

前端工程

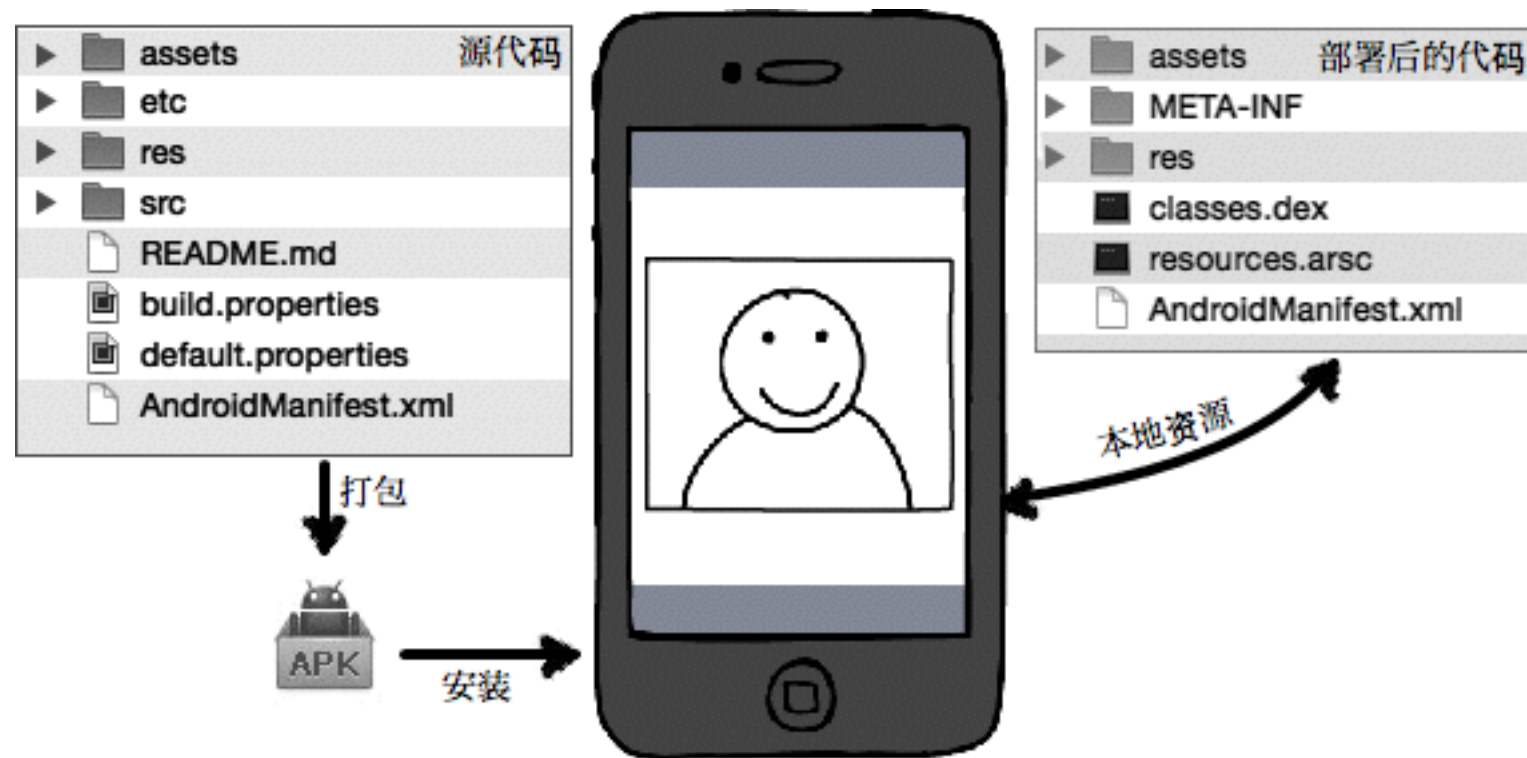
by 张云龙

“从输入URL到页面加载完的过程中都发生了什么事情？”

一道著名的前端技术面试题

“从准备写第一行代码到项目发布上线的过程中都发生了什么事情？”

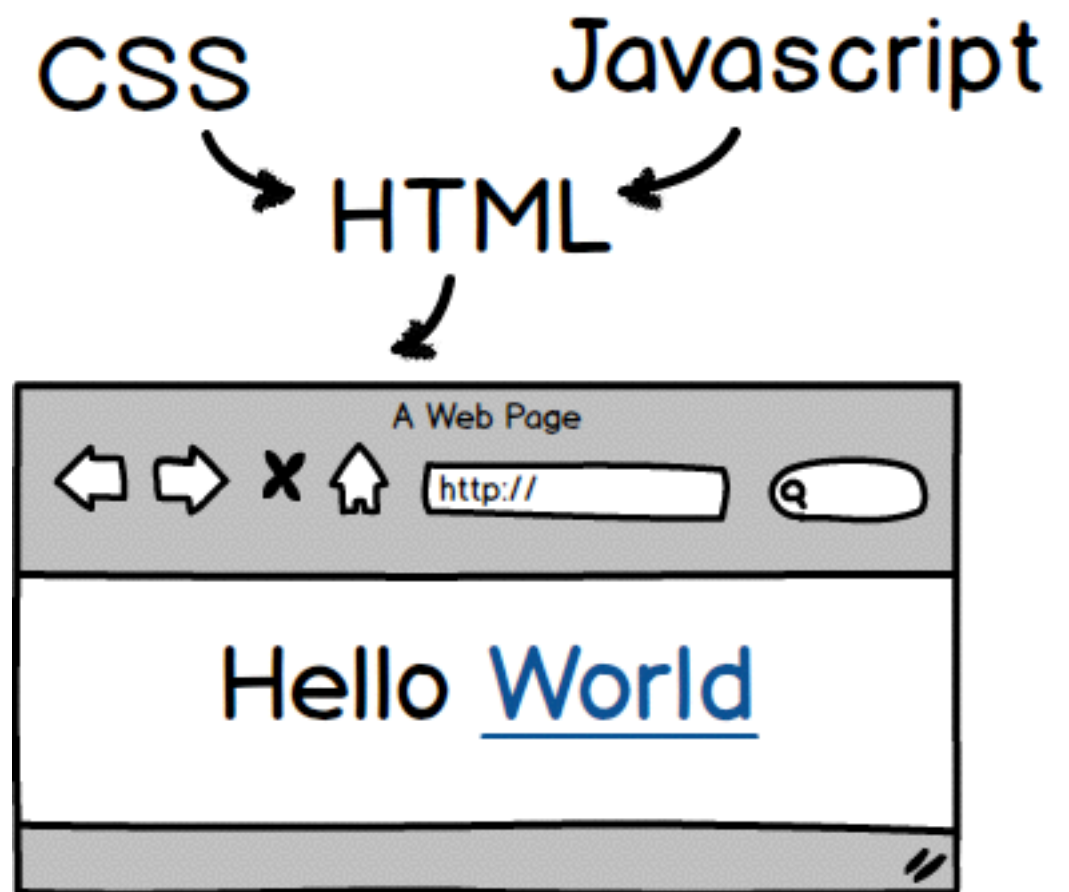
一道关于前端工程的领域问题



前端是一种特殊的GUI软件

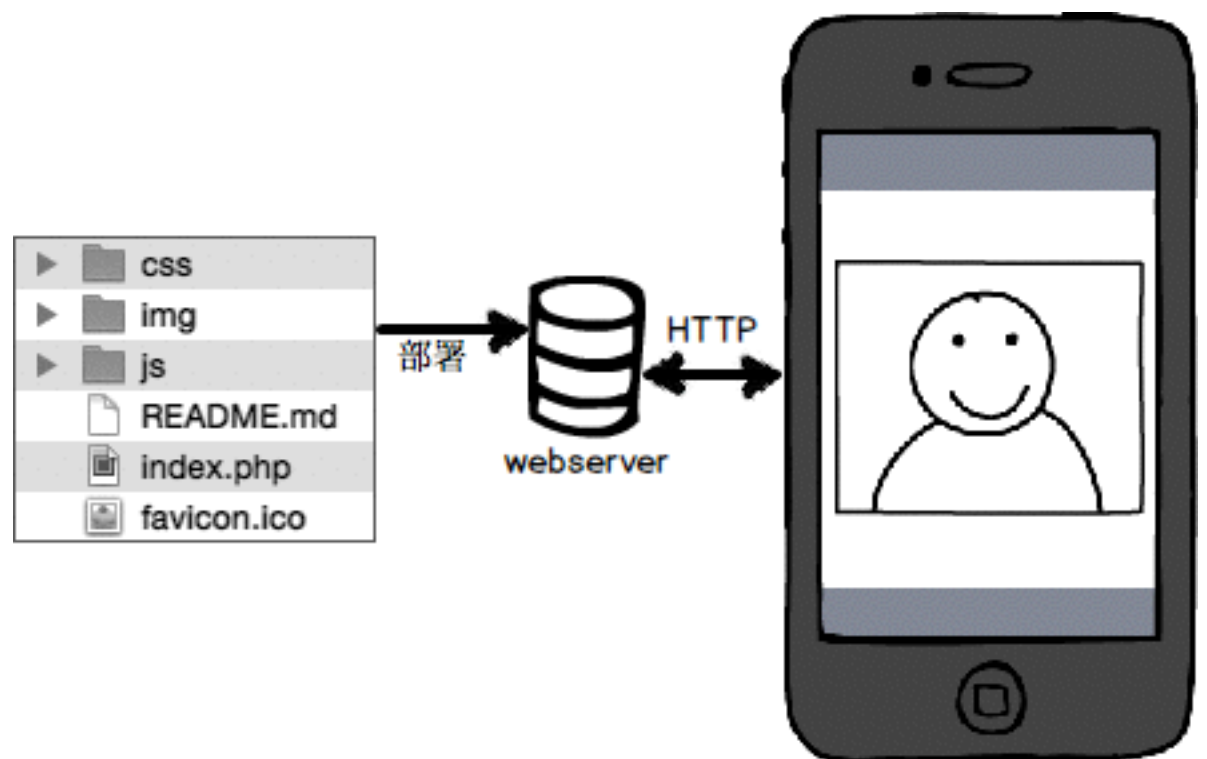
前端是一种特殊的GUI软件

- 特殊性一：由三种风格迥异的语言开发而成



前端是一种特殊的GUI软件

- 特殊性一：由三种风格迥异的语言开发而成
- 特殊性二：远程部署，运行时增量安装



什么是前端工程？

前端工程是一门研究用工程化方法构建和维护有效的、实用的和高质量的前端应用的学科。





一个记事本，一个浏览器，就可以进行前端开发了，这么简单的事，有必要搞得这么复杂吗？

在2010年的Velocity China大会上， 来自Facebook的David Wei博士分享了一些数据

- Facebook整站有**10000+**个静态资源；
- 每个静态资源都有可能被翻译成超过**100**种语言版本；
- 每种资源又会针对浏览器生成**3**种不同的版本；
- 要针对不同带宽的用户做**5**种不同的打包方法；
- 有**3**、**4**个不同的用户组，用于小批次体验新的产品功能；
- 还要考虑不同的送达方法，可以直接送达，或者通过iframe的方式提升资源并行加载的速度；
- 静态资源的压缩和非压缩状态可切换，用于调试和定位线上问题

5年前，Facebook整个网站各种页面状态的资源组合情况就有**300w**种之多

前端工程

组织架构

工程部署

性能优化

工具平台

开发流程

统计监控

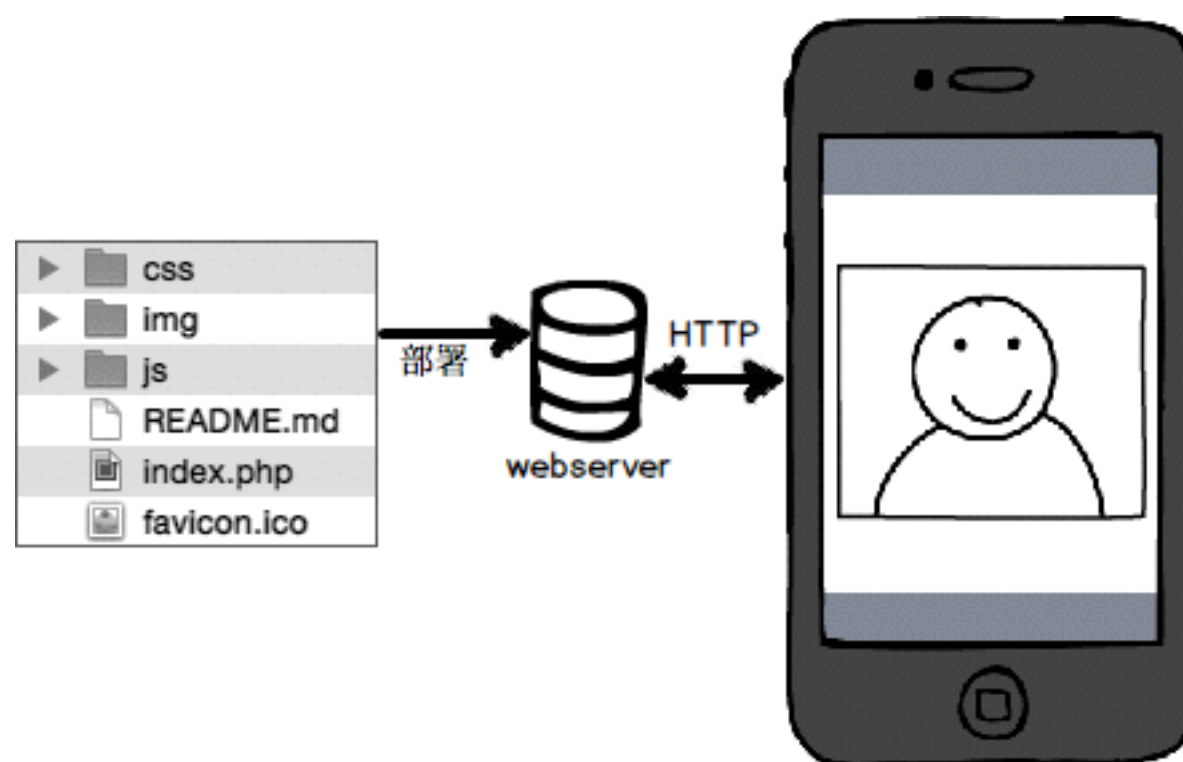
前端安全

系统测试

前端的特殊性使其在每项工程问题上都有着有别
于传统GUI软件的独特实践方式

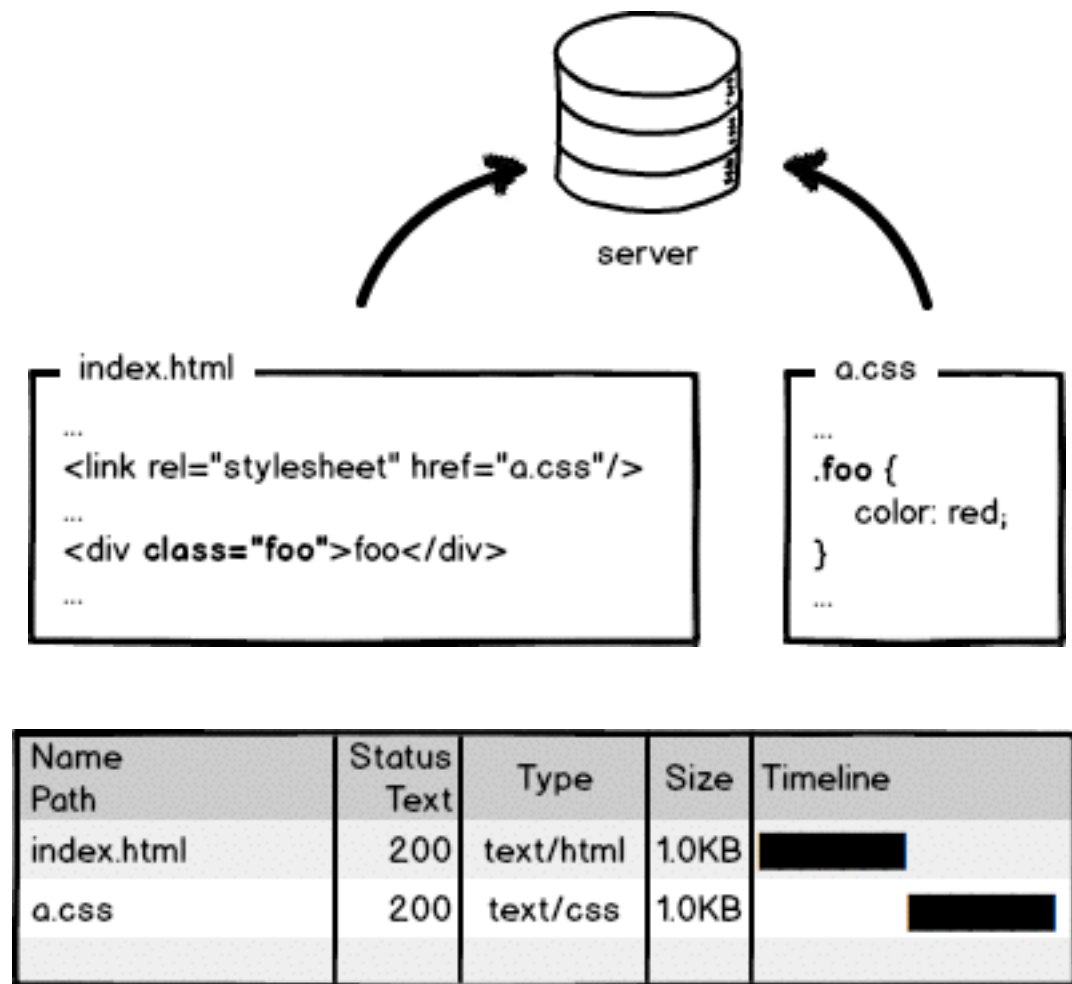
曾经在知乎上回答过一个问题：

大公司里怎样开发和部署前端代码？



大公司怎样部署前端代码

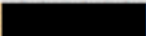

- 从一个简单的页面说起



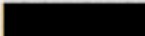

大公司怎样部署前端代码

- 从一个简单的页面说起
- 充分利用浏览器缓存

协商缓存

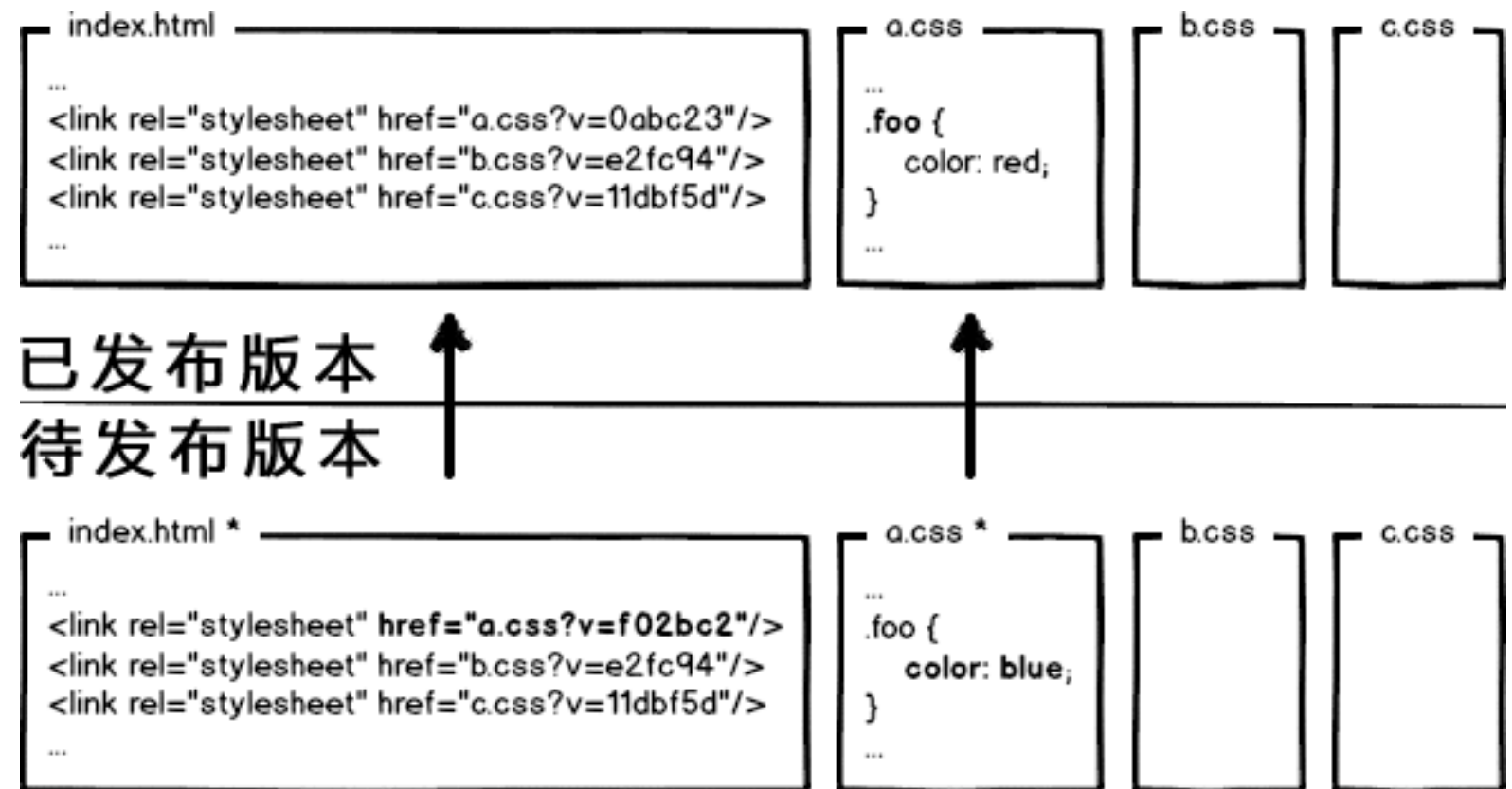
Name Path	Status Text	Type	Size	Timeline
index.html	200	text/html	1.0KB	
a.css	304	text/css	280B	

本地缓存

Name Path	Status Text	Type	Size	Timeline
index.html	200	text/html	1.0KB	
a.css	200	text/css	(from cache)	

大公司怎样部署前端代码

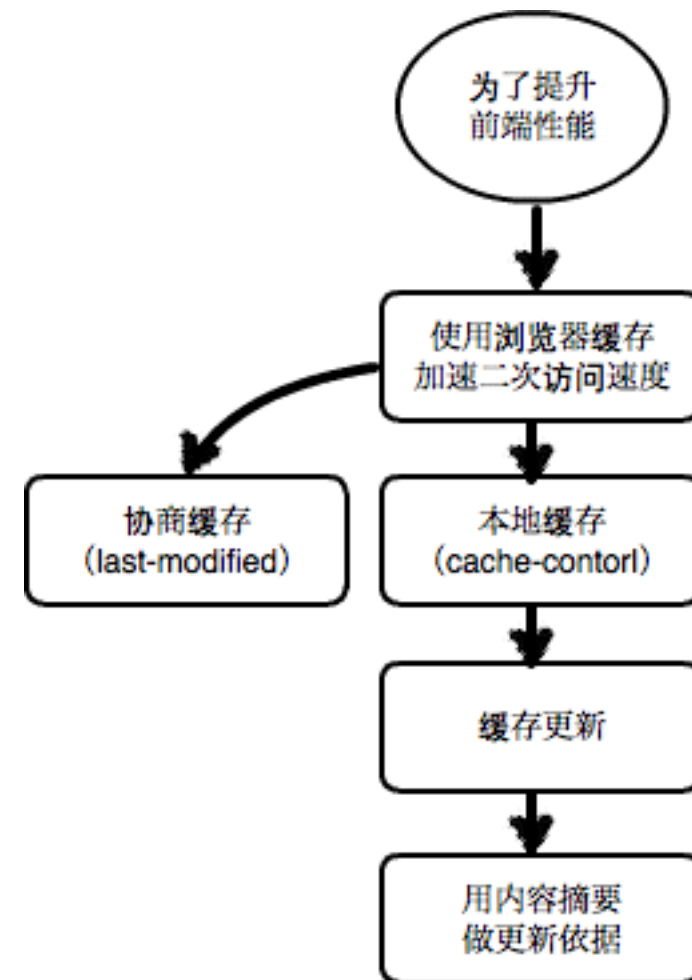
- 从一个简单的页面说起
- 充分利用浏览器缓存
- 精确的缓存更新机制



使用内容摘要（MD5）作为版本戳

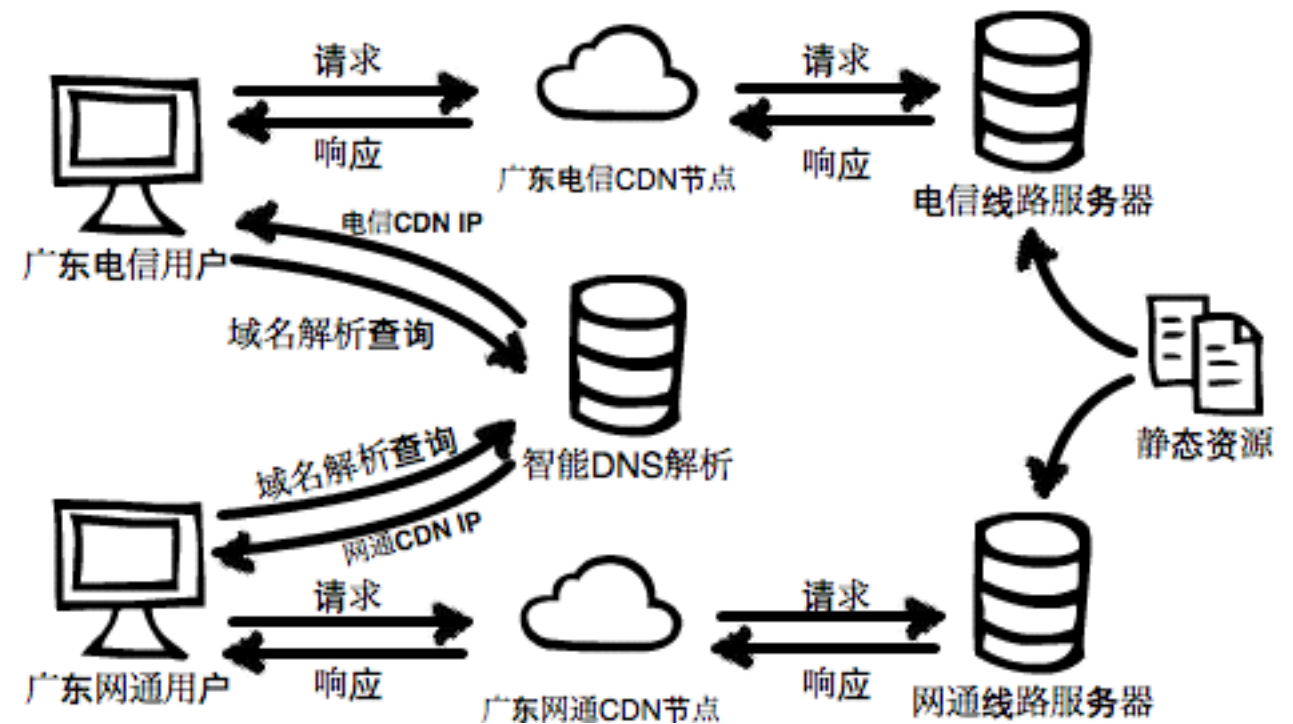
大公司怎样部署前端代码

- 从一个简单的页面说起
- 充分利用浏览器缓存
- 精确的缓存更新机制



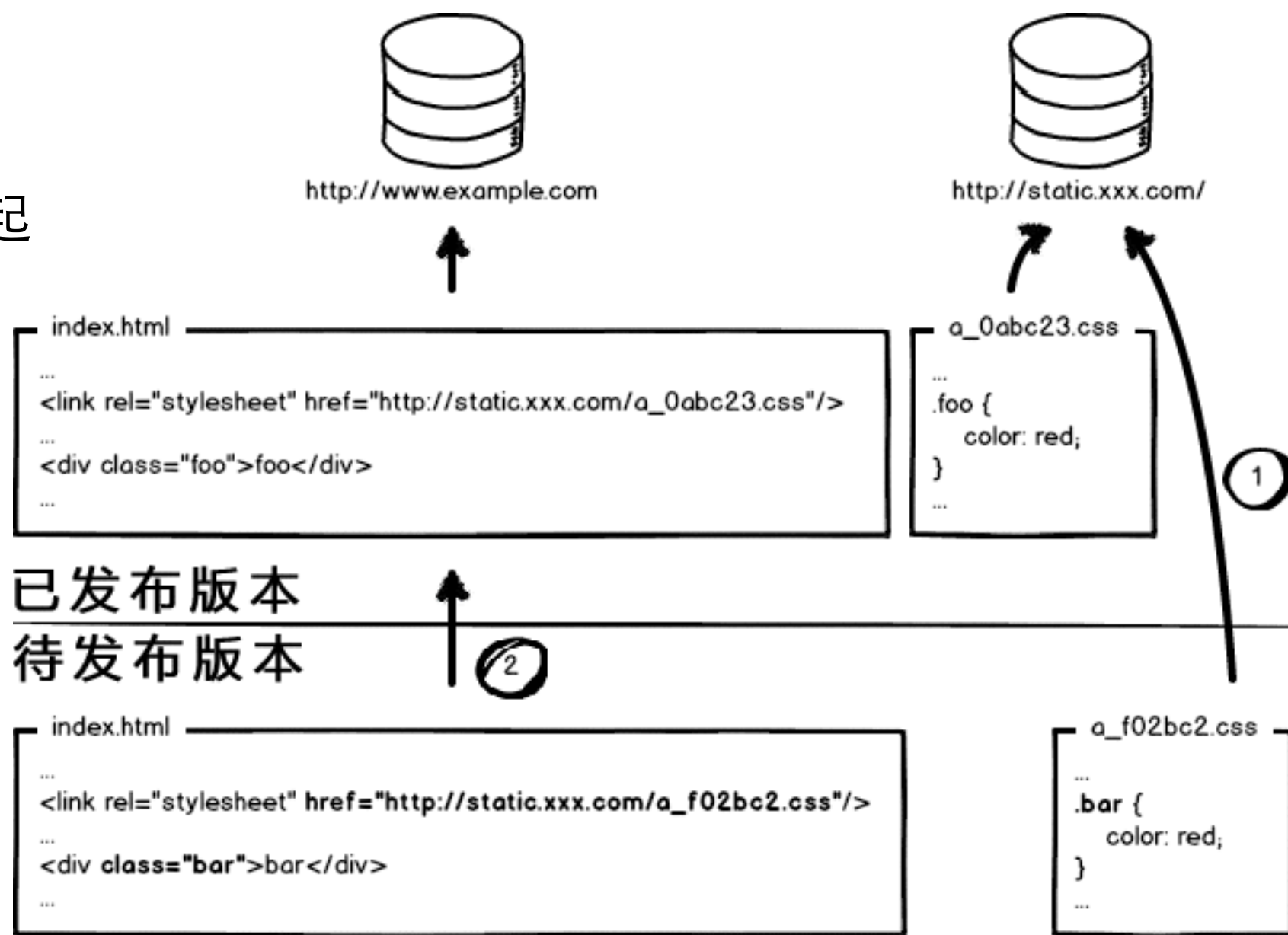
大公司怎样部署前端代码

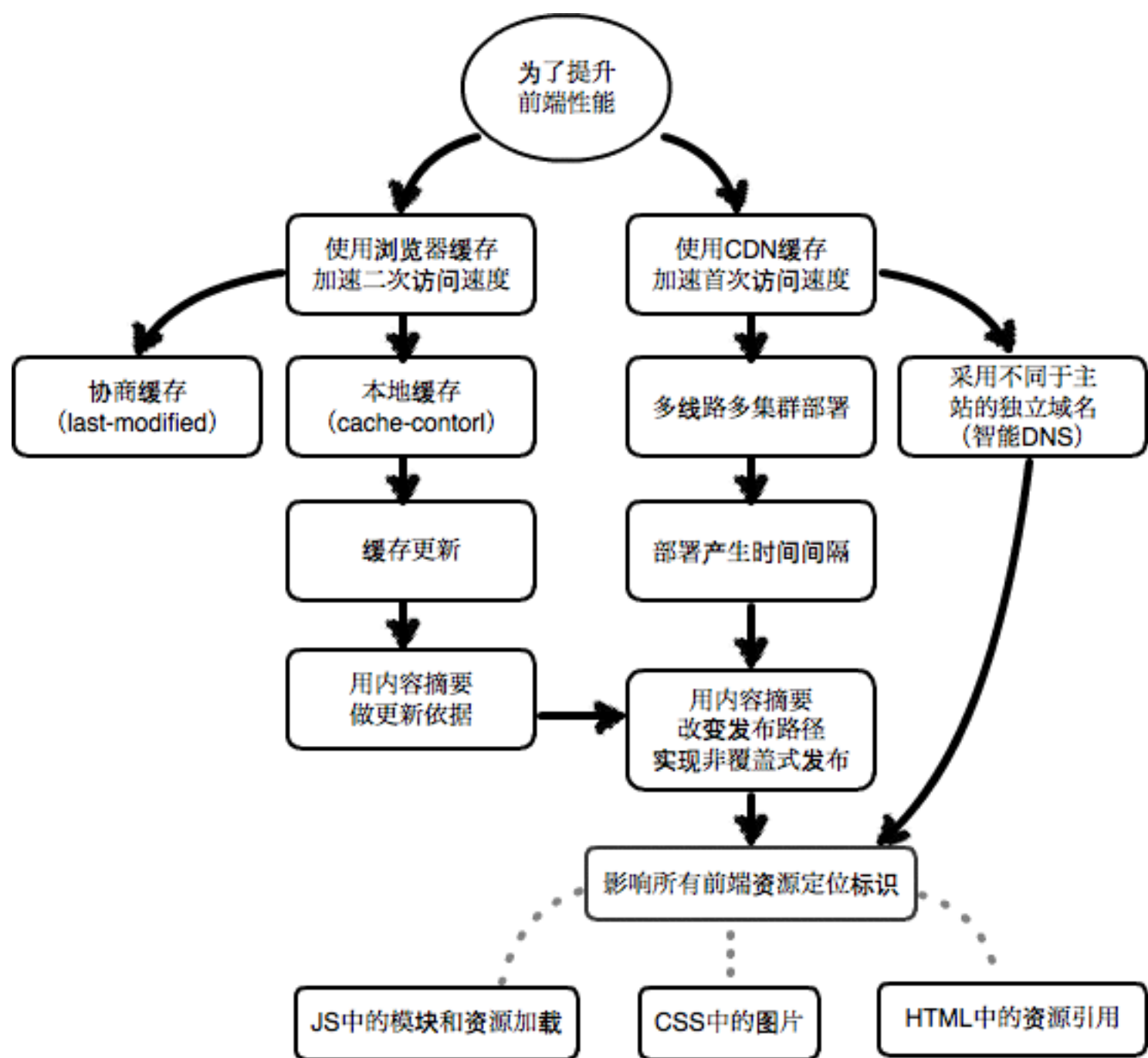
- 从一个简单的页面说起
- 充分利用浏览器缓存
- 精确的缓存更新机制
- CDN网络缓存



大公司怎样部署前端代码

- 从一个简单的页面说起
- 充分利用浏览器缓存
- 精确的缓存更新机制
- CDN网络缓存
- 非覆盖式发布





缓存控制对前端工程的影响

资源打包，就是把所有资源合并成一个请求加载么？

一个小故事

——2010年的Velocity China大会，David Wei博士

```
<html>
...
<link rel="stylesheet" href="A.css">
<link rel="stylesheet" href="B.css">
<link rel="stylesheet" href="C.css">
...
<div>A组件的HTML内容</div>
<div>B组件的HTML内容</div>
<div>C组件的HTML内容</div>
...
</html>
```

第一天



```
<html>
...
<link rel="stylesheet" href="A-B-C.css">
...
<div>A组件的HTML内容</div>
<div>B组件的HTML内容</div>
<div>C组件的HTML内容</div>
...
</html>
```

第二天



```
<html>
...
<link rel="stylesheet" href="A-B-C.css">
...
<div>A组件的HTML内容</div>
<div>B组件的HTML内容</div>
<div>
  {if $use_C}
    <div>C组件的HTML内容</div>
  {/if}
</div>
...
</html>
```

第三天



差不多一个意思

```
<html>
...
<link rel="stylesheet" href="A-B-C.css">
...
<div>A组件的HTML内容</div>
<div>B组件的HTML内容</div>
<div>
  {*if $use_C}
    <div>C组件的HTML内容</div>
  {/if*}
</div>
...
</html>
```

第四天




一个小故事

——2010年的Velocity China大会，David Wei博士

```
<html>
...
<link rel="stylesheet" href="A-B-C-D-E-F-G-H....css">
...
{if $use_E}
  <div>E组件的HTML内容</div>
{else}
  <div>F组件的HTML内容</div>
  <div>G组件的HTML内容</div>
{/if}
...
</html>
```

后来.....



资源管理的挑战

- 就近引用原则

```
<html>
```

```
...
```

```
<link rel="stylesheet" href="A.css">
```

```
<link rel="stylesheet" href="B.css">
```

```
<link rel="stylesheet" href="C.css">
```

```
...
```

```
<div>A组件的HTML内容</div>
```

```
<div>B组件的HTML内容</div>
```

```
<div>C组件的HTML内容</div>
```

```
...
```

```
</html>
```

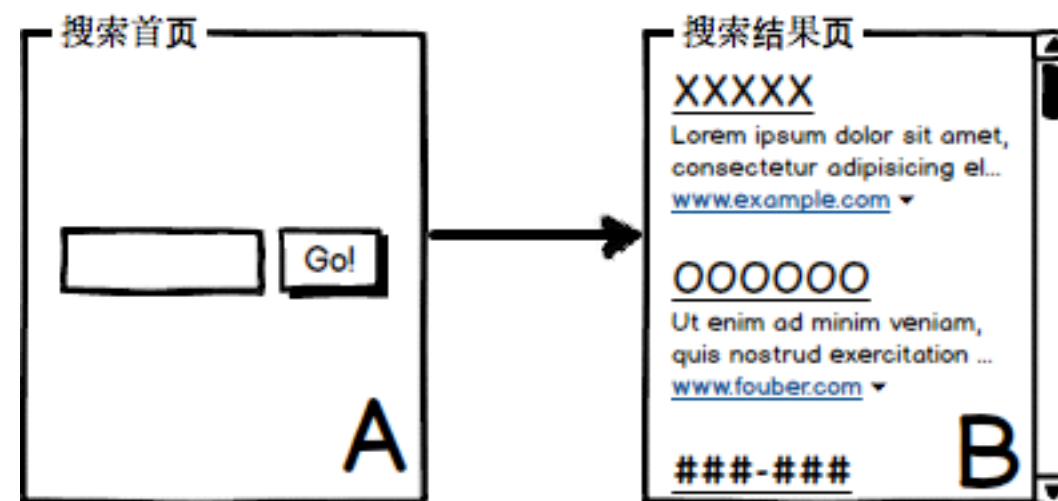
第一天



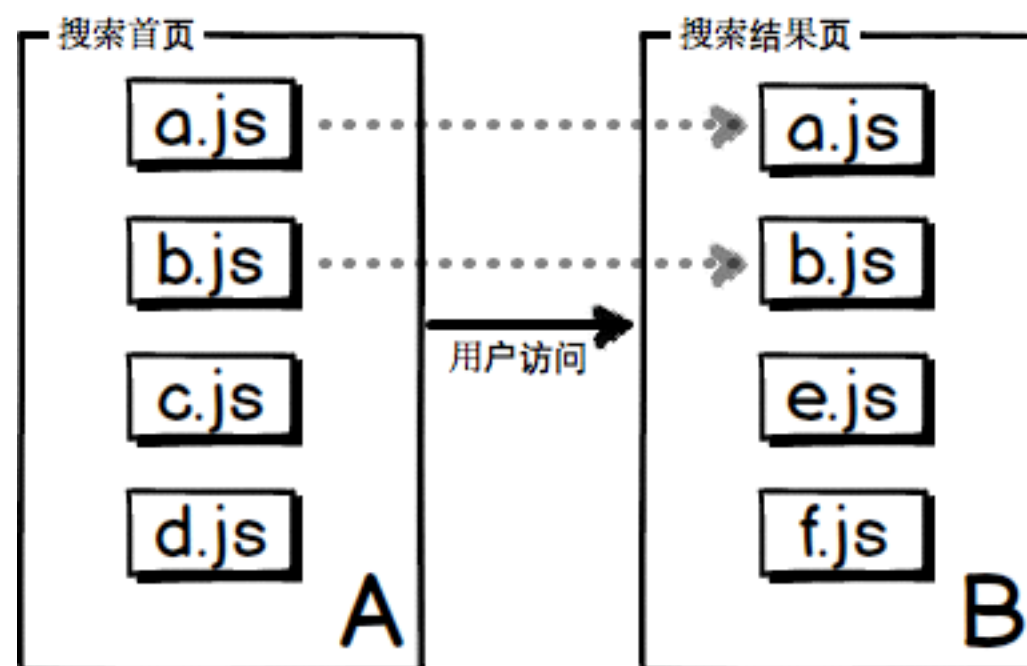
罪魁祸首——资源的使用与引用声明分开维护

资源管理的挑战

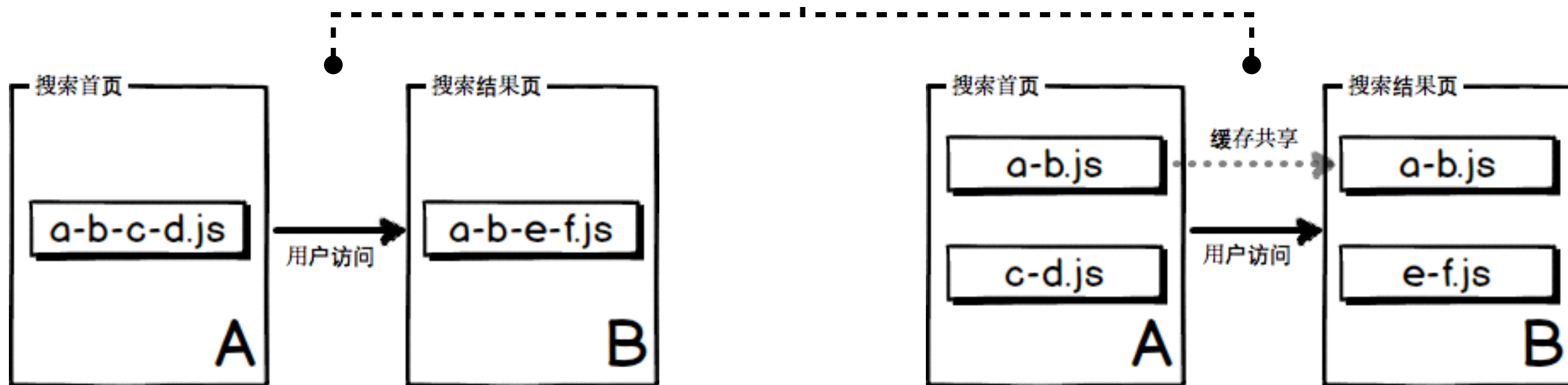
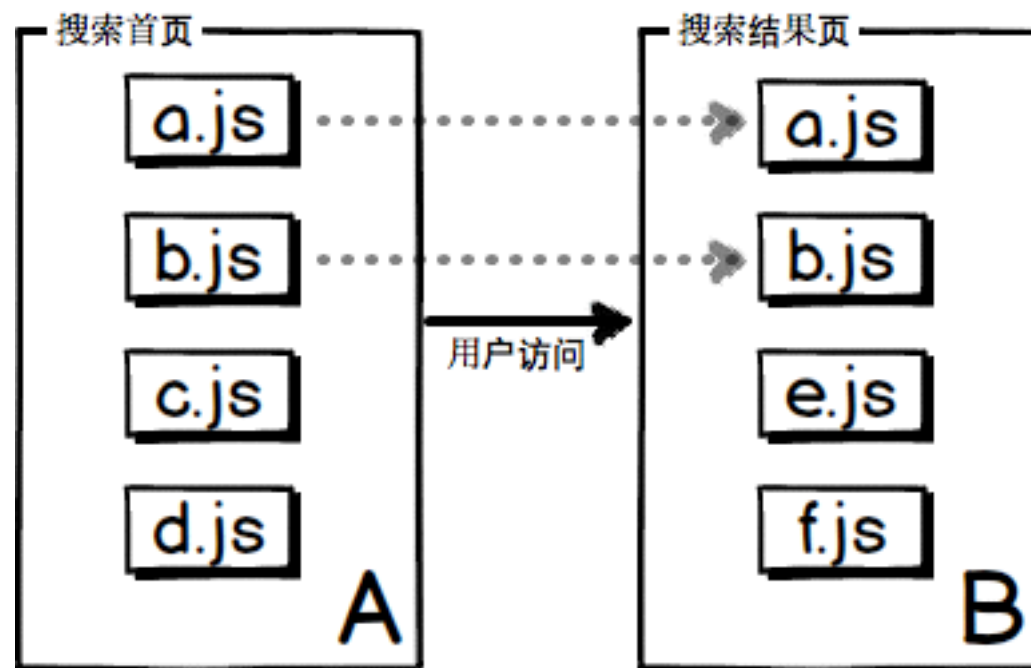
- 就近引用原则
- 页面间缓存共享



跨页面资源复用



资源管理的挑战



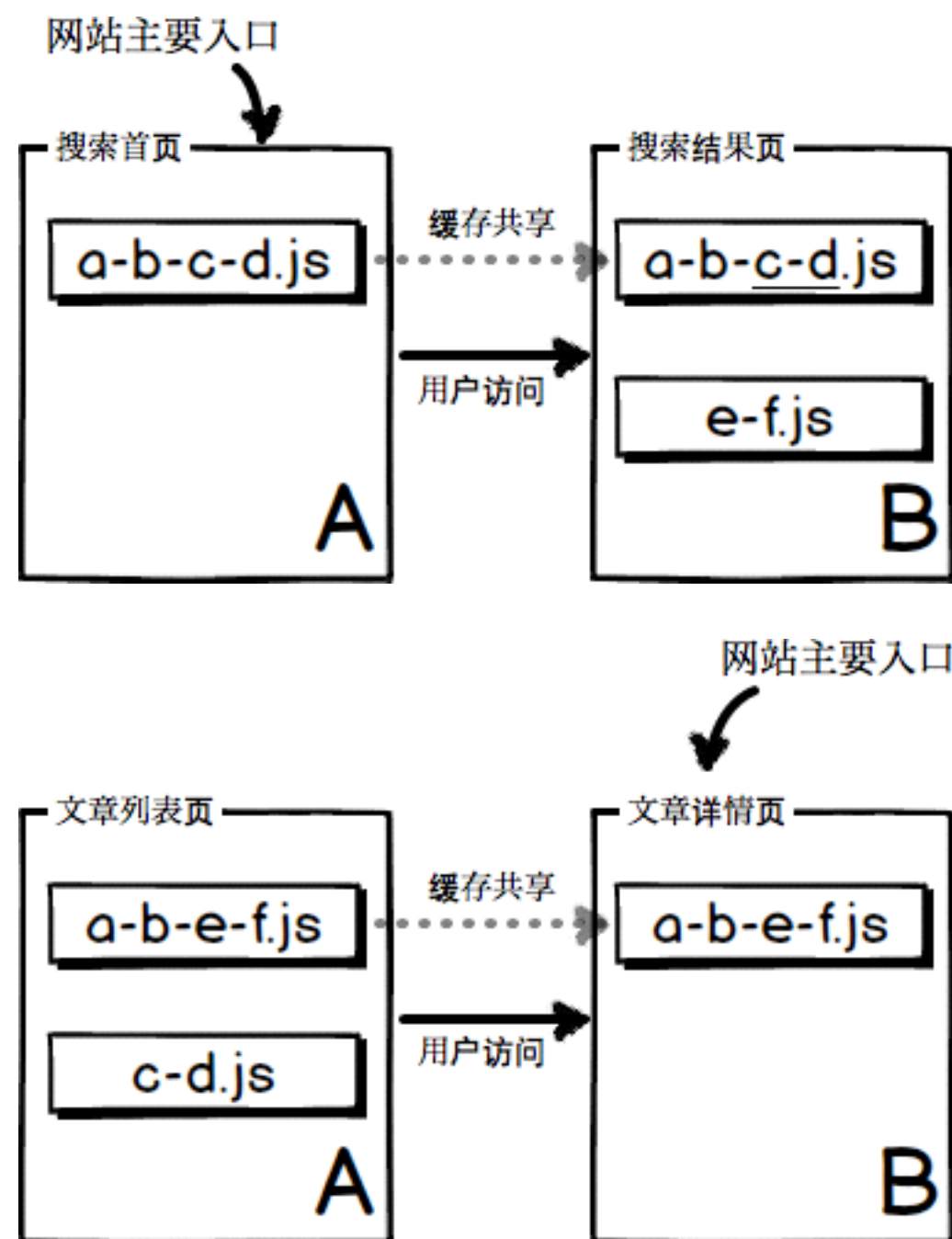
all-in-one

基于页面共享的资源合并



资源管理的挑战

- 就近引用原则
- 页面间缓存共享



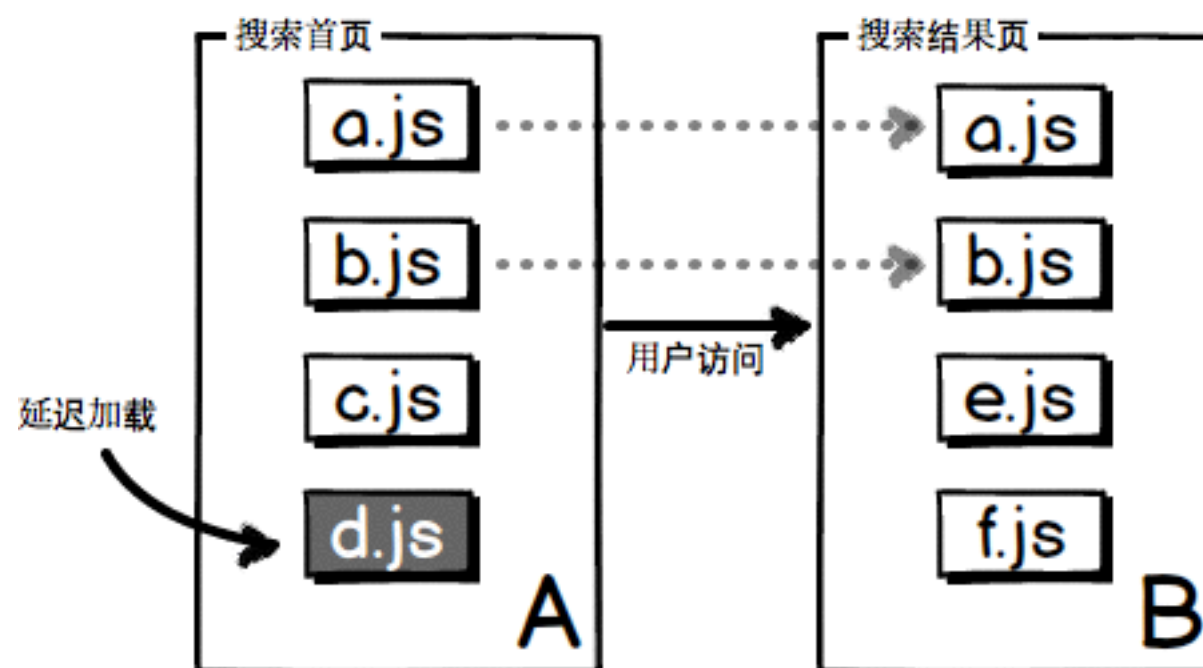
资源管理的挑战

- 就近引用原则
- 页面间缓存共享
- 缓存失效率

$$P = 1 - (1 - P_0) \times (1 - P_1) \times (1 - P_2) \times \dots \times (1 - P_n)$$

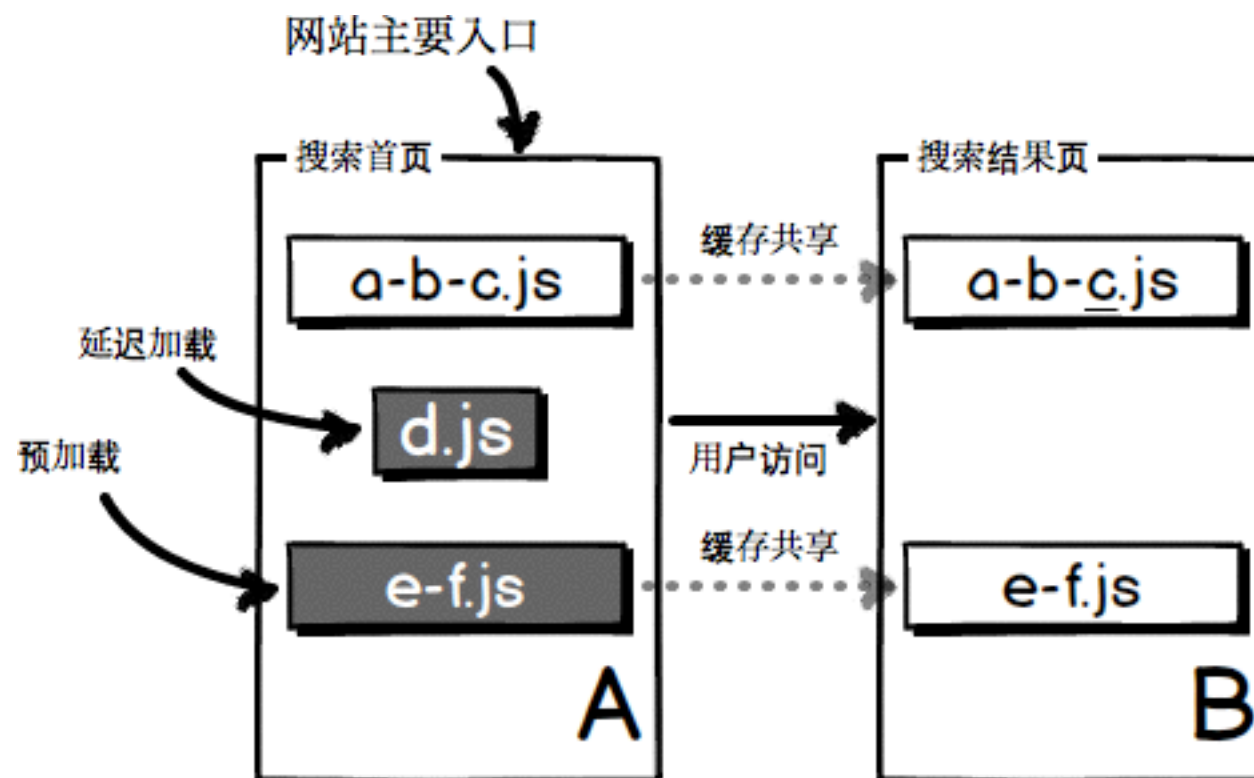
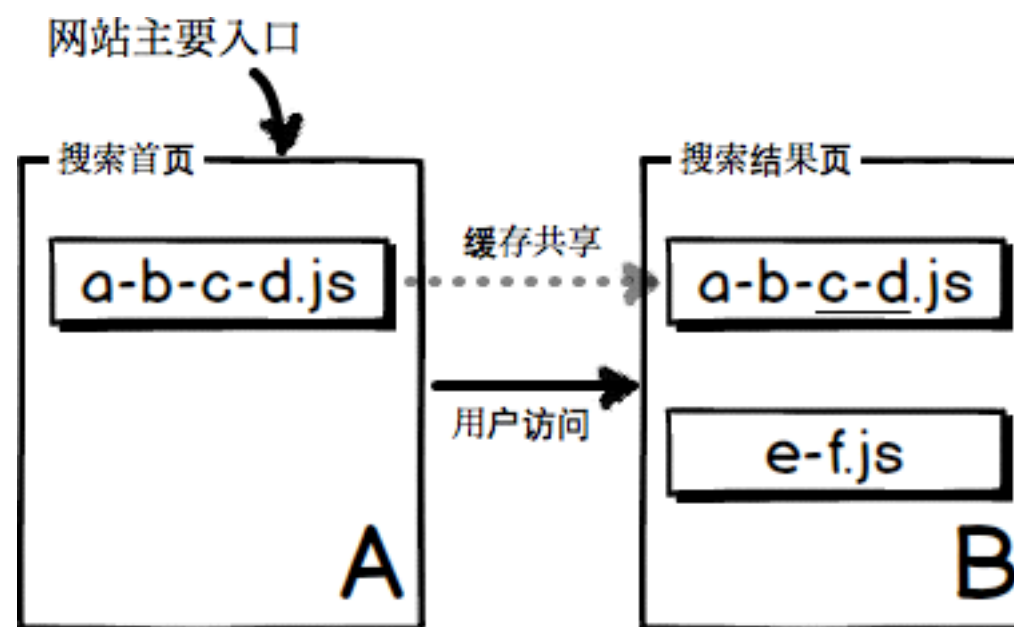
资源管理的挑战

- 就近引用原则
- 页面间缓存共享
- 缓存失效率
- 延迟加载与预加载



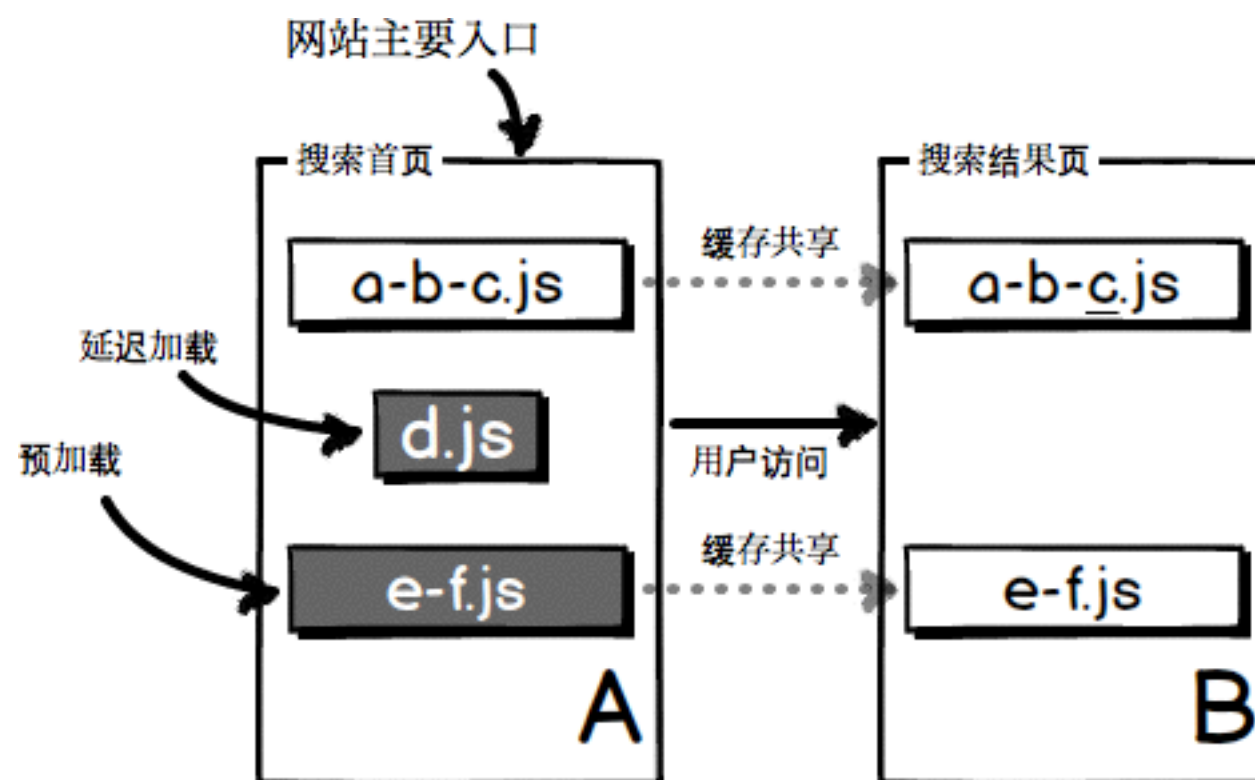
资源管理的挑战

- 就近引用原则
- 页面间缓存共享
- 缓存失效率
- 延迟加载与预加载



资源管理的挑战

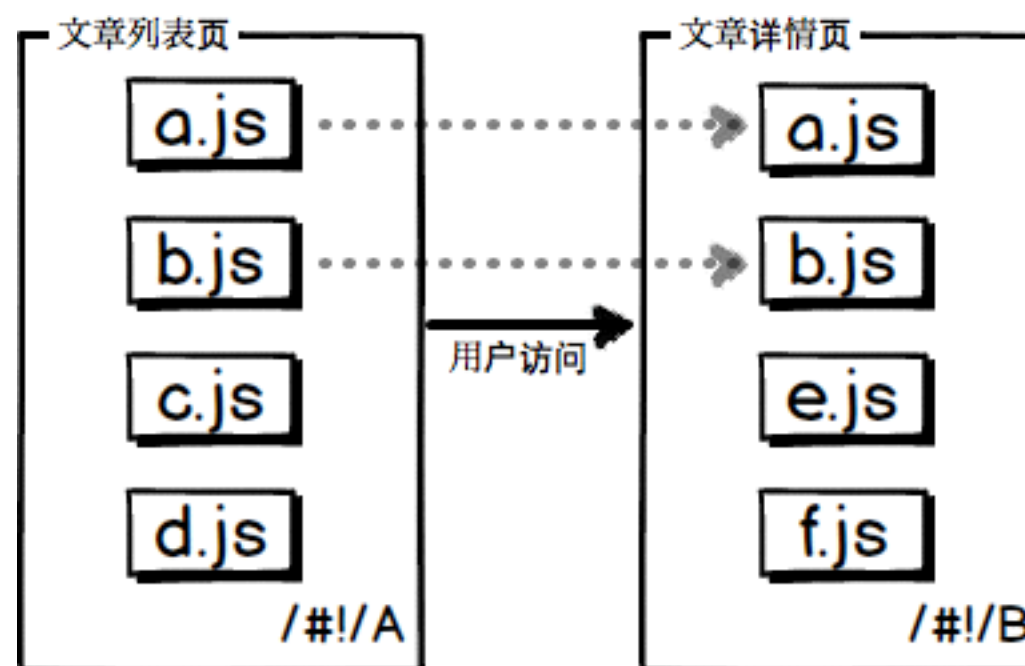
- 就近引用原则
- 页面间缓存共享
- 缓存失效率
- 延迟加载与预加载



```
...  
<script src="a-b-c.js"></script> <!--初始加载 a-b-c.js-->  
<script>  
    load('d.js', function(){ //延迟加载 d.js  
        preload('e-f.js'); //预加载 e-f.js  
    });  
</script>  
...
```

资源管理的挑战

- 就近引用原则
- 页面间缓存共享
- 缓存失效率
- 延迟加载与预加载
- 单页面应用

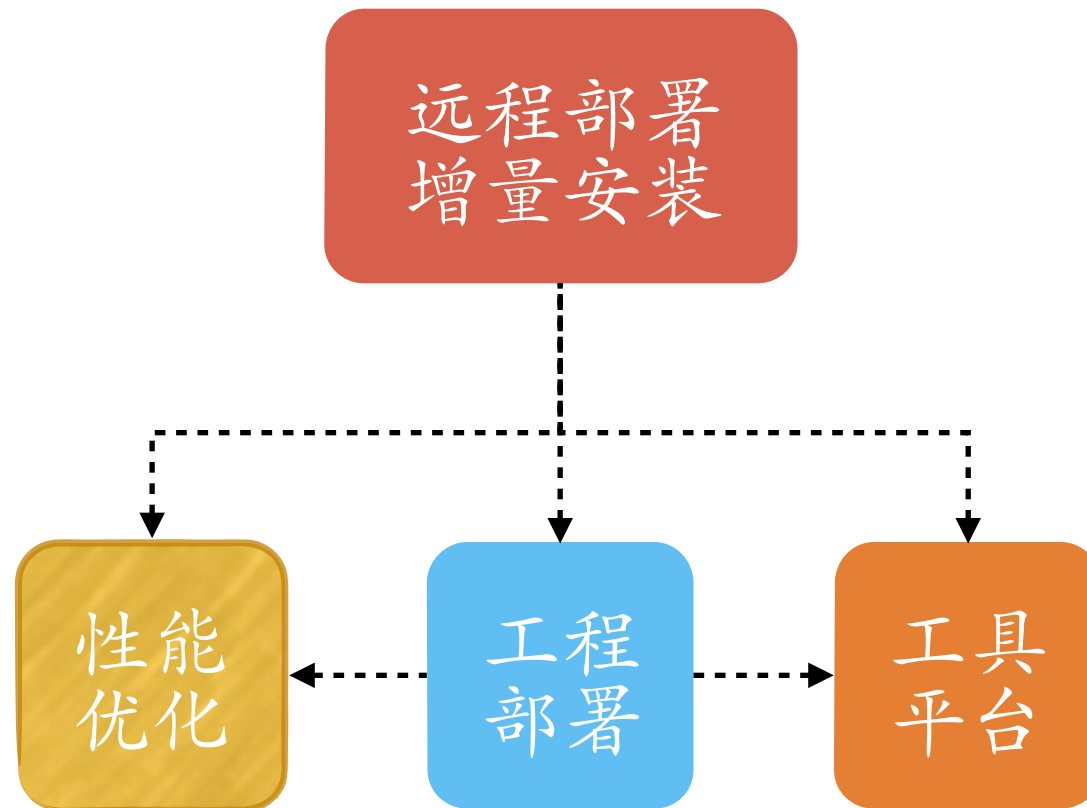


资源管理的挑战

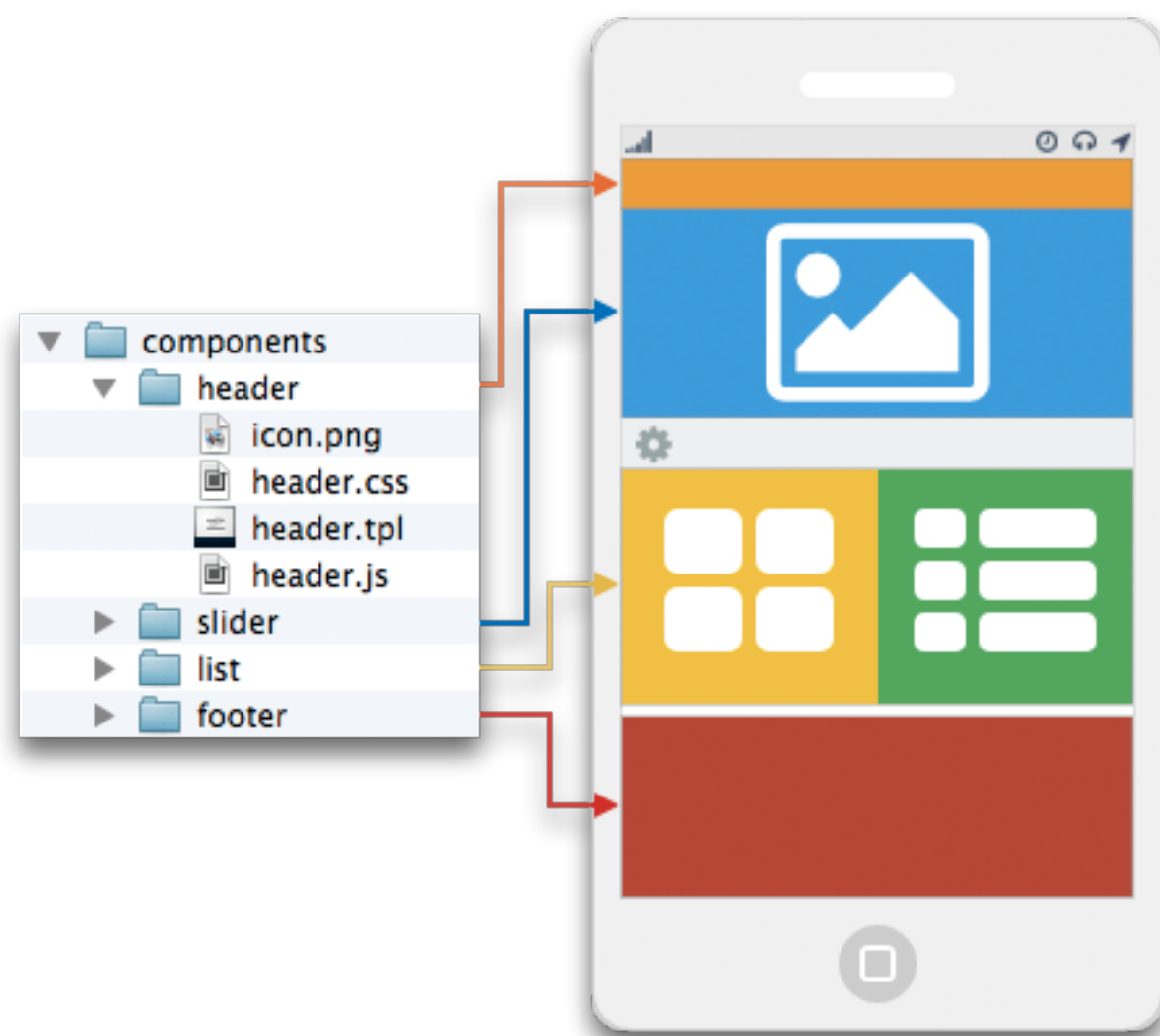
- 就近引用原则
- 页面间缓存共享
- 缓存失效率
- 延迟加载与预加载
- 单页面应用

```
<!DOCTYPE html>
<html>
  ...
  <body>
    <script src="router.js"></script>
    <script src="loader.js"></script>
    <script>
      router('/#!/A', function(){
        loader('a.js', 'b.js', 'c.js', 'd.js', function(){
          // 初始化虚拟页面A
        });
      });
      router('/#!/B', function(){
        loader('a.js', 'b.js', 'e.js', 'f.js', function(){
          // 初始化虚拟页面B
        });
      });
      router.start();
    </script>
  </body>
</html>
```


前端特殊性对工程的影响



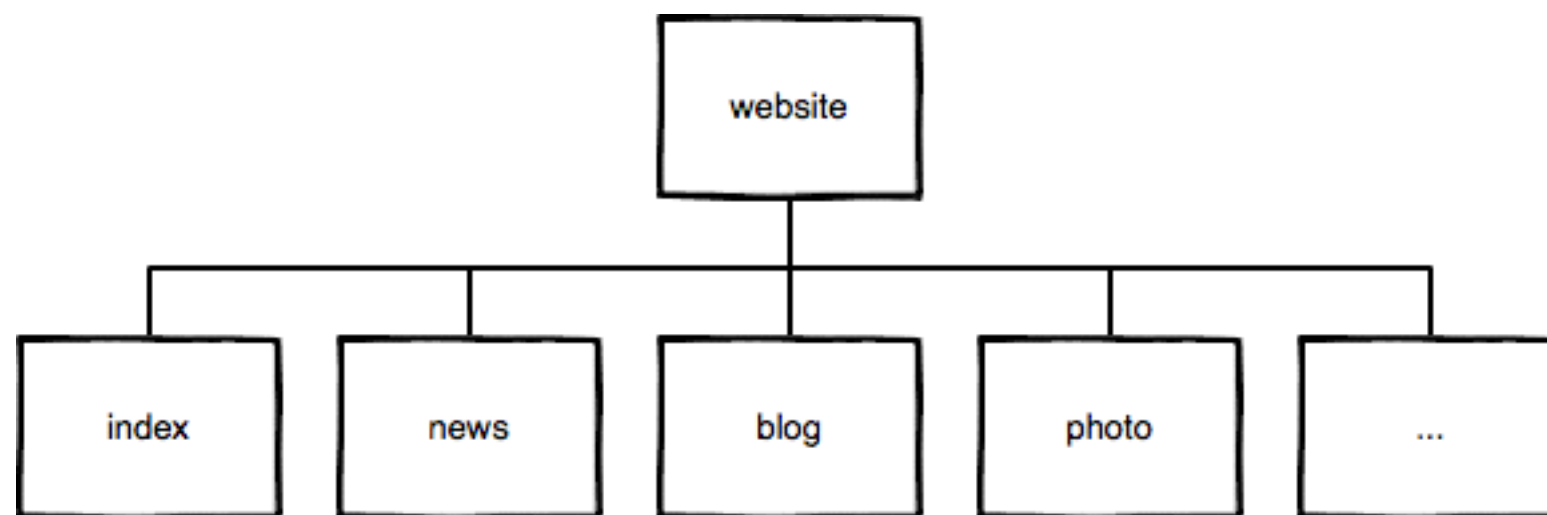
分而治之是编程领域中解决复杂问题的重要思想之一



前端工程领域的分治实践——组件化开发

分而治之的开发模式

- 站点由页面组成



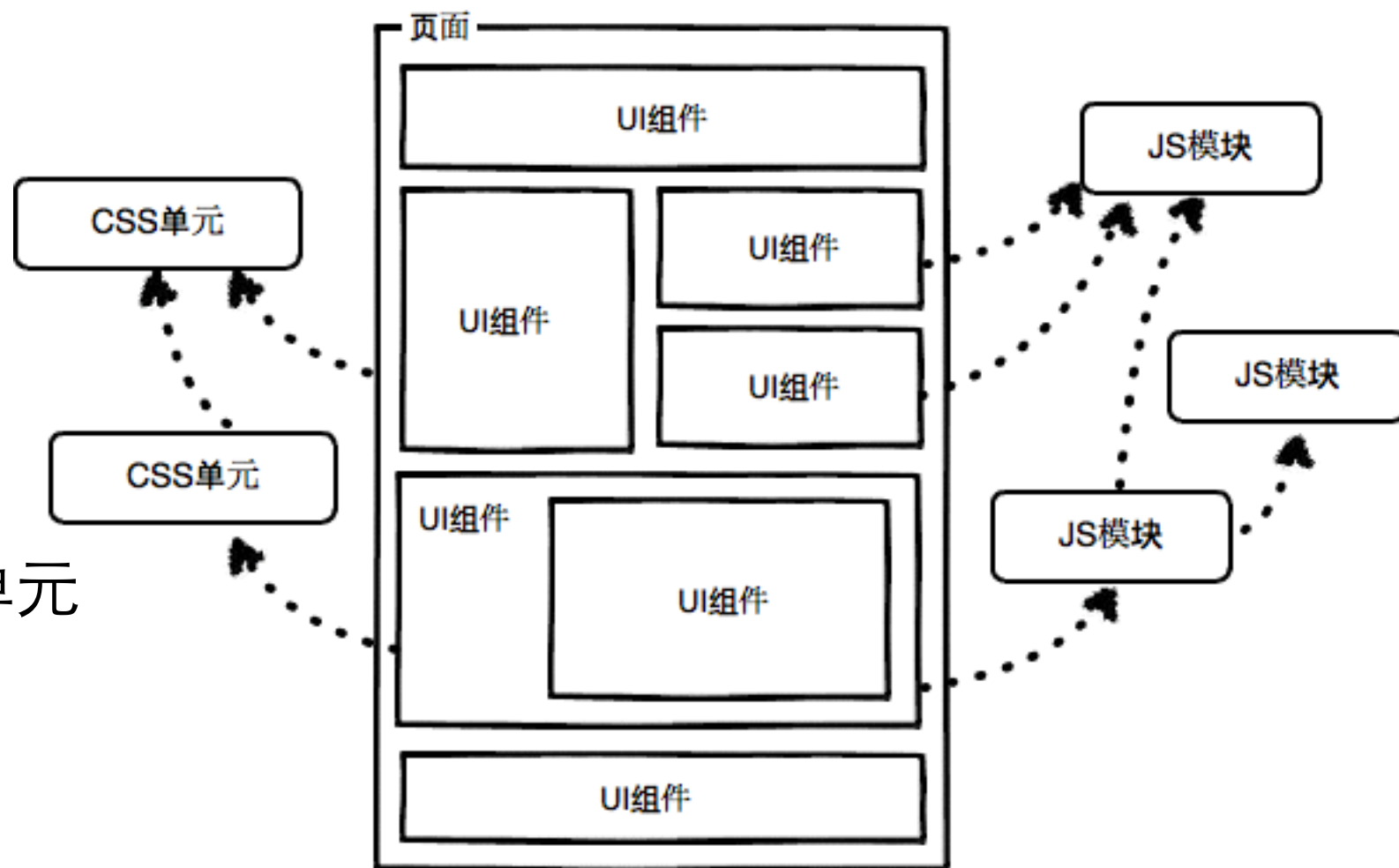
分而治之的开发模式

- 站点由页面组成
- 页面由组件组成



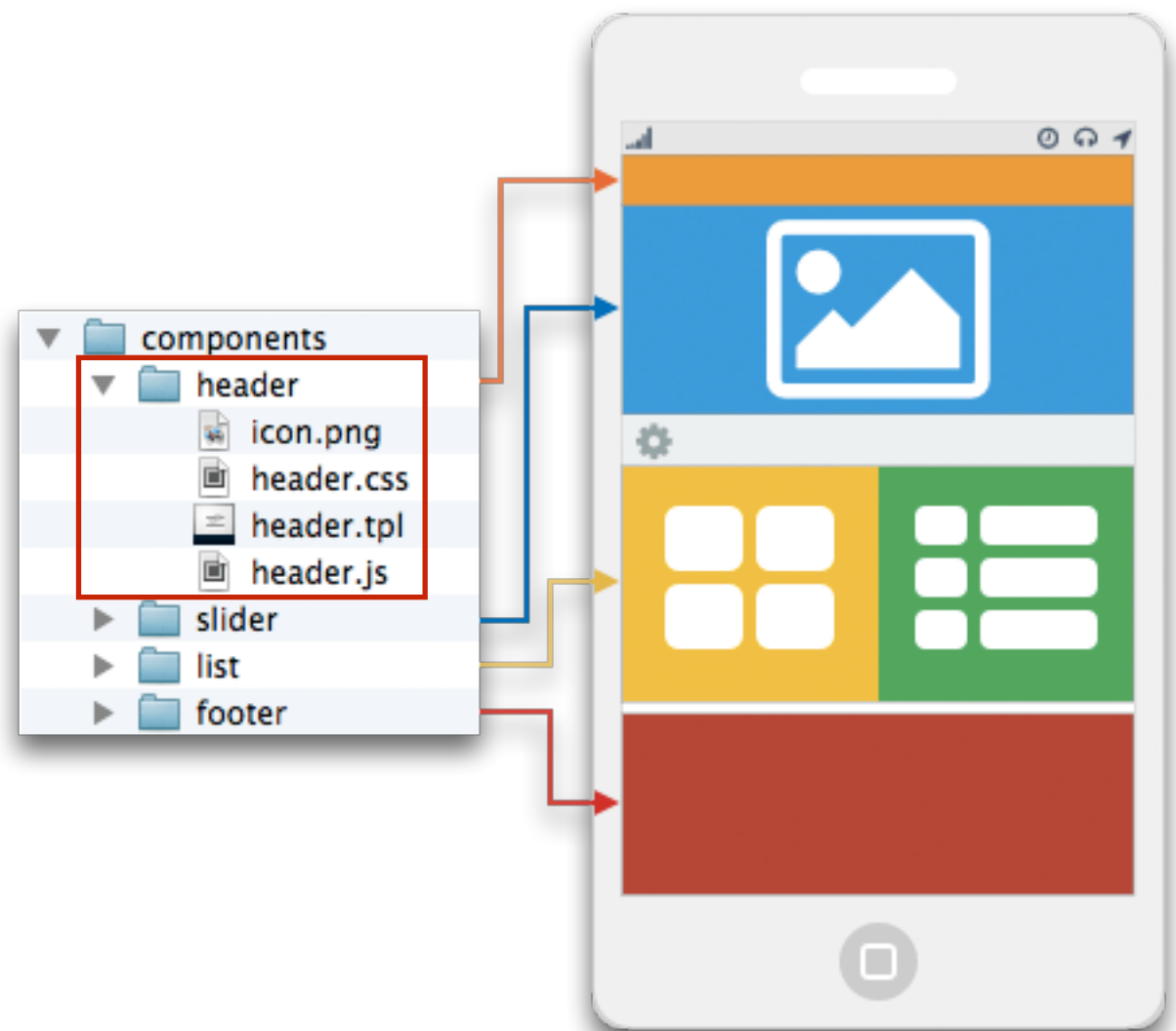
分而治之的开发模式

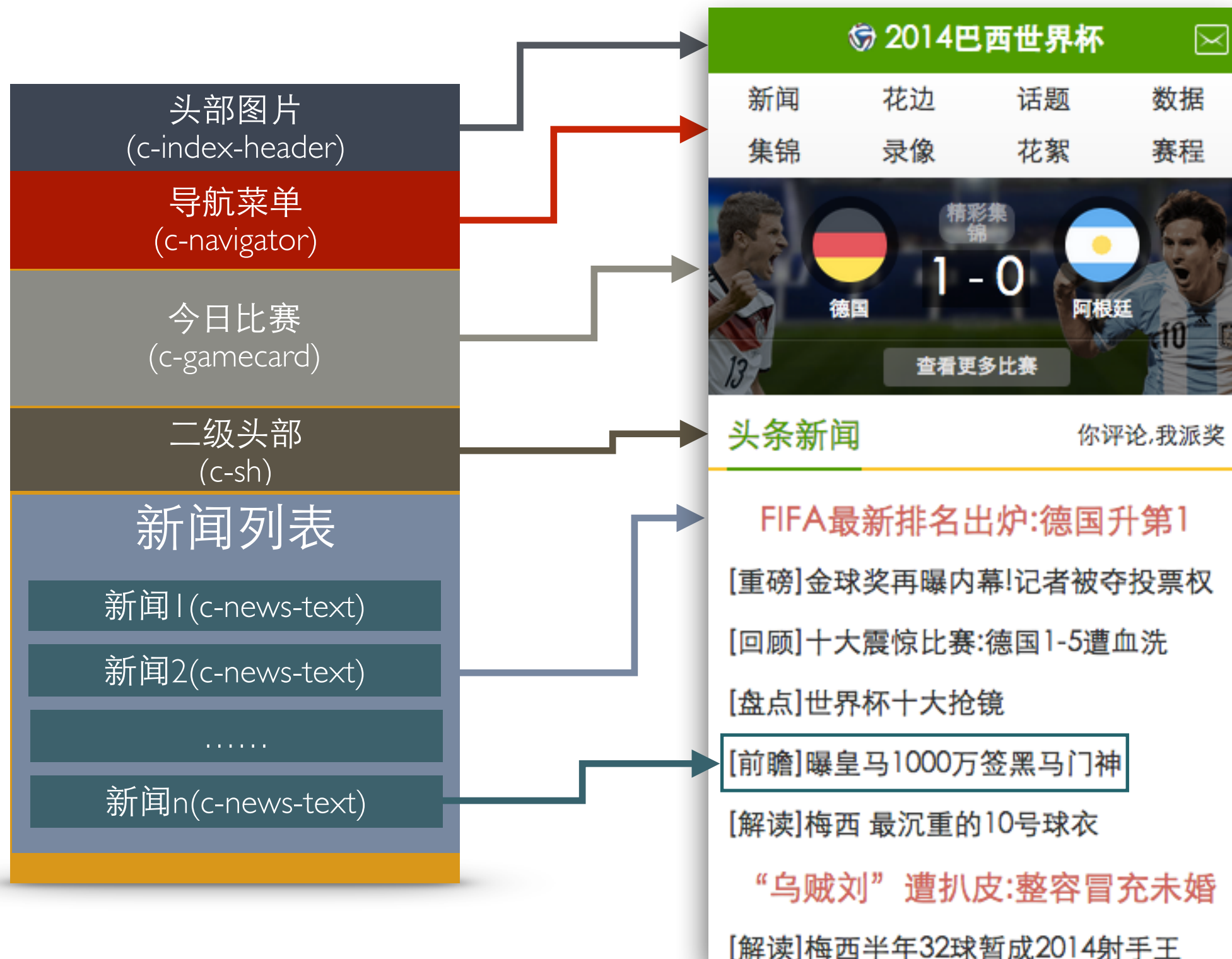
- 站点由页面组成
- 页面由组件组成
- 组件依赖JS模块和CSS单元



分而治之的开发模式

- 站点由页面组成
- 页面由组件组成
- 组件依赖JS模块和CSS单元
- 资源内聚，就近维护





分而治之让我们这样看待前端项目

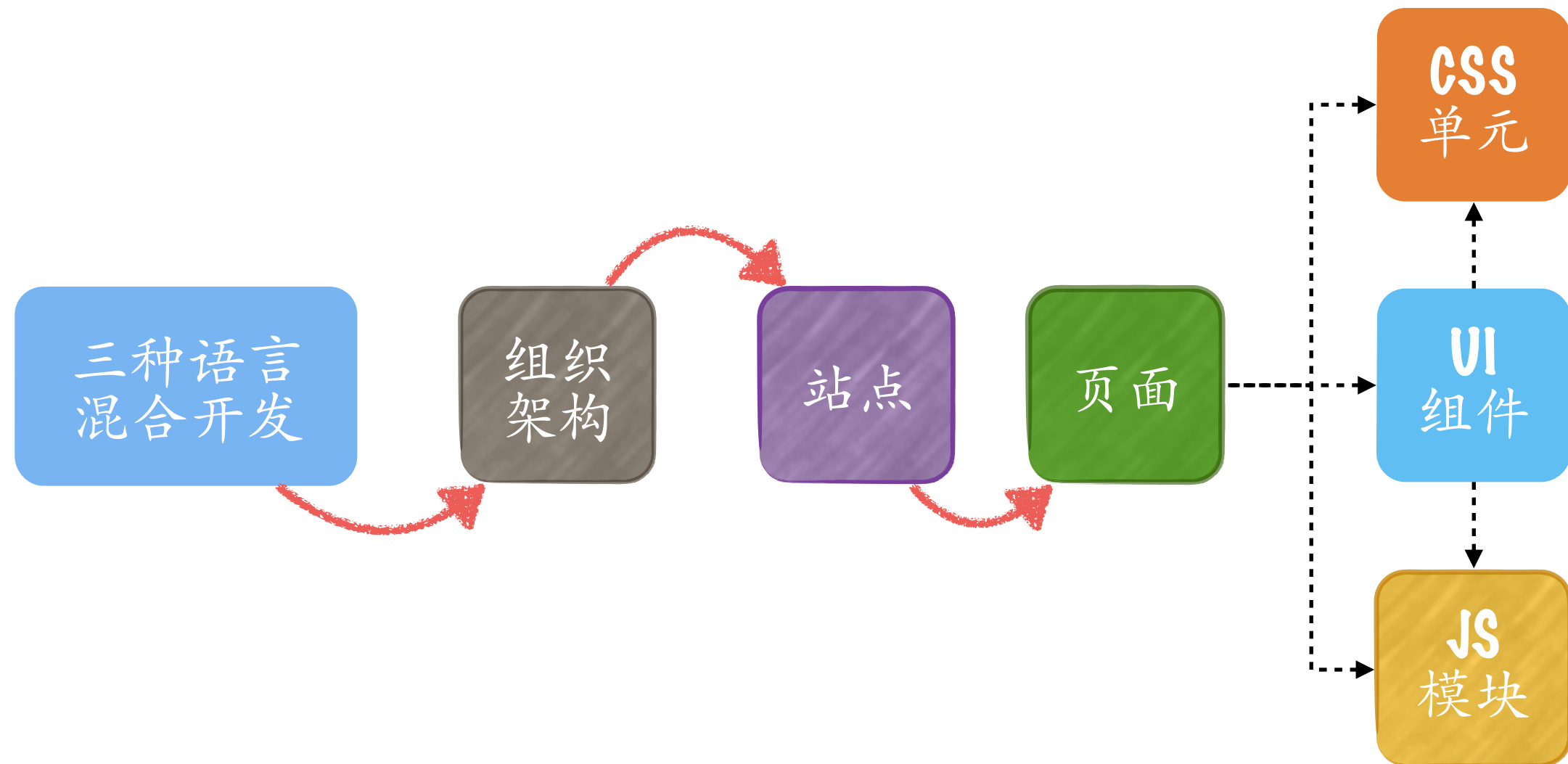
Components

component 拥有自己的组件状态行为，提供接口（数据/方法）为其它组件所用。

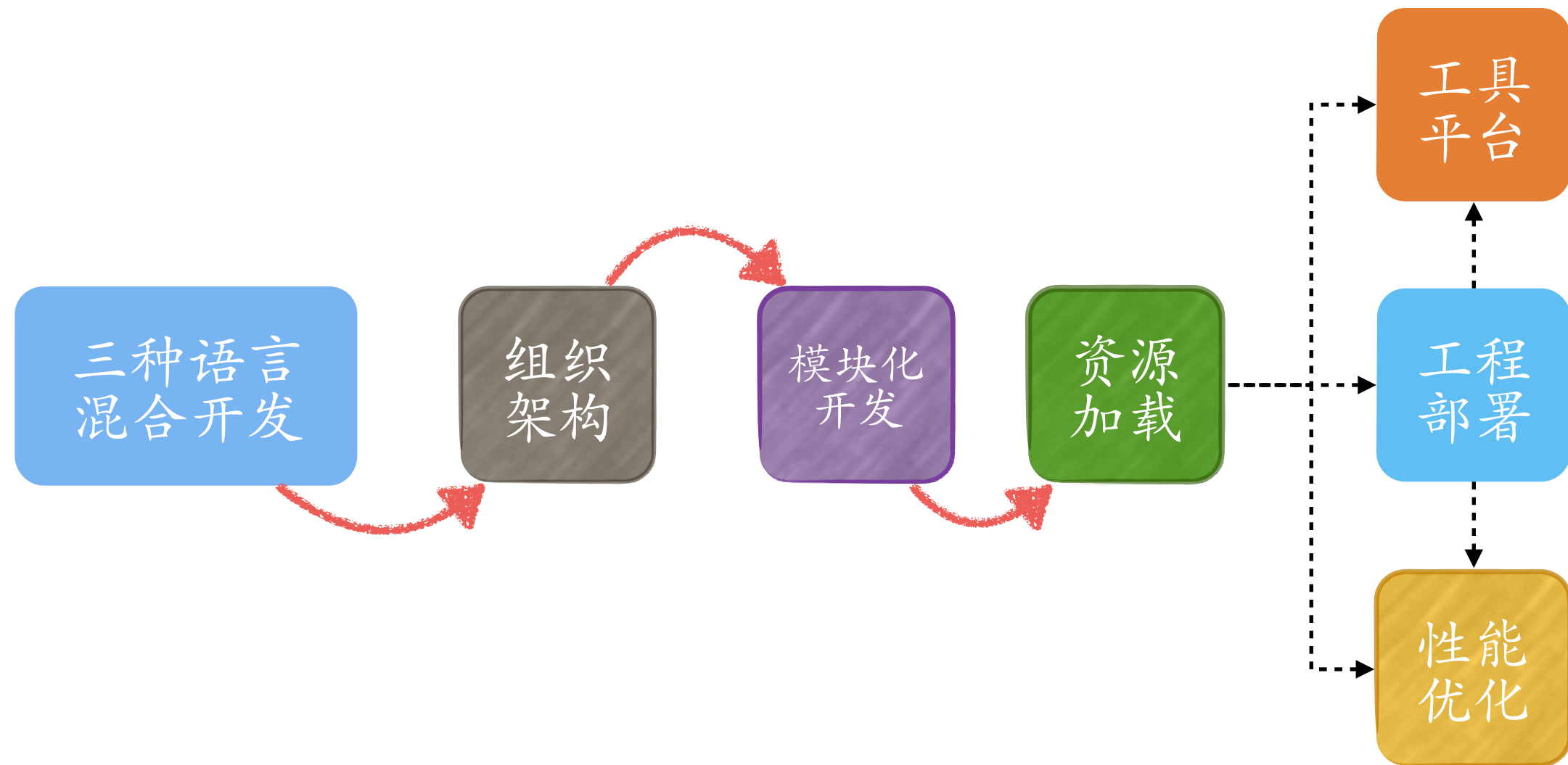
编号	ID	描述	依赖	接口	路径	UI
COMP01	c-index-header	首页头部	首页	onRoute	components/p-index	
	c-navigator	首页导航	首页	onRoute	components/c-navigator	
	c-gamecard	比赛卡片	首页	onRoute,onMore	components/c-gamecard	
	c-sh	首页二级头部	首页	onRedirect	components/c-sh	
	c-news-text	新闻列表项	首页, 资讯页	onRedirect	components/c-new-text	

分而治之让我们这样开发前端项目

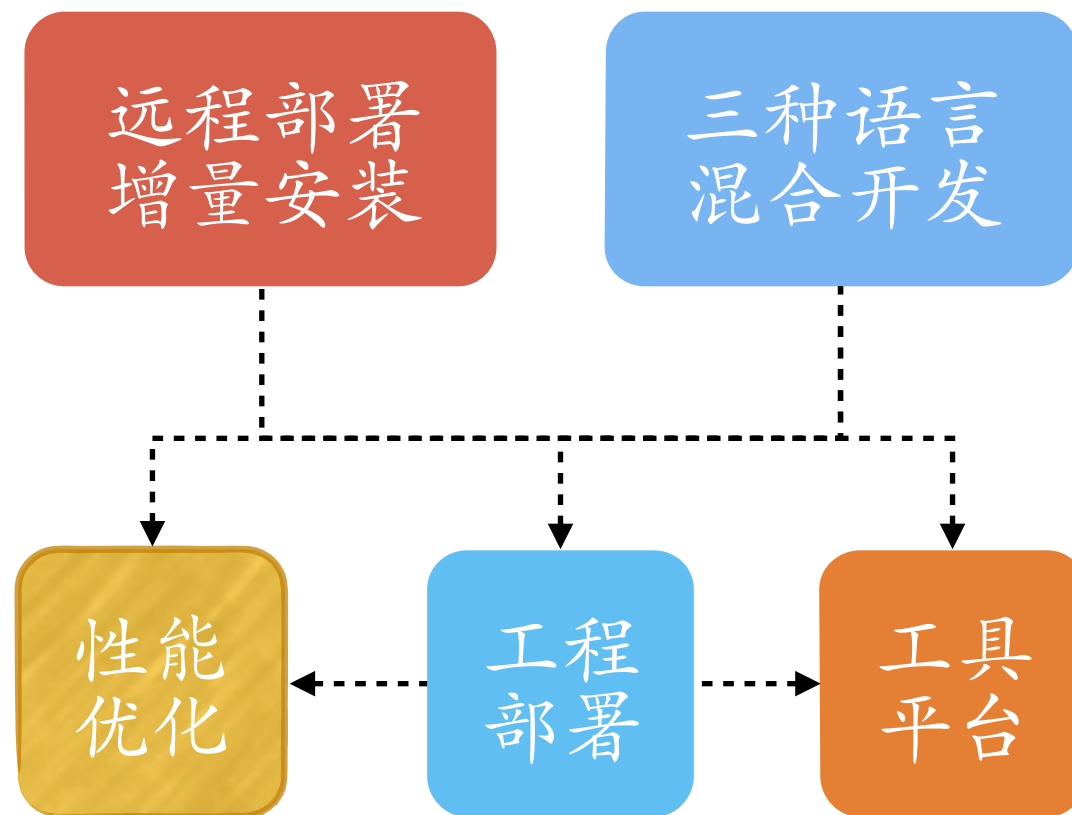
前端特殊性对架构的影响



前端特殊性对工程的影响



前端特殊性对工程的影响



解决方案？

“关于此，我确信已发现了一种美妙的解法，可惜这里空白的地方太小，写不下。”

–Pierre de Fermat

在前端，工程问题远比想象的要多很多！

“或许现在很多企业和团队尚未重视前端工程，或许前端工程在很多人眼里还只是“构建工具”的代名词，又或许未来前端领域的变革使得一切工程问题从根本上得到解决。不管怎样，我只是希望当下能认真的记录自己在前端工程领域的所见所想，与正在经历前端工程化改进，并被此过程困扰的同学交流心得。”

To be continued...