

FEX Web 前端研发部

静态资源自动合并系统

Shared By 王程

关于我

- 来自FIS小组

- 负责的工作

- 静态资源管理和优化

- 插播广告

- 更多精彩尽在[Fis官网](#)

大纲

1. 挑战和目标

2. 架构

3. 实现

挑战与目标

- 为什么合并？



减少摆渡次数，减少乘客等待时间
减少HTTP请求书，加快页面渲染速度

静态资源合并中的挑战

- 第一天：开发一个新的页面

<CSS for feature A>

<CSS for feature B>

<CSS for feature C>

<Html for feature A>

<Html for feature B>

<Html for feature C>



写了一个包含A，
B，C功能的页面

静态资源合并中的挑战

- 第二天：性能优化准则告诉我们 ...

<CSS for feature A>

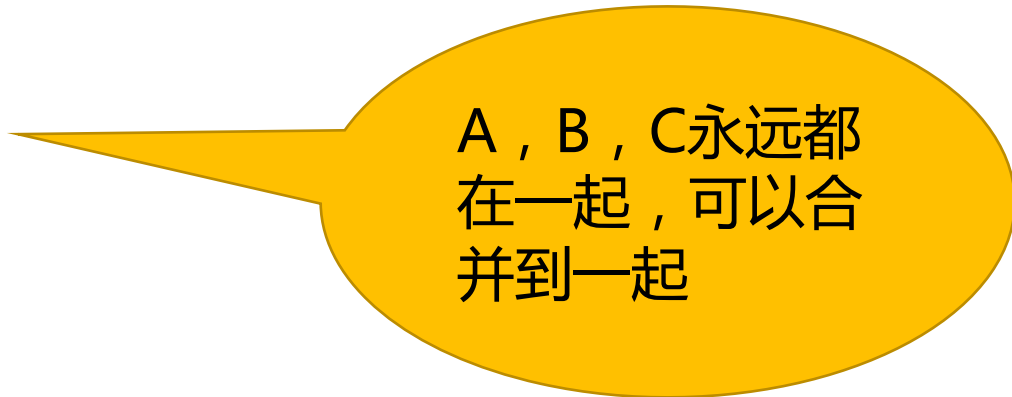
<CSS for feature B>

<CSS for feature C>

<Html for feature A>

<Html for feature B>

<Html for feature C>



A, B, C永远都在一起，可以合并到一起

静态资源合并中的挑战

- 第二天：请求减少了，性能优化了！

<CSS for feature A&B&C>

<Html for feature A>

<Html for feature B>

<Html for feature C>



三个请求变一个
请求！！！！

静态资源合并中的挑战

- 第三天：功能C不再使用

<CSS for feature A&B&C>

<Html for feature A>

<Html for feature B>

//不再使用 {<Html for feature C>}

静态资源合并中的挑战

- 第四天：网页出现了冗余资源

<CSS for feature A&B&C>

<Html for feature A>

<Html for feature B>

//不再使用{<Html for feature C>}

很难从资源包里
删除功能C的代码！

静态资源合并中的挑战

- 第N天：网页有很多冗余资源

<CSS for feature A&B&C&D&E&F&...>

if(非登录用户) <Html for feature D>

<Html for feature E>

if(登录用户){<Html for feature F>}

很多冗余的Css
合并到了一个包
里面

静态资源合并中的挑战

第一天：开发一个新的页面

```
<CSS for feature A>  
<CSS for feature B>  
<CSS for feature C>  
<Html for feature A>  
<Html for feature B>  
<Html for feature C>
```

资源合并：性能优化

第二天：

```
<CSS for feature A&B&C>  
<Html for feature A>  
<Html for feature B>  
<Html for feature C>
```

资源合并
减少请求

人工维护
合并方案

需求变更：性能恶化

第N天：网页有很多冗余资源

```
<CSS for feature A&B&C&D&E&F...>  
if(非登录用户)<Html for feature D>  
<Html for feature E>  
if(登录用户){<Html for feature F>}
```

不断变更：持续恶化

第四天：功能C不再使用

```
<CSS for feature A&B&C>  
<Html for feature A>  
<Html for feature B>  
//不再使用 {<Html for feature C>}
```

静态资源合并--目标

- 用户体验

- 只加载页面用到的Js和Css，减少冗余资源
- 不同的网络状况(Mobile/PC)，采用不同的合并方案

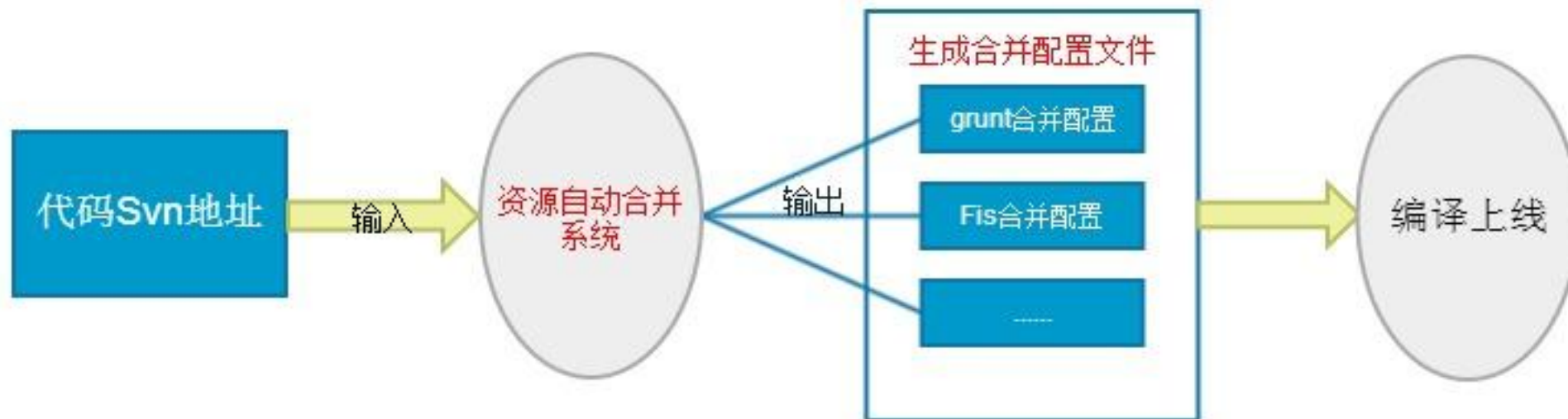
静态资源自动合并系统

- 产品开发体验

- 资源合并对工程师透明

架构

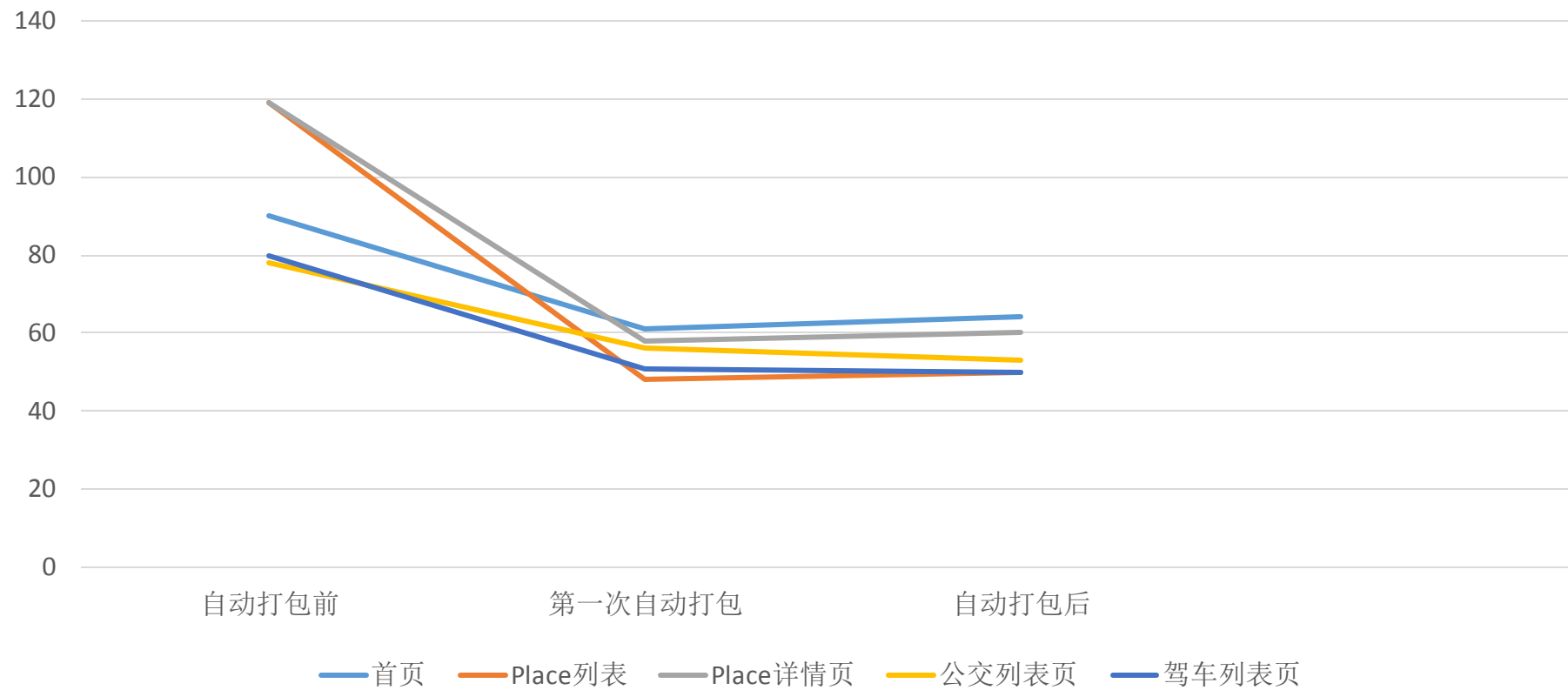
静态资源自动合并系统



静态资源自动合并系统 - 流程图

静态资源自动合并 — 收益

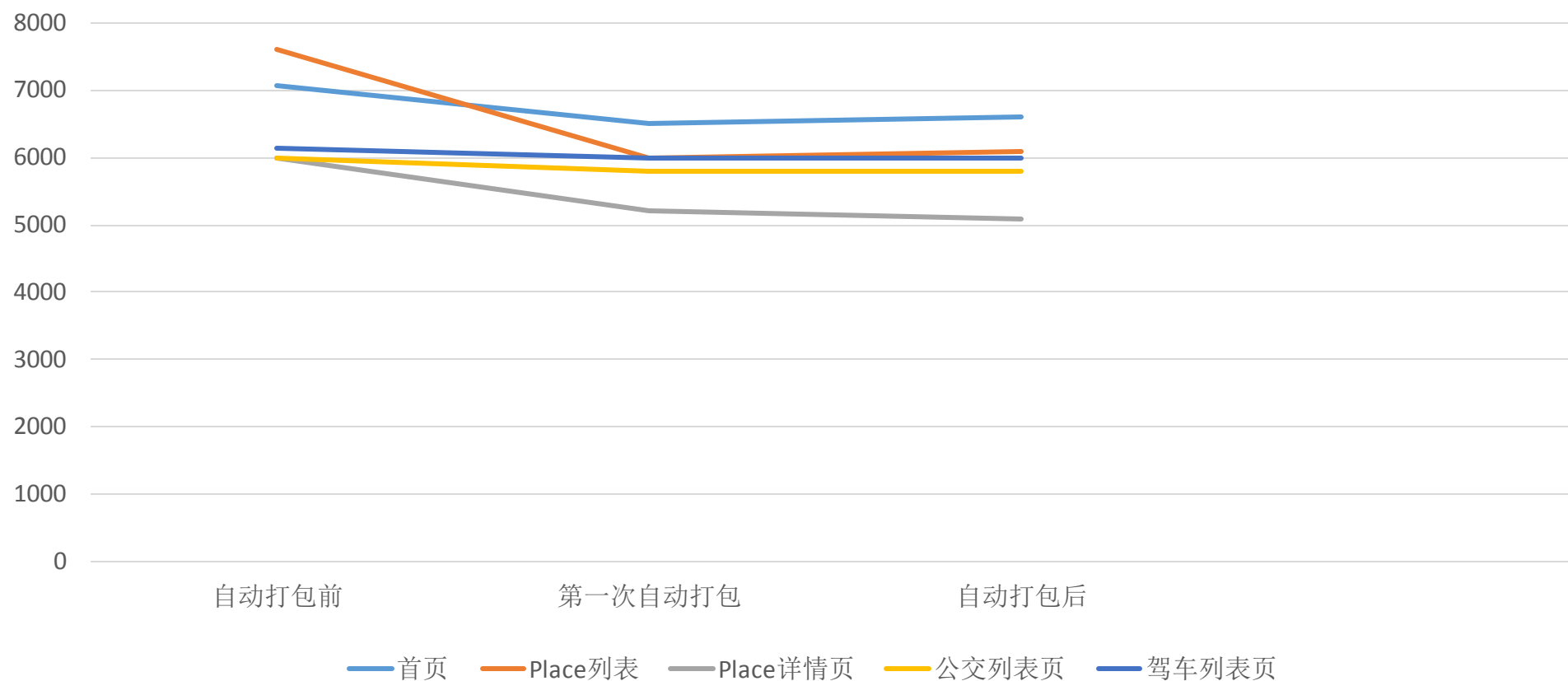
JS加载大小变化



自动合并前后js加载量平均减少35%左右

静态资源自动合并 — 收益

页面加载时间变化



自动合并前后
页面加载时间
减少10%

目标分析 – 需要做哪些

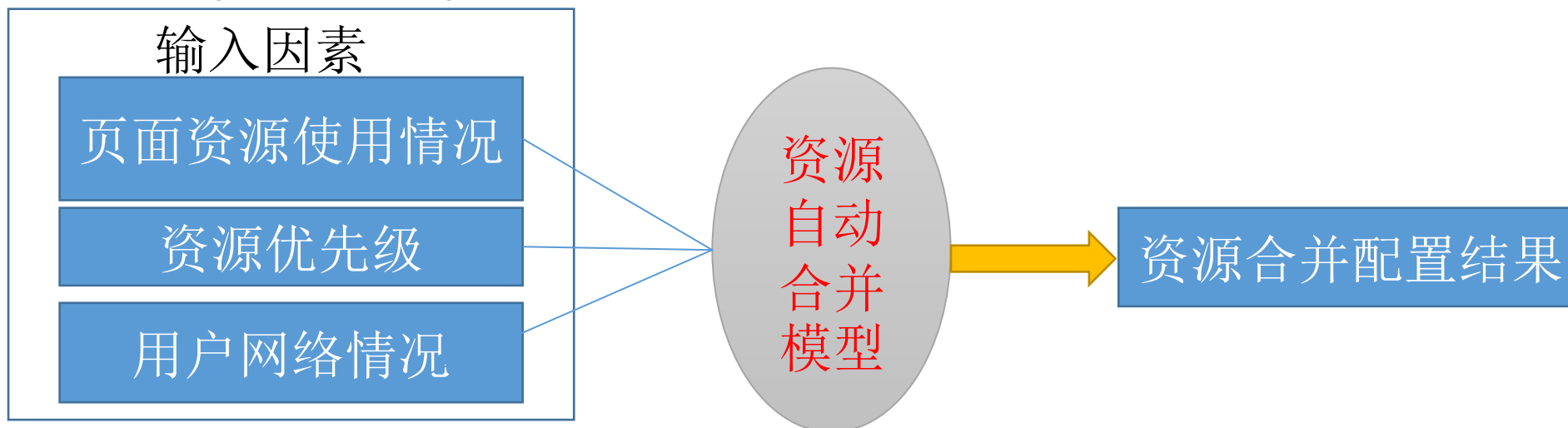
- 只加载页面用到的Js和Css，减少冗余资源

统计页面用到了哪些Js和Css

- 能够首先加载重要的静态资源，加快页面首屏的展现速度

统计哪些是重要的静态资源

- 不同的网络状况(Mobile/PC)，采用不同的合并方案



需要做哪些

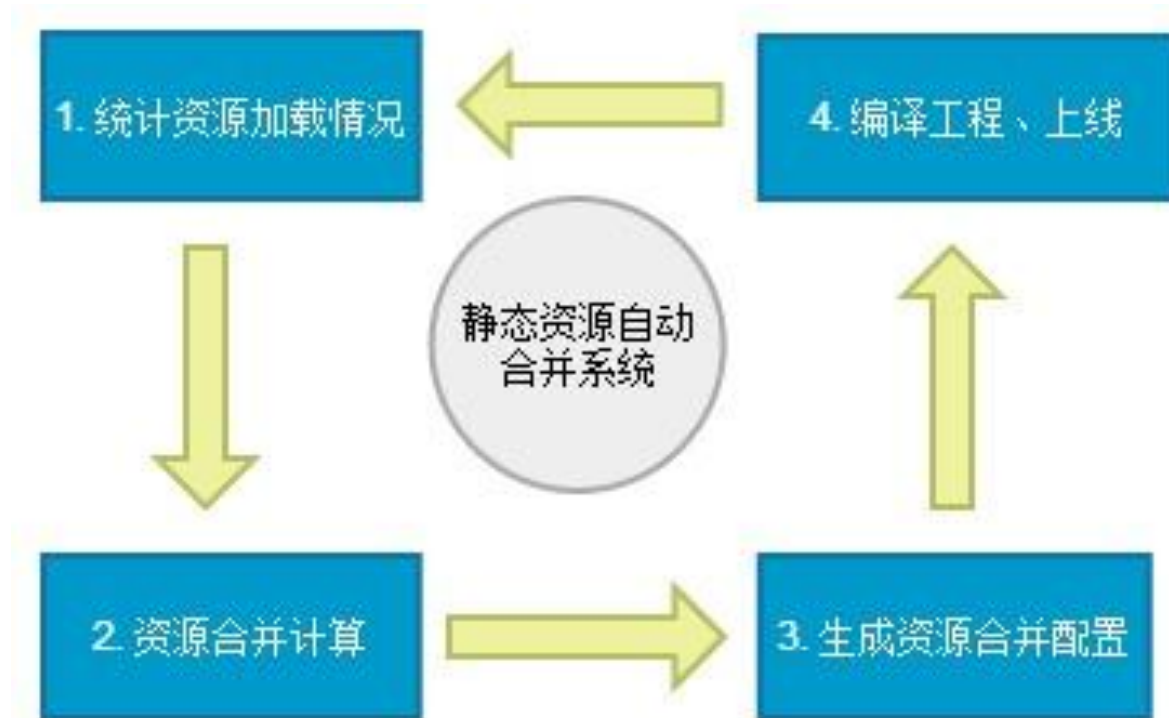
- 数据统计

- 页面使用了哪些Js和Css
- Js和Css的优先级
- 页面的PV

- 静态资源合并模型

- 将经常使用的Js和Css合并到一起
- 针对不同的网络进行优化

静态资源自动合并系统



静态资源自动合并系统-工作原理图

实现

怎么做



数据统计

合并模型

静态资源自动合并 — 统计



• 统计什么

- 页面静态资源使用情况
- 页面的PV
- 静态资源的大小

静态资源自动合并 — 统计



• 如何统计

- 组件化开发接口接入统计

• 示例

- Require :加载Js或Css组件
`{require name= "a.js" }`
- Widget :使用模版组件，并加载同名的Js和Css
`{widget name= "a" }`
- 只需要在require和widget接口接入统计脚本

静态资源自动合并 — 统计

- 统计的结果

	Page_1	Page_2	Page_3	Page_4
访问量	10	1	10	1
A.Css	√	√	√	√
B.Css	√	√	√	√
C.css	√			
D.Css				√

静态资源自动合并 — 合并模型

• 资源合并算法

- 合并收益：对于同时使用A和B的页面节省了一个网络来回时间
- 合并损失：对于只使用A的页面，浪费的B的大小
- 转化为时间：损失的大小 / 下载速度
- 纯收益：合并收益 - 合并损失

	Page_1	Page_2	Page_3	Page_4
A.Css	√	√	√	
B.Css	√	√		√

静态资源自动合并 — 合并模型

	Page_1	Page_2	Page_3	Page_4	Page_5
访问量	100	1	20	10	1
A.Js (1KB)	√	√	√	√	√
B.Js (1KB)	√	√	√	√	√
C.Js (300B)	√	√			
D.Js (2KB)					√
E.Js (700B)		√	√		
F.Js (600B)		√	√	√	√

• 常量

- 网络来回时间 = RTT = 50ms
- 下载速度 = Speed = 100kb/s

• 计算

- 收益 = $50\text{ms} * (100+1+20+10+1)$
- 损失 = 0

• 总结

- 不论RTT和Speed为何值，A和B肯定会被合并到一起。

静态资源自动合并 — 合并模型

	Page_1	Page_2	Page_3	Page_4	Page_5
访问量	100	1	20	10	1
A.Js (1KB)	√	√	√	√	√
B.Js (1KB)	√	√	√	√	√
C.Js (300B)	√	√			
D.Js (2KB)					√
E.Js (700B)		√	√		
F.Js (600B)		√	√	√	√

• 常量

- 网络来回时间 = $RTT = 50ms$
- 下载速度 = $Speed = 100kb/s$

• 计算

- 收益 = 0
- 损失 = $(2 * (100 + 1) + 0.3 * 1) / Speed$

• 总结

- 不论RTT和Speed为何值，C和D肯定不会合并到一起。

静态资源自动合并 — 合并模型

	Page_1	Page_2	Page_3	Page_4	Page_5
访问量	10M	1M	200K	10K	1K
A.Js (1KB)	√	√	√	√	√
B.Js (1KB)	√	√	√	√	√
C.Js (300B)	√	√			
D.Js (2KB)					√
E.Js (700B)		√	√		
F.Js (600B)		√	√	√	√

• 计算

- 收益 = $RTT * (1 + 0.2)M$
- 损失 = $(0.6 * 0.01 + 0.7 * 0.01)M / Speed$

• 总结

- 最终收益取决于RTT和Speed
- **移动网络**建立网络来回时间较长
收益更大一些，**会合并**到一起；
- **PC网络**建立网络来回时间较短
收益小一些，**不会合并**

静态资源自动合并 — 合并模型

	Page_1	Page_2	Page_3	Page_4	Page_5
访问量	10M	1M	200K	10K	1K
A.Js (1KB)	√	√	√	√	√
B.Js (1KB)	√	√	√	√	√
C.Js (300B)	√	√			
D.Js (2KB)					√
E.Js (700B)		√	√		
F.Js (600B)		√	√	√	√

• 结果

- Pkg1 : A,B,C
- Pkg2 : E,F

• 问题

- 资源增加算法复杂度指数级增加

静态资源自动合并 — 合并模型

	Page_1	Page_2	Page_3	Page_4	Page_5	Page_6	Page_7	Page_100
访问量	1000M	1000M	100M	10M	10M	10M	1M		1K
1.Css	√	√	√	√	√	√	√		√
2.Css	√	√	√	√	√	√	√		√
3.Css	√	√	√	√	√	√	√		√
4.Css	√	√			√		√		√
5.Css	√		√	√					√
6.Css		√	√		√	√	√		
.....									
100.Css	√		√	√			√		

无法遍历所有合并可能

合并模型 — 二次贪心

	Page_1	Page_2	Page_3	Page_4	Page_5	Page_6	Page_7	Page_100
访问量	1000M	1000M	100M	10M	10M	10M	1M		1K
(1+2).css	√	√	√	√	√	√			
(3+4).css	√	√	√		√		√		√
5.Css	√		√	√					√
6.Css		√	√		√	√	√		
.....									
100.Css	√		√	√			√		

静态资源自动合并 — 目标回顾

- **目标一**：只加载页面使用的Js和Css，减少冗余资源加载
- **实现**：收益损失模型
 - 页面共用资源则有合并收益，算法中倾向于合并
 - 页面不共用资源则有合并损失，算法中倾向于分开
- **目标二**：优先加载重要的静态资源，加快页面首屏的展现速度
- **实现**：资源分类
 - 统计资源时可以设置资源的优先级
 - 在打包过程中将高优先级和低优先级的资源分开打包

静态资源自动合并 — 目标回顾

- 目标三：不同的网络状况(Mobile/PC)，采用不同的合并方案
- 实现：收益损失模型
 - RTT和下载速度两个常量可以控制资源合并的结果

静态资源自动合并 — 未来

- 线上全自动

- 完全对工程师透明
- 实时优化

- 提供云服务对外开源

总结

静态资源合并

Web性能优化的重要手段

挑战

- 需求的变更、业务的复杂使得性能优化很难做的很好
- 对于大规模网站，资源合并就会更加困难

静态资源自动合并系统

- 系统的目标
- 合并模型来保证目标的实现

Thanks

感谢大家的光临！

- FEX官网 <http://fex.baidu.com>
- FIS官网 <http://fis.baidu.com>
- 参考文章A：[静态资源自动合并工具](#)
- 参考文章C：<http://...>