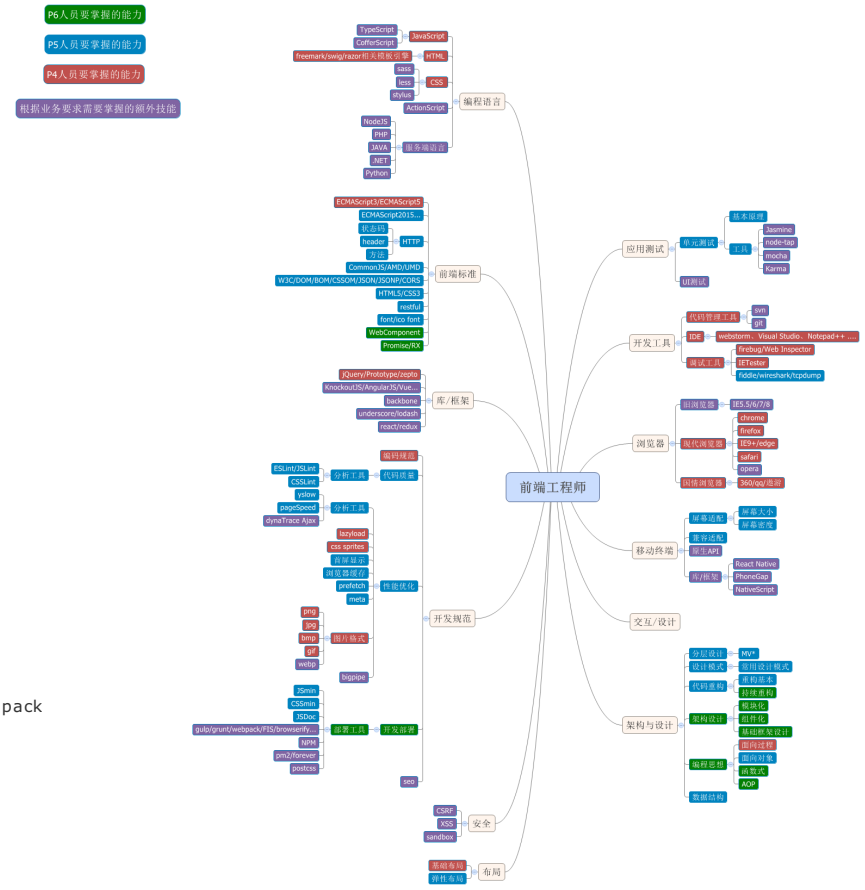
Web前端

技术树



checklist http://git.sdp.nd/fed/checklist

JS模块化的演进

模块系统主要解决模块的定义、依赖和导出，先来看看已经存在的模块系统。

<script>标签

```
<script src="module1.js"></script>
<script src="module2.js"></script>
```

这是最原始的 JavaScript 文件加载方式，如果把每一个文件看做是一个模块，那么他们的接口通常是暴露在全局作用域下，也就是定义在 window 对象中，不同模块的接口调用都是一个作用域中，一些复杂的框架，会使用命名空间的概念来组织这些模块的接口，典型的例子如 YUI 库。

> 这种原始的加载方式暴露了一些显而易见的弊端：

1. 全局作用域下容易造成变量冲突
2. 文件只能按照 <script> 的书写顺序进行加载
3. 开发人员必须主观解决模块和代码库的依赖关系
4. 在大型项目中各种资源难以管理，长期积累的问题导致代码库混乱不堪

CommonJS

http://wiki.commonjs.org/wiki/CommonJS

服务器端的 Node.js 遵循 CommonJS规范，该规范的核心思想是允许模块通过 require 方法来同步加载所要依赖的其他模块，然后通过 exports 或 module.exports 来导出需要暴露的接口。

```
require("module");
require("../file.js");
exports.doStuff = function() {};
module.exports = someValue;
```

优点

1. 服务器端模块便于重用
2. NPM 中已经有将近20万个可以使用模块包
3. 简单并容易使用

缺点：

1. 同步的模块加载方式不适合在浏览器环境中，同步意味着阻塞加载，浏览器资源是异步加载的
2. 不能非阻塞的并行加载多个模块

实现：

1. 服务器端的 Node.js
2. Browserify，浏览器端的 CommonJS 实现，可以使用 NPM 的模块，但是编译打包后的文件体积可能很大
3. modules-webmake，类似Browserify，还不如 Browserify 灵活
4. wreq，Browserify 的前身

AMD

https://github.com/amdjs/amdjs-api
Asynchronous Module Definition 规范其实只有一个主要接口 define(id?, dependencies?, factory)，它要在声明模块的时候指定所有的依赖 dependencies，并且还要当做形参传到 factory 中，对于依赖的模块提前执行，依赖前置。

```
define("module", ["dep1", "dep2"], function(d1, d2) {
  return someExportedValue;
});
require(["module", "../file"], function(module, file) { /* ... */ });
```

优点：

1. 适合在浏览器环境中异步加载模块
2. 可以并行加载多个模块

缺点：

1. 提高了开发成本，代码的阅读和书写比较困难，模块定义方式的语义不顺畅
2. 不符合通用的模块化思维方式，是一种妥协的实现

实现：

1. RequireJS
2. curl

CMD

https://github.com/cmdjs/specification/blob/master/draft/module.md
Common Module Definition 规范和 AMD 很相似，尽量保持简单，并与 CommonJS 和 Node.js 的 Modules 规范保持了很大的兼容性。

```
define(function(require, exports, module) {
  var $ = require('jquery');
  var Spinning = require('./spinning');
  exports.doSomething = ...
  module.exports = ...
})
```

优点：

1. 依赖就近，延迟执行
2. 可以很容易在 Node.js 中运行

缺点：

1. 依赖 SPM 打包，模块的加载逻辑偏重

实现：

1. Sea.js
2. coolie

UMD

https://github.com/umdjs/umd
Universal Module Definition 规范类似于兼容 CommonJS 和 AMD 的语法糖，是模块定义的跨平台解决方案。

ES6 模块

入门 http://es6.ruanyifeng.com/
EcmaScript6 标准增加了 JavaScript 语言层面的模块体系定义。ES6 模块的设计思想，是尽量的静态化，使得编译时就能确定模块的依赖关系，以及输入和输出的变量。CommonJS 和 AMD 模块，都只能在运行时确定这些东西。

```
import "jquery";
export function doStuff() {}
module "localModule" {}
```

优点：

1. 容易进行静态分析
2. 面向未来的 EcmaScript 标准

缺点：

1. 原生浏览器端还没有实现该标准
2. 全新的命令字，新版的 Node.js才支持

实现：

1. Babel

期望的模块系统

可以兼容多种模块风格，尽量可以利用已有的代码，不仅仅只是 JavaScript 模块化，还有 CSS、图片、字体等资源也需要模块化。

前端模块加载

前端模块要在客户端中执行，所以他们需要增量加载到浏览器中。

模块的加载和传输，我们首先能想到两种极端的方式，一种是每个模块文件都单独请求，另一种是把所有模块打包成一个文件然后只请求一次。显而易见，每个模块都发起单独的请求造成了请求次数过多，导致应用启动速度慢；一次请求加载所有模块导致流量浪费、初始化过程慢。这两种方式都不是好的解决方案，它们过于简单粗暴。

分块传输，按需进行懒加载，在实际用到某些模块的时候再增量更新，才是较为合理的模块加载方案。

要实现模块的按需加载，就需要一个对整个代码库中的模块进行静态分析、编译打包的过程。

所有资源都是模块

在上面的分析过程中，我们提到的模块仅仅是指JavaScript模块文件。然而，在前端开发过程中还涉及到样式、图片、字体、HTML 模板等等众多的资源。这些资源还会以各种方言的形式存在，比如 coffeescript、 less、 sass、众多的模板库、多语言系统（i18n）等等。

如果他们都可以视作模块，并且都可以通过require的方式来加载，将带来优雅的开发体验，比如：

```
require("./style.css");
require("./style.less");
require("./template.jade");
require("./image.png");
```

那么如何做到让 require 能加载各种资源呢？

静态分析

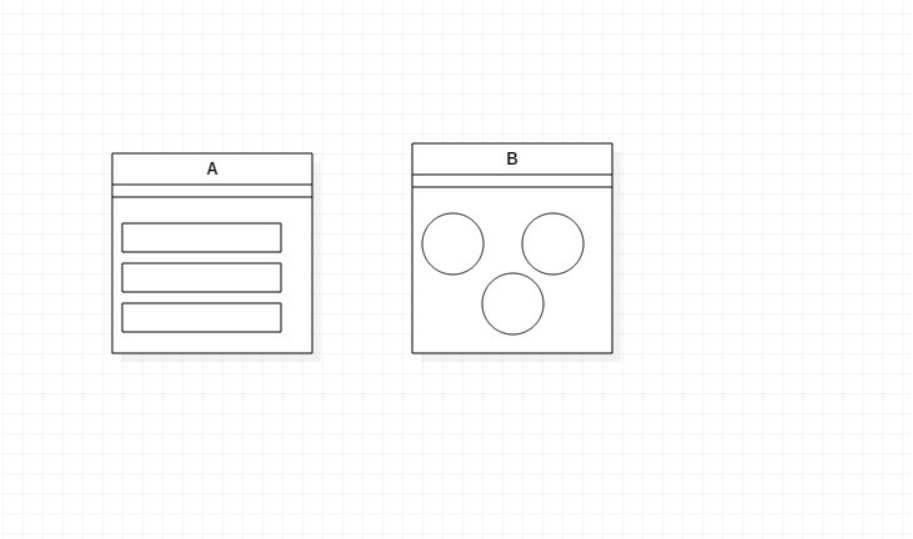
在编译的时候，要对整个代码进行静态分析，分析出各个模块的类型和它们依赖关系，然后将不同类型的模块提交给适配的加载器来处理。比如一个用 LESS 写的样式模块，可以先用 LESS 加载器将它转成一个CSS 模块，在通过 CSS 模块把他插入到页面的 <style> 标签中执行。Webpack 就是在这样的需求中应运而生。

同时，为了能利用已经存在的各种框架、库和已经写好的文件，我们还需要一个模块加载的兼容策略，来避免重写所有的模块。

css模块化

用面向对象的三大特性来说明CSS模块化——封装、继承、多态

封装



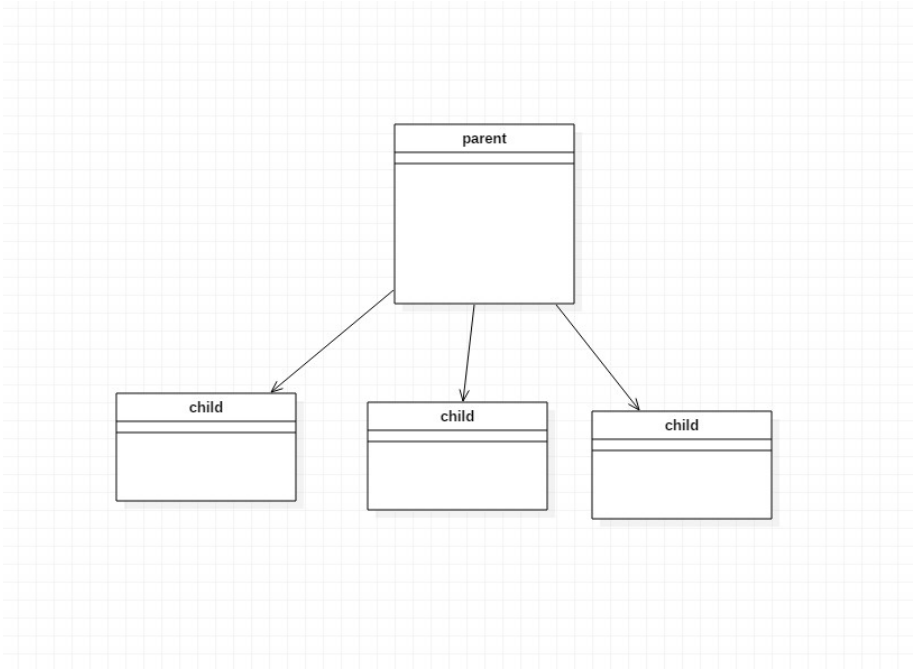
封装是实现CSS模块化的最基本要求，封装成的各个单元就是基本的CSS模块，可灵活用于组建页面的各种显示样式。
HTML代码

```
<div class=”module-a”>
  <h3>标题1</h3>
  <p>描述文字</p>
</div>
<div class=”module-b”>
  <h3>标题2</h3>
  <ul>
    <li>列表</li>
    <li>列表</li>
    <li>列表</li>
  </ul>
</div>
```

CSS代码

```
.module-a{.....}
.module-a h3{.....}
.module-a p{.....}
.module-b{.....}
.module-b h3{.....}
.module-b ul{.....}
.module-b ul li{.....}
.module-b ul li{.....}
.module-b ul li{.....}
```

继承



继承可谓是CSS模块化的关键所在

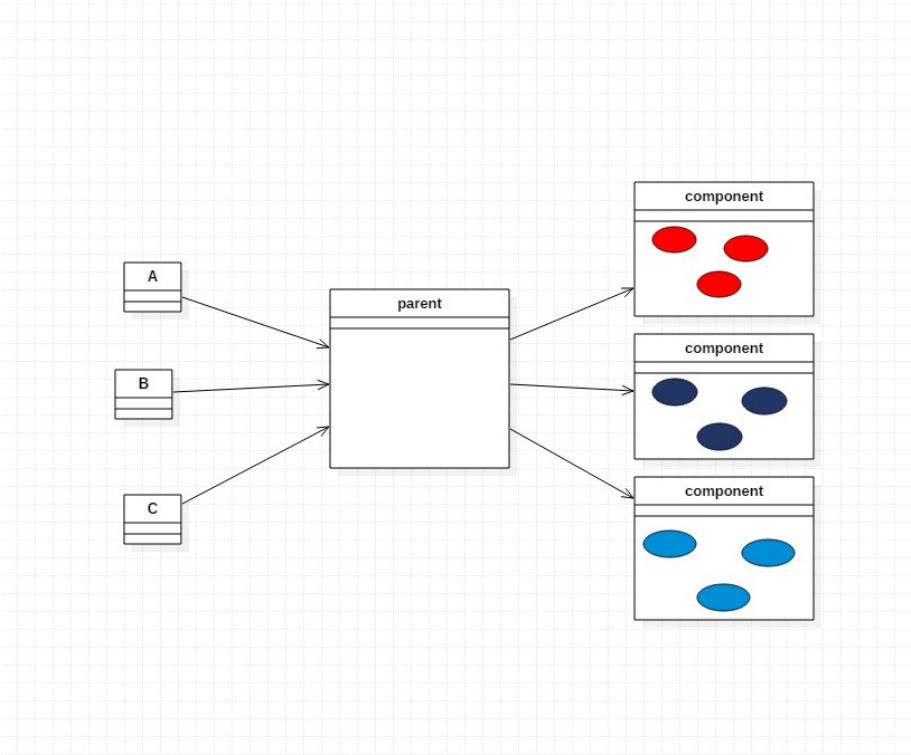
HTML代码

```
<div class=” module module-A module-A-a”></div>

<div class=” module module-A module-A-b”></div>
```

CSS代码

```
.module {.....}

    .module-A {.....}

        .module-A-a {.....}

        .module-A-b {.....}

    .module-B {.....}

        .module-B-a {.....}

        .module-B-b {.....}
```

多态



多态主要用于同一模块在页面的不同部分或者不同页面之间呈现出不同的样式

HTML代码

主页：

```
<body class="index">
  <ul class="nav">
    <li class="index"><a href="#">主页</a></li>
    <li class="page1"><a href="#">内页1</a></li>
    <li class="page2"><a href="#">内页2</a></li>
  </ul>
</body>
```

内页1：

```
<body class="page1">
  <ul class="nav">
    <li class="index"><a href="#">主页</a></li>
    <li class="page1"><a href="#">内页1</a></li>
    <li class="page2"><a href="#">内页2</a></li>
  </ul>
</body>
```

CSS代码

```
//定义常态

.nav{.....}

.nav .index{.....}

.nav .page1{.....}

.nav .page2{.....}

//定义高亮态

.index .nav .index{.....}

.page1 .nav .page1{.....}

.page2 .nav .page2{.....}
```

以上这个就是多态的经典应用之一。

Sass

http://sass.bootcss.com/
成熟、稳定、强大的 CSS 扩展语言.

> 编译工具：gulp-sass,grunt-sass,sass-loader

包管理

1. npm https://npmjs.org/doc/
2. bower

工具

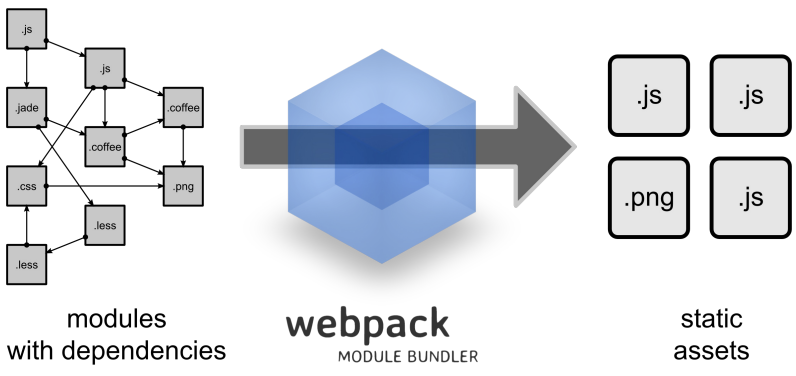
es6编译工具

> babel https://github.com/babel/babel
> Traceur https://github.com/google/traceur-compiler

前端工程与自动化

一般gulp+webpack

> webpack 当下最热门的一个模块打包器。它将根据模块的依赖关系进行静态分析，然后将这些模块按照指定的规则生成对应的静态资源。



> gulp 自动化构建工具 http://www.gulpjs.com.cn/
> grunt 自动化构建工具 http://www.gruntjs.net/

常用的工具(以gulp 为例，webpack,grunt基本上也都有)

```
// include gulp-util
var gutil = require('gulp-util');

// include jshint
var jshint = require('gulp-jshint');        //js语法检查
var concat = require('gulp-concat');        //合并文件
var uglify = require('gulp-uglify');        //js压缩代码
var csslint = require('gulp-csslint');      //css语法检查
var minifycss = require('gulp-minify-css');         //css压缩
var htmlhint = require('gulp-htmlhint');        //html语法检查
var imagemin = require('gulp-imagemin');        //图片压缩
var rename = require('gulp-rename');        //文件更名
var rev = require('gulp-rev');          //更改文件版本号
var revcollector = require('gulp-rev-collector');       //用于rev生成版本号后，替换页面路径
var notify = require('gulp-notify');            //提示信息
var browserSync = require('browser-sync');          //browser-sync自动刷新
var fileinclude = require('gulp-file-include');             // includer 包含模版文件
```

目前较火的前端MVVM框架

1. reactjs
   stars:48000+
2. angular
3. vuejs
   stars:26000+

threejs

> 官网http://threejs.org/
> 关于threejs的书籍只有英文原版的：
> 《Three.Js Essentials》、《threejs-cookbook》、《Learning Three.js》
> 源代码注释：https://github.com/omni360/three.js.sourcecode
> unity-to-threejs
> http://helloenjoy.com/2013/from-unity-to-three-js/
> http://helloracer.com/