系就会越复杂



define(id, factory);

```
1 define('User/Menus', function (require, module, exports) {  
2  
3     //加载公共模块  
4     var $ = require('$');  
5  
6     //加载子模块  
7     var API = require(module, 'API');  
8  
9     //导出对象  
10    return {};  
11 });
```

在定义模块时指定了名称(id)的模块，就叫 **具名模块**。



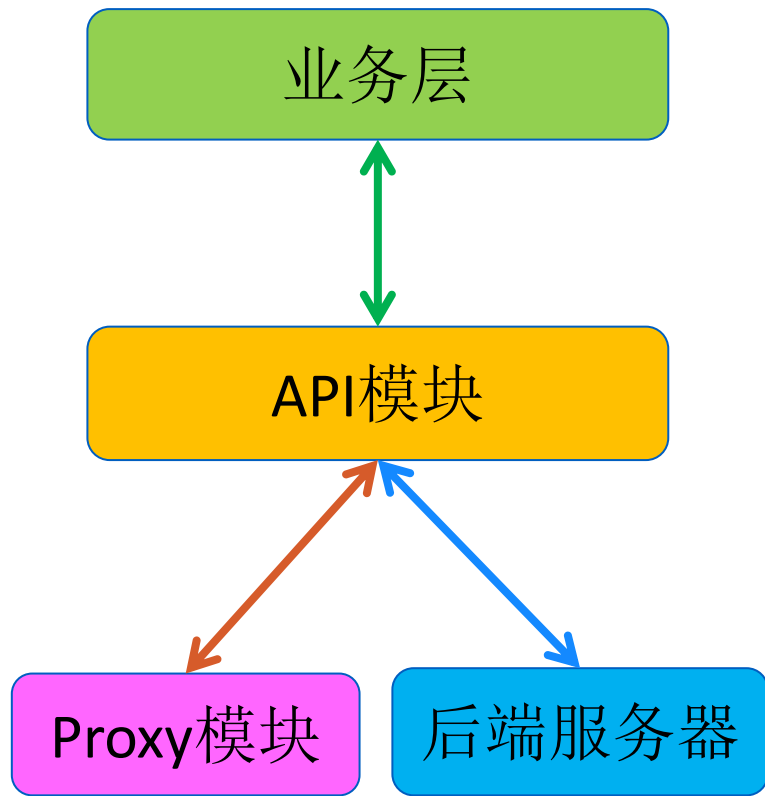
具名
模块

- ☞ 参数 **id** 表示模块的名称(id)，具有**唯一性**。
- ☞ 参数 **factory** 表示工厂函数，在首次 **require** 时会给调用，且**仅调用一次**。
- ☞ id 既表示了模块的名称，也决定了其所在的逻辑位置，即表示出其与父模块关系。
- ☞ id 中带有 **"/"** 的表示私有模块，仅其父模块可加载与调用。
- ☞ id 中不带 **"/"** 的表示公共模块，对所有模块可见。



- 增加了模块的可见性约束，让整个模块系统形成一棵模块树，避免了网状结构的引用和依赖方式。
- 强制使用具名的模块定义方式，解除绑定模块路径与名称的关系，也消除了模块位置发生变化时带来的一连串依赖改动。
- 增加了父模块加载子模块的约束方式，简化了长命名的加载方案。
- 模块与模块的逻辑关系仅通过名称即可限定，有效解决了模块之间的复杂的依赖关系。
- 可以很好的使用事件驱动的方式进行模块之间的通信，从而解除耦合。



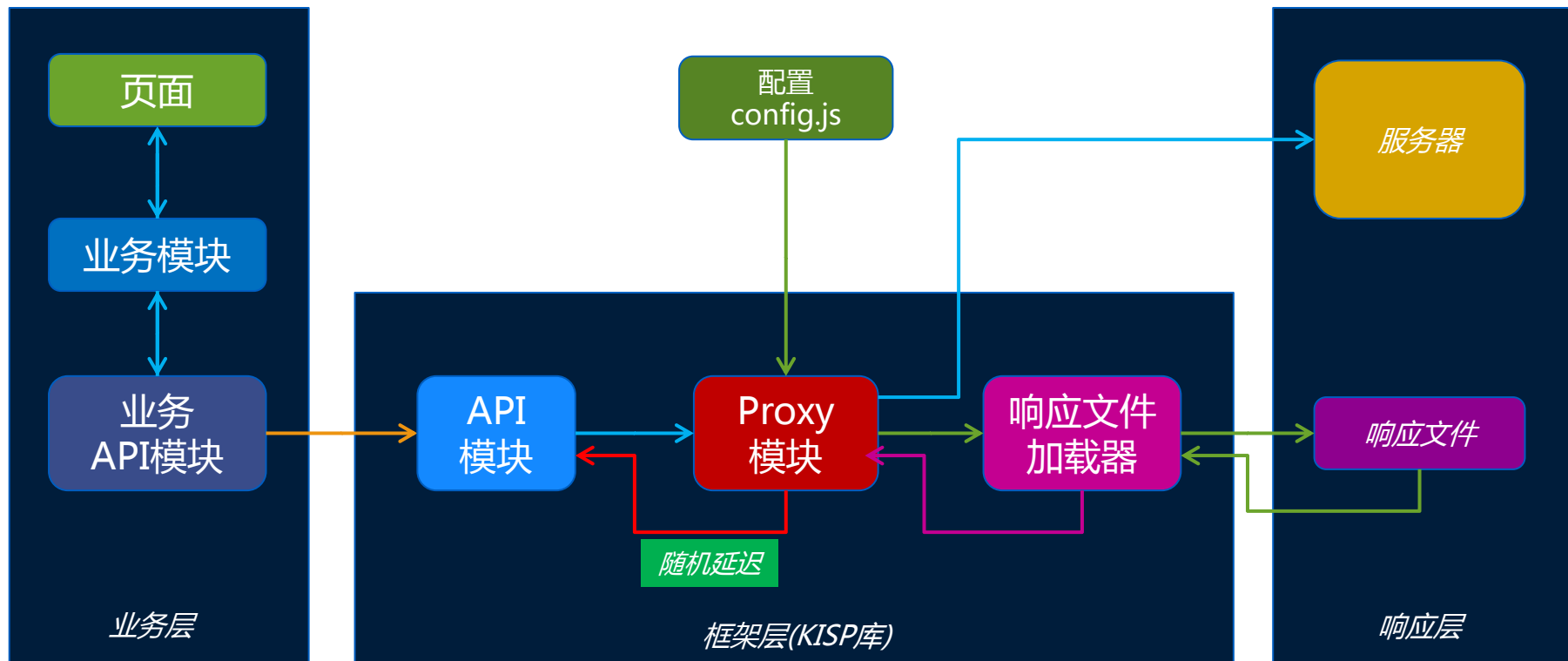


这样做的好处

1. 前端页面可以脱离后台接口运行，提高并行开发的效率，更直接提高了前端的开发效率。
2. 方便前端修改业务数据，针对不同的业务数据进行测试，更容易发现由于数据的不同而导致的问题。
3. 避免后台接口不可用而影响前端的开发进度。
4. 真正实现前后端完全分离、并发开发的利器，让前端有自己独立的设计和开发模式。
5. 业务层对数据的来源一无所知，避免底层数据的改动而导致业务层逻辑的改动。



本地代理模拟服务器响应过程

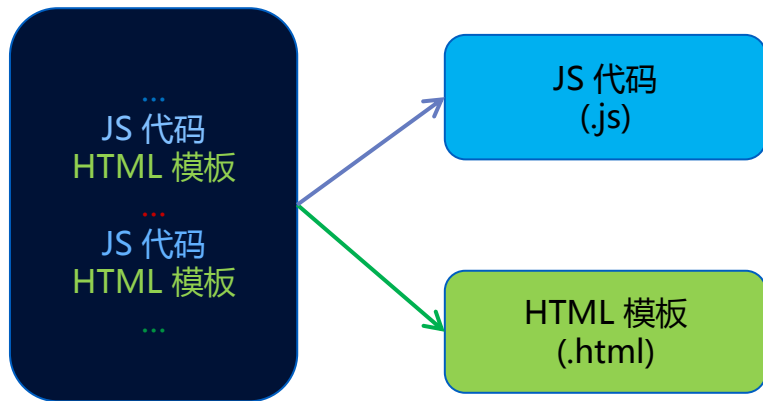


将HTML模板从代码中分离出来

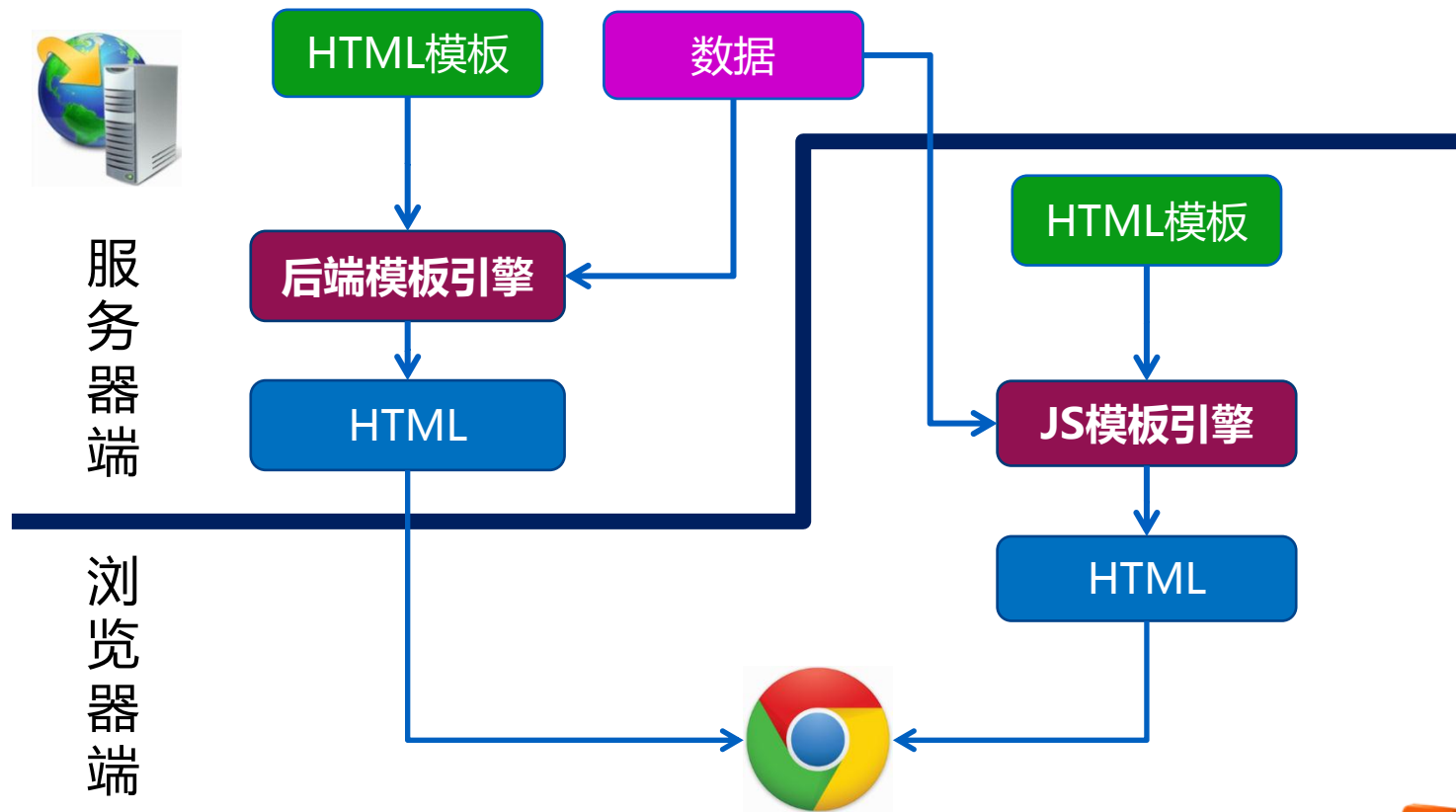
HTML模板用来提供给JS代码进行模板填充，从而生成页面UI元素。对JS来说，模板填充的本质只是在操作字符串，相比于传统的直接操作DOM元素来说，效率是非常高的。

这样做的好处

1. js 代码更清晰简洁，更容易维护
2. html 模板更容易修改，不易出错
3. js 性能更好，避免了字符串的拼接
4. js 代码和 html 模板更容易复用和共享



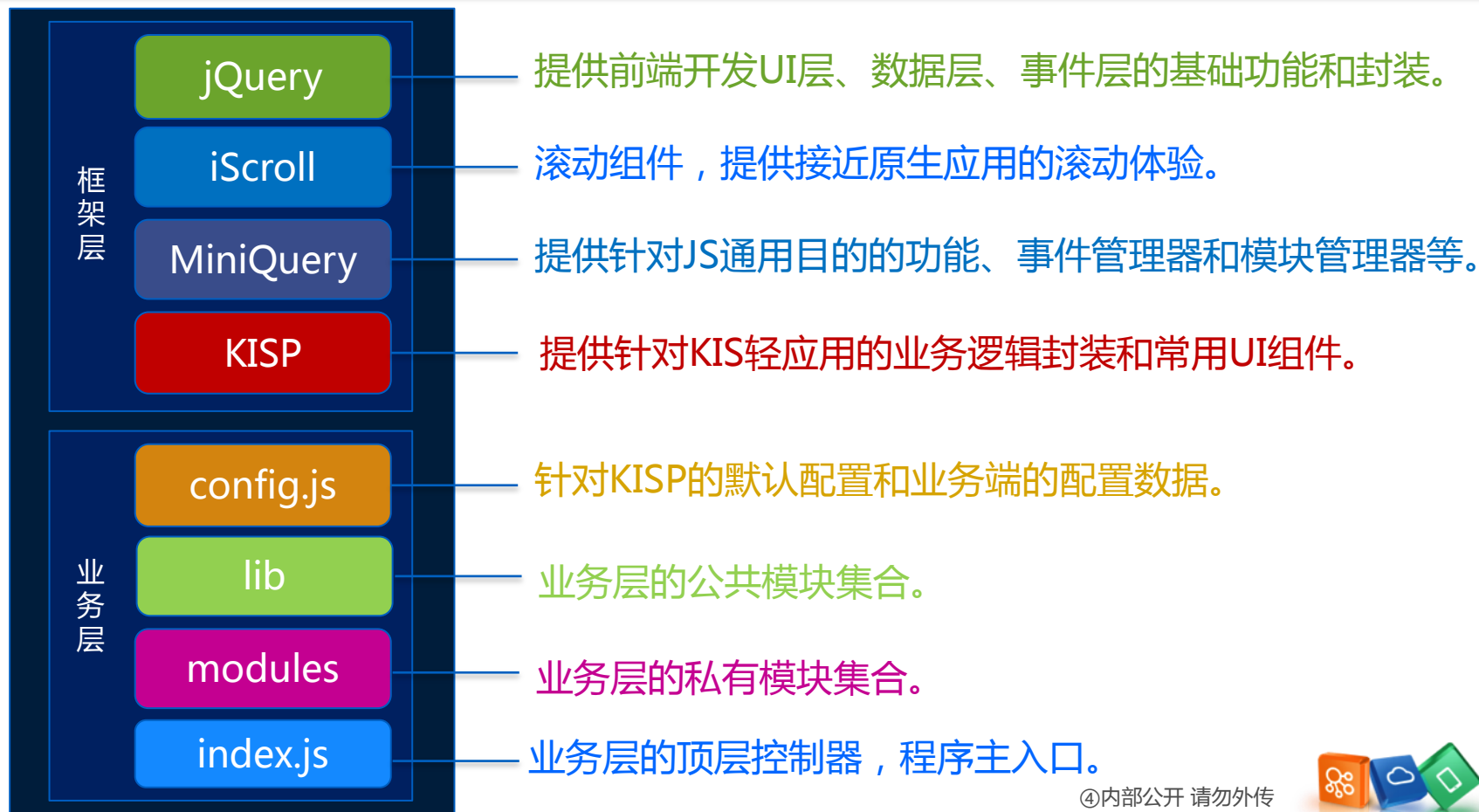
前后端的模板填充模型



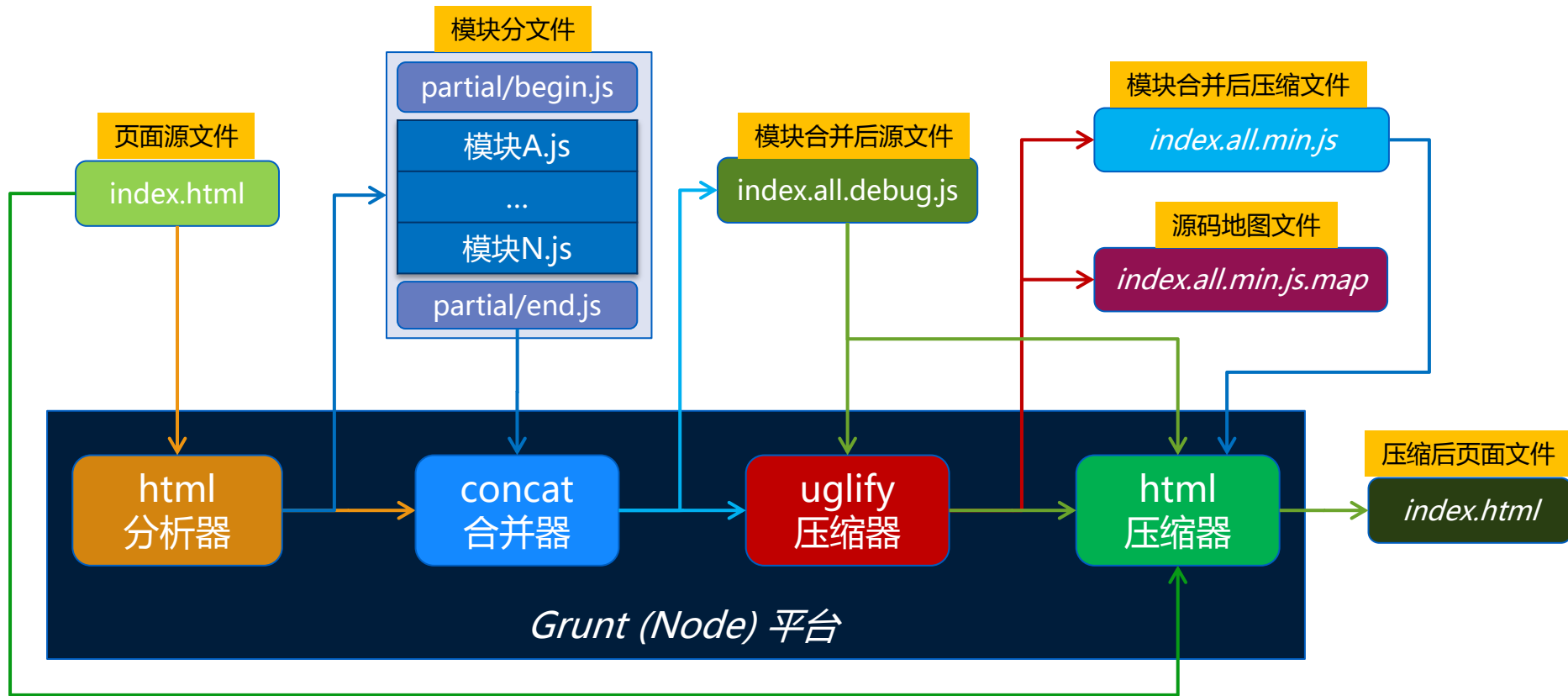
模板引擎（Template Engine）是Web开发中将展现层和数据层分离的一项技术。模板引擎根据一定的语义，将数据填充到模板中，产生最终的HTML页面。模板引擎渲染的位置可分为客户端和服务端端。

	优点	缺点
服务端	<ul style="list-style-type: none">• 计算快• 减少HTTP请求• 页面呈现较快• 对SEO友好	<ul style="list-style-type: none">• 不同的架构和语言，采用的渲染技术不同• 模板资源无法共享• 不利于前后端分离，后端掺杂HTML，让后端开发人员头疼
客户端	<ul style="list-style-type: none">• 负载均衡，减少服务器渲染压力• 节省流量，只传输数据• 前后端分离，服务器专注逻辑和数据；客户端专注展现和UI• 不必关心后端采用的技术，只要接口和数据不变，后端可以采用任何一种语言• 利于研发流程的分工，前后端可以并行开发	<ul style="list-style-type: none">• 数据跟HTML要分开HTTP请求，增加HTTP请求数





基于grunt的页面模块合并与引用替换过程



这样做的好处

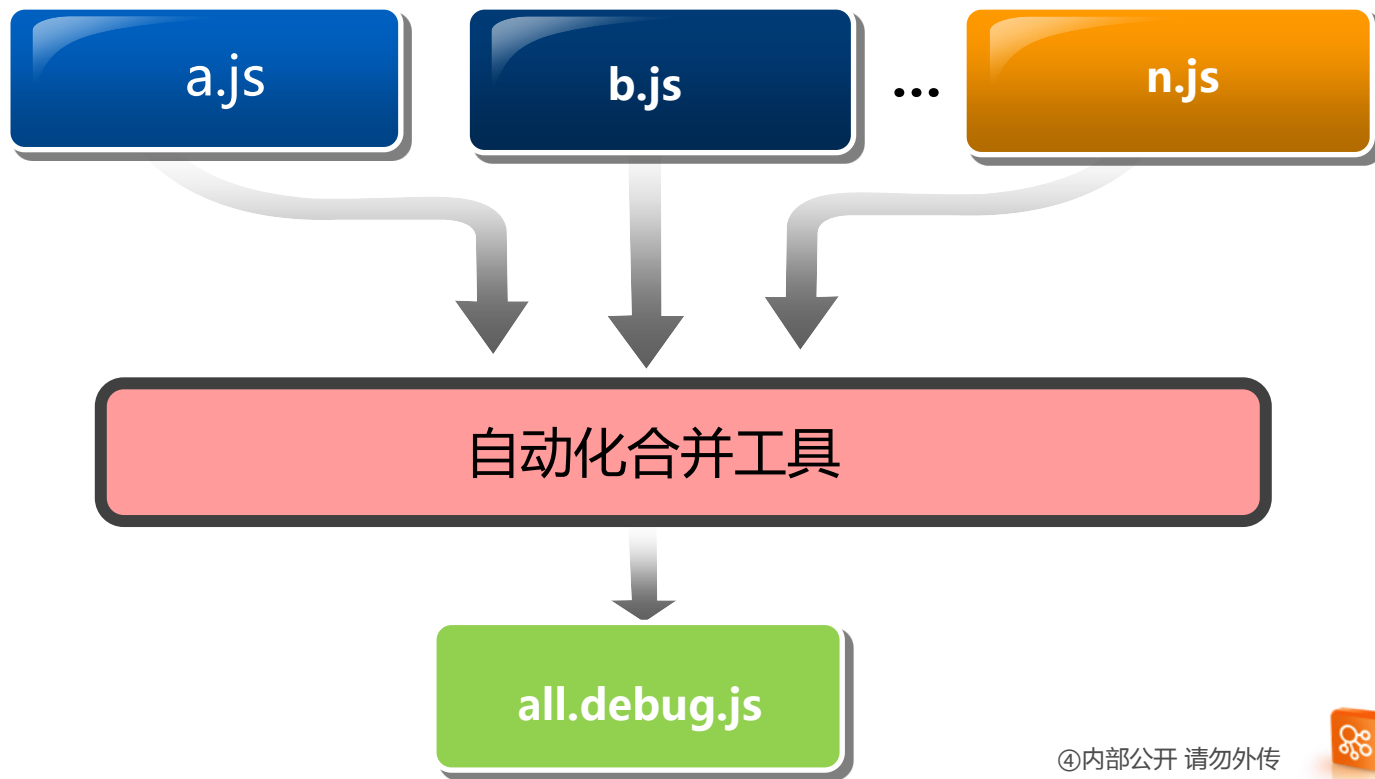
1. 提供一键合并、压缩和替换的自动化构建方式，让站点从开发到发布更高效、更智能。
2. 提供不同级别的发布控制，更加灵活，更加方便。
3. 基于Grunt JS进行开发，对前端人员更友好，可扩展性更好。

级别	描述
0	最原始的开发版，保留。
1	打包，压缩，删除所有的 .less 文件，并把分模块 js 文件打包成一个 .all.debug.js 引用并插入到 .html 文件中。
2	在 1 的基础上，压缩，把 .debug. 引用改成 .min. 插入到 .html 文件中。即修改 html 中相应的 .debug.css 和 .debug.js 的引用为 .min.css 和 .min.js。
3	在 2 的基础上，删除所有的 .map .debug.js .debug.css 文件。
4	在 3 的基础上，压缩 .html 文件。
5	在 4 的基础上，压缩 config.js 文件。

在 cmd 中运行 `grunt build:0|1|2|3|4|5` 即可调用



合并文件可以减少http请求数，是前端性能优化的首要任务



压缩文件可以进一步减小文件体积，加快文件加载速度，提高用户体验，是发布前的环节





[HOME](#) | [DOWNLOADS](#) | [DOCS](#) | [COMMUNITY](#) | [ABOUT](#) | [JOBS](#) | [BLOG](#)

Node.js® is a platform built on **Chrome's JavaScript runtime** for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.

Current Version: v0.10.29

INSTALL

[DOWNLOADS](#)

[API DOCS](#)

<http://www.nodejs.org/>

会自动根据你的操作系统
安装合适的位数版本



安装构建工具：Grunt

```
D:\Users\thinkpad\Desktop\KIS轻应用前端开发培训\approve-kisp\bin>grunt  
'grunt' 不是内部或外部命令，也不是可运行的程序  
或批处理文件。
```

安装前

```
D:\Users\thinkpad\Desktop\KIS轻应用前端开发培训\approve-kisp\bin>npm install -g  
grunt-cli  
C:\Users\thinkpad\AppData\Roaming\npm\grunt -> C:\Users\thinkpad\AppData\Roaming  
\npm\node_modules\grunt-cli\bin\grunt  
grunt-cli@0.1.13 C:\Users\thinkpad\AppData\Roaming\npm\node_modules\grunt-cli  
├── resolve@0.3.1  
├── nopt@1.0.10 (abbrev@1.0.5)  
├── findup-sync@0.1.3 (lodash@2.4.1, glob@3.2.11)
```

安装 grunt :
npm install -g grunt-cli

```
D:\Users\thinkpad\Desktop\KIS轻应用前端开发培训\approve-kisp\bin>npm install  
npm WARN package.json approve@2.0.0 No description  
npm WARN package.json approve@2.0.0 No repository field.  
npm WARN package.json approve@2.0.0 No README data
```

安装依赖的插件包：
npm install

node_modules	2014/7/31 3:45 ...	文件夹
begin.js	2014/7/17 2:46 ...	JS 文件
end.js	2014/7/17 2:46 ...	JS 文件
Gruntfile.js	2014/7/22 4:36 ...	JS 文件
package.json	2014/7/22 2:57 ...	JSON 文件



对 bin 目录进行构建：*grunt*

```
D:\Users\thinkpad\Desktop\KIS轻应用前端开发培训\approve-kisp\bin>grunt
Running "concat:css" <concat> task
File ../css/all.debug.css created.

Running "concat:js" <concat> task
File ../index.debug.js created.

Running "uglify:index" <uglify> task
File ../index.min.js created: 55.45 kB → 20.94 kB

Running "less:debug" <less> task
File ../css/index.debug.css created: 13.42 kB → 13.42 kB

Running "less:min" <less> task
File ../css/index.min.css created: 13.42 kB → 9.95 kB
File ../css/all.min.css created: 21.48 kB → 16.03 kB

Done, without errors.
```

构建成功！



未来...

目标：打造一个Web开发平台

让开发更轻松！

让团队更专注！

让产品更专业！



to do...



感謝
ขอบคุณ
terima kasih
Thanks
ありがとう
谢谢

