前端工程化

iQiYi前端组JS代码部署自动化实践

部署系统核心需求

- 把SVN trunk/branch上的代码经过构建过程部署到线上服务器/测试机，并对最终用户生效
- 备份与回滚
- 安全/稳定/高效

总体流程

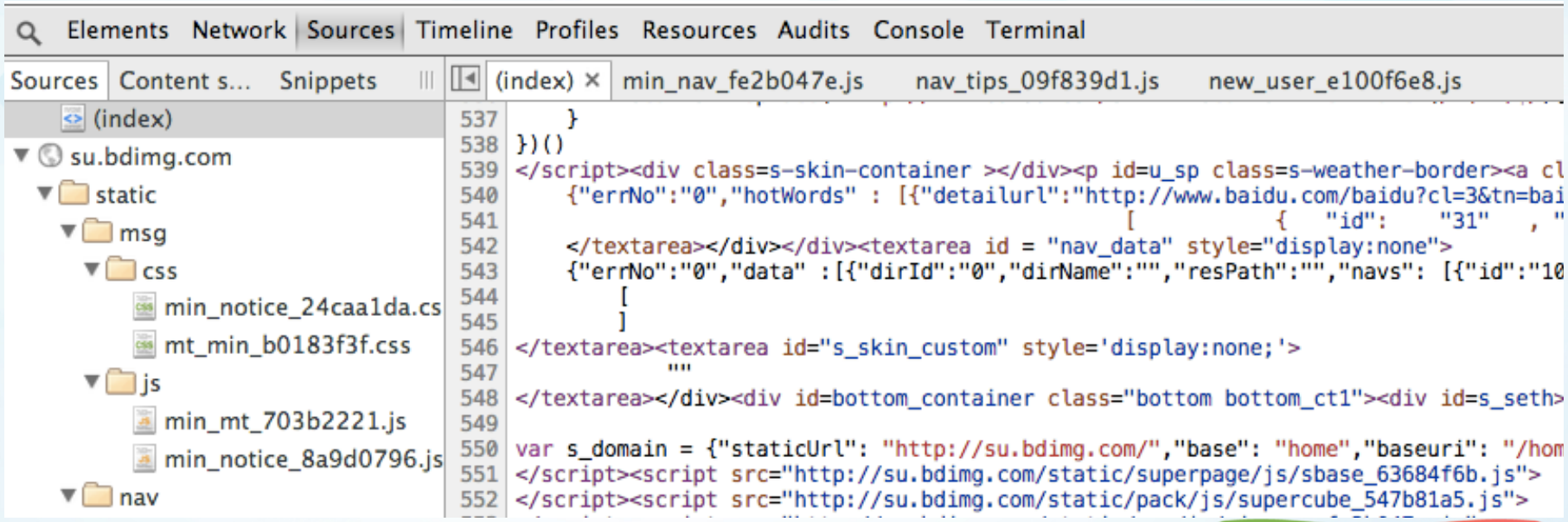
SVN服务器 --> 部署服务器 → 内网源服务器 →公网CDN -> 浏览器

如何生效？—— 静态资源版本更新方式

- 增量式
  - 部署到新的目录/文件名加MD5后缀，不覆盖线上同名文件
  - 页面中JS/CSS的url的pathname中包含版本号
- 覆盖式
  - 新上线的文件覆盖线上的同名文件
  - 页面中JS/CSS的pathname后面加版本号
- 增量式+覆盖式（iQiYi使用的）
  - 版本号存在小文件ver.js中，覆盖更新
  - 业务代码大文pathname中包含版本号，增量更新

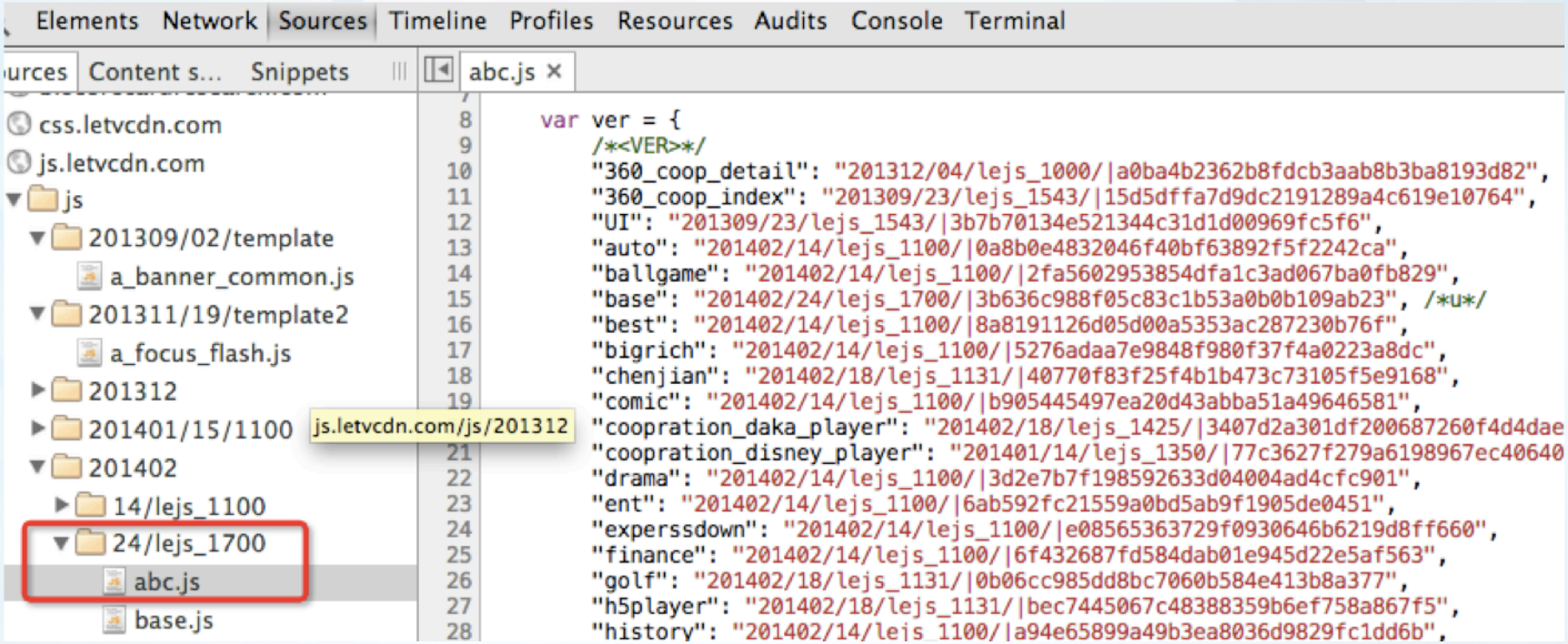
增量式版本更新方式

- 新上线文件的路径全新，立即生效
- 页面中JS/CSS的pathname中包含版本信息，硬链接，强缓存
- 版本更新依赖页面，任何一个JS/CSS更新，都要重新发布页面
- 只需回滚页面
- 性能最好，但对开发不友好，需要资源定位工具



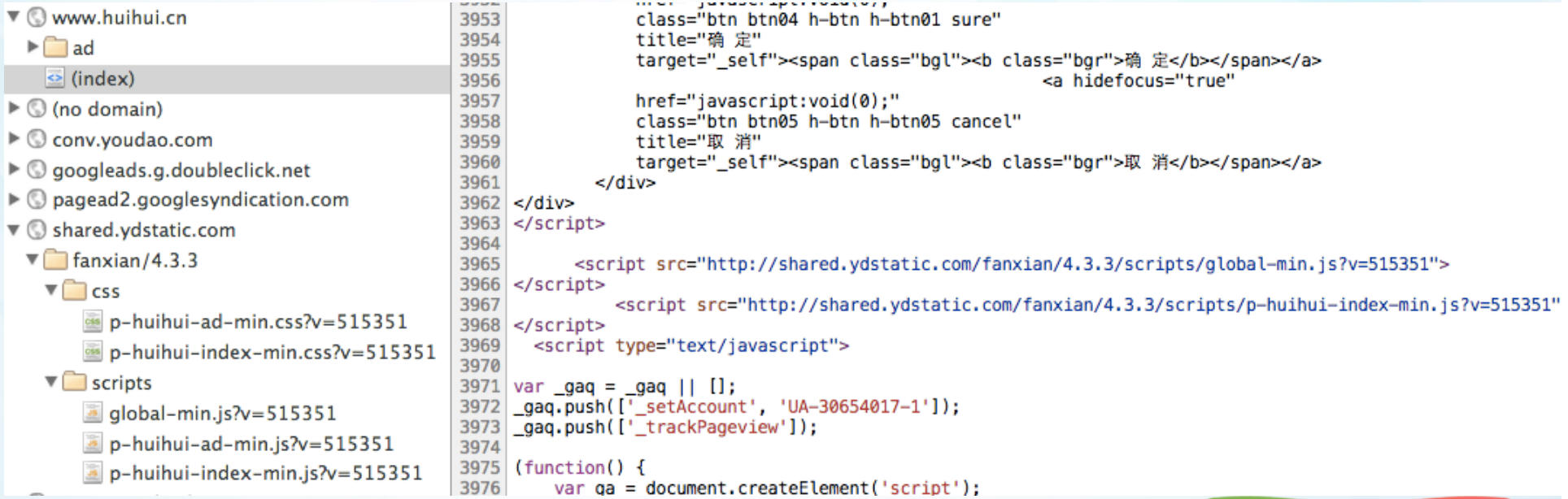
增量式版本更新（示例二）

- 版本号专门存储在一个JS文件中，页面上只有一个JS的硬链接
- 版本号/业务文件强缓存，但多一次版本号文件的http请求
- JS上线，只需要改变页面上版本文件的链接，页面改动较小
- 只需回滚页面



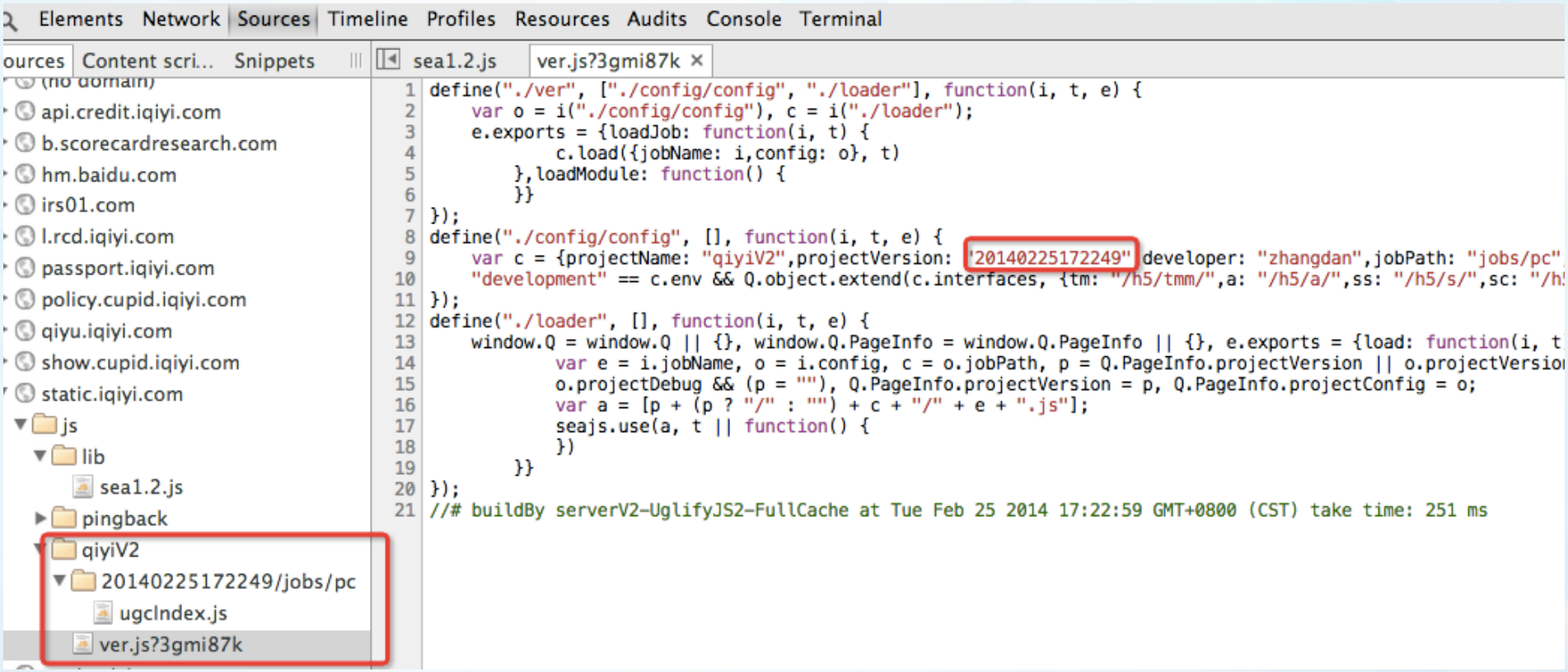
覆盖式版本更新方式

- 新上线的文件覆盖线上的同名文件，生效慢，弱缓存
- 不需要额外的http请求
- JS/CSS/HTML覆盖顺序问题
- JS/CSS/HTML都回滚
- 对开发而言，最简单



增量式+覆盖式（iQiYi使用的）

- 版本号存在小文件ver.js中，覆盖更新，弱缓存
- 业务代码大文pathname中包含版本号，增量更新，强缓存
- JS上线，不需要改动页面，但性能较差，多一个http请求
- 只需要回滚版本号JS文件
- 实质是全量式+覆盖式（版本号是整个项目的版本号，非单个文件的）



SVN服务器与部署服务器

- SVN文件读写
  - Checkout/export
  - Diff（显示文件变动列表）
  - Tag（备份trunk/大文件）
- 合并压缩
  - 计算匿名模块ID和依赖关系
  - 添加ID/deps
  - Combine/compress
  - removeSea（去除对模块加载器的依赖）
- 版本更新
  - 更新版本号
  - 整站部署

SVN 仓库布局

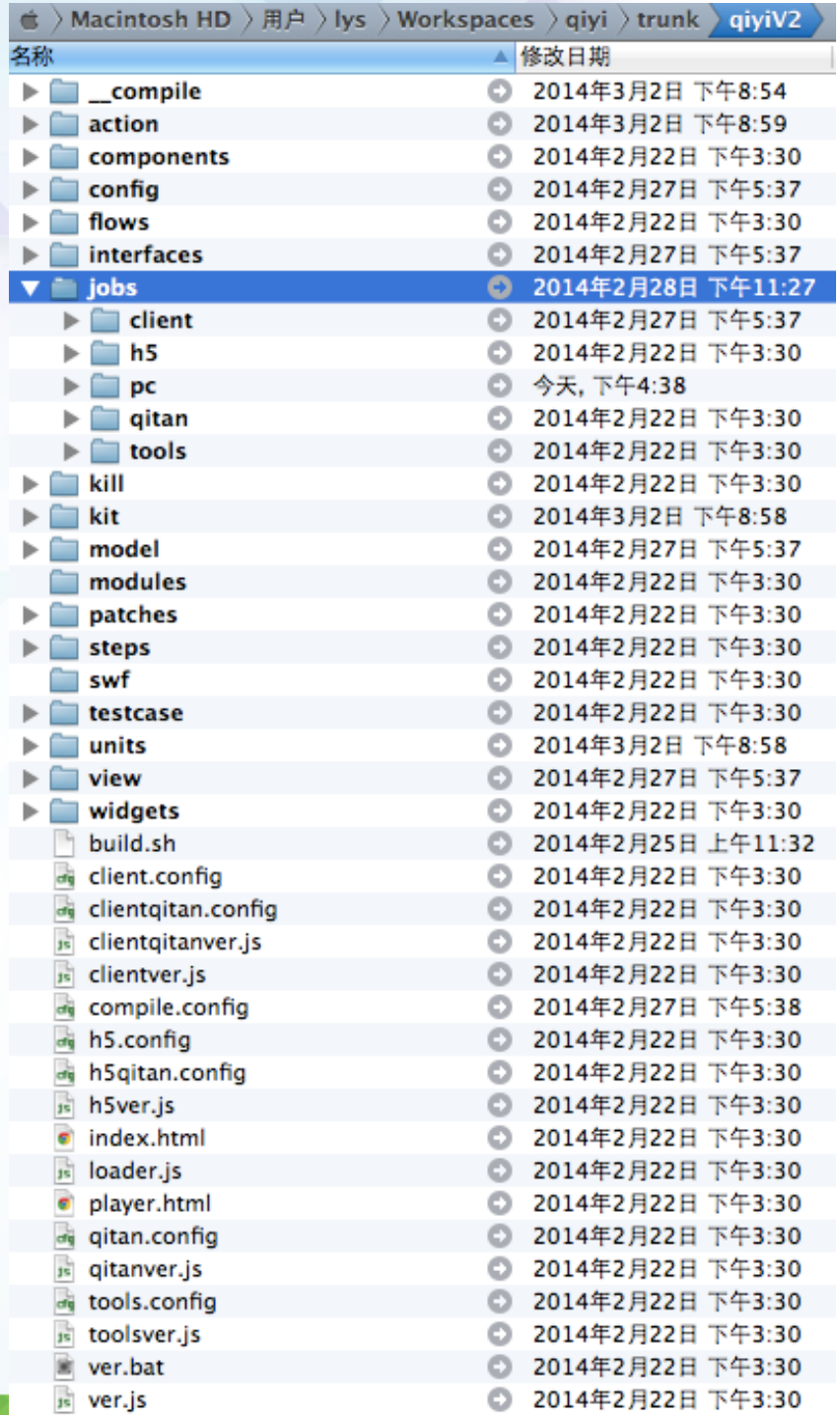
- trunk
- branch
- tags（历史上线的trunk快照，与trunk diff用）
- temp（与branch做diff使用）
- online（线上合并压缩后的代码备份，回滚用）

三套部署系统

- 基础库（不包含业务代码）
  - 需要removeSea，离开seajs也能使用
- 多个业务线项目共用一个trunk
  - 最大程度共用代码
- 每个业务线项目单独一个trunk
  - 业务线共用的代码放置于common中，单独作为一个trunk
  - 各个业务线项目通过svn external 共用common的代码

多个业务线共用一个trunk

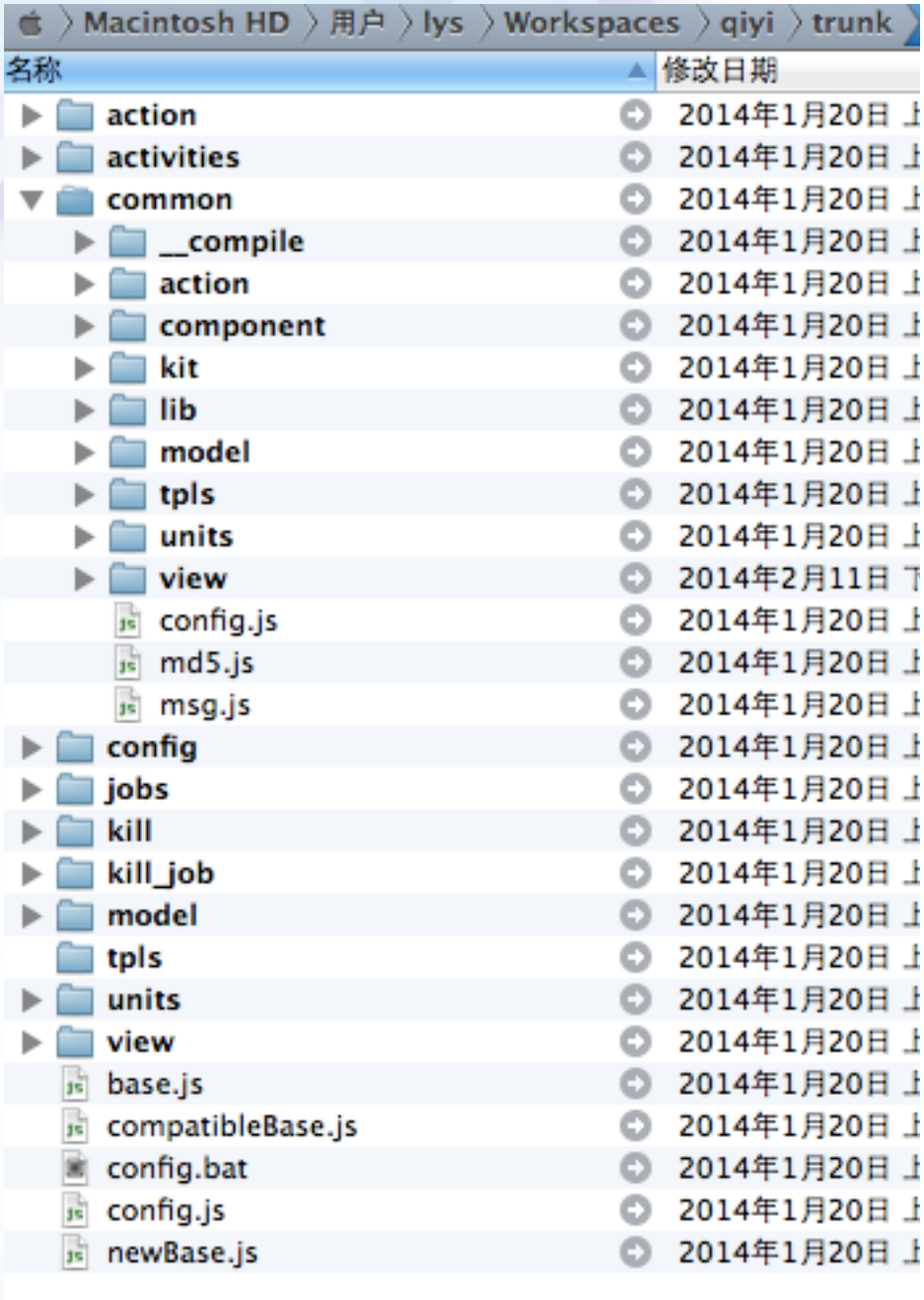
- 不同项目的入口文件
放在不同的目录下



各个业务线单独一个trunk

- 共用的代码common放在一个trunk
- 业务线的trunk把common链接为一个子目录

- 外链关系维护麻烦



共用trunk——部署流程最简单

包含svn外链——部署流程svn操作复杂

基础库部署

- 多了一个removeSea，去除合并后大文件中的define/require/exports
- 替代方案：在大文件之前自己实现define/require

关键环节——计算ID与依赖

- 开发时是匿名模块define(factory(){})
- require是相对路径

- 合并之前要先转成define(id, [deps], factory(){})格式

SPM合并匿名模块过程

1. 递归计算直接依赖和全部依赖

1) 读取文件jobA, 通过工具分析得到直接依赖unitA/unitB

2) 读取文件unitA,得到直接依赖widgetA/widgetB

3) 读取widgetA, 得到其直接依赖（无，退回unitA）。

4) 读取widgetB, 得到直接依赖（无，退回unitA，unitA依赖的都处理完，退到jobA）。

5)读取文件unitB, 重复2)-4)

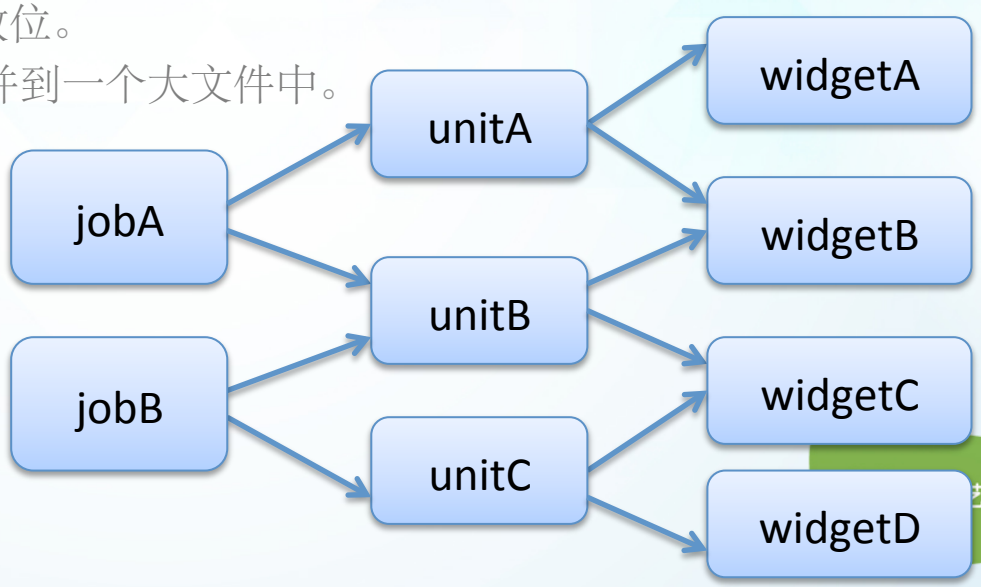
6)最后得到jobA的全部依赖unitA/unitB/widgetA/widgetB/widgetC

2. 计算和添加ID/deps

7)计算jobA和它的全部依赖相对于jobA的相对路径

8)把相对路径作为各个匿名模块的id，添加到define的第一个参数位。把直接依赖数组添加到define的第二个参数位。

9)把jobA和它的全部依赖合并到一个大文件中。



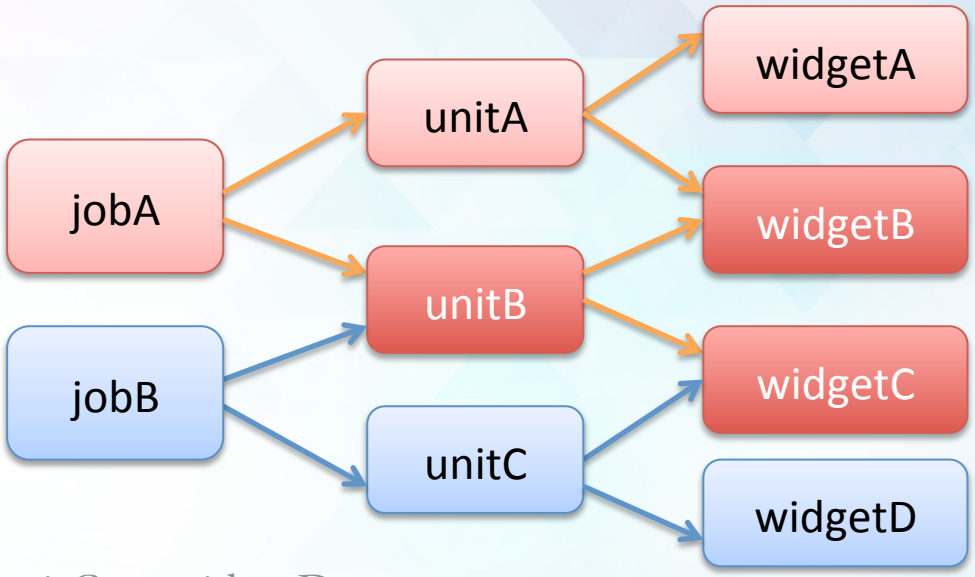
整站部署带来的问题

- 上线版本更新机制，决定了每次上线要重新部署整站的代码
- 从1500个小文件中合并生成150左右的大文件
- 单个入口文件依赖了300-500左右的小文件
- 合并耗7分钟！

改进的合并环节——加入小文件缓存

- 处理jobA时，已经把unitB和它的全部依赖模块都处理过了！每个小模块最好只处理一次！

- 对于右图
  - spm处理12次
    - jobA+5个全部依赖
    - jobB+5个全部依赖
  - 改进后的只需要处理9次
    - jobA+5个全部依赖
    - jobB+2个尚未处理的依赖unitC，weidgetD



- 改进之后，整站合并耗时直接由7分钟降到20秒左右
  - job文件之间的重合度非常高！

前端利器 —— UglifyJS

- 语法树工具
  - 获取依赖信息
  - 给匿名模块加ID和deps
  - 压缩代码
  - removeSea

在线调试环境：
http://hzxiaosheng.github.io/cmdBuild/demos/test-uglifyjs/basic.html

removeSea实现——举个例子

```
define("./main", ["./b", "./c"], function(require, exports, module){
    var b = require("./b");
    var c = require("./c");
    var ret = b * c;
    module.exports = ret;
});
define("./a", [], function(require, exports, module){
    module.exports = 1;
});
define("./b", [], function(require, exports, module){
    module.exports = 2;
})
define("./c", ["./a", "./b"], function(require, exports ,module){
    var a = require("./a")
    var b = require("./b");
    module.exports = a + b;
})
```

在线使用removeSea功能
http://hzxiaosheng.github.io/cmdBuild/demos/test-uglifyjs/removeSea-simple.html

removeSea之后的结果

```
window._MC_ = window._MC_ || {};

(function(__id__, __mod__) {
    __mod__[__id__] = 2;
})("b", (_MC_["b"] = {}) && _MC_);

(function(__id__, __mod__) {
    __mod__[__id__] = 1;
})("a", (_MC_["a"] = {}) && _MC_);

(function(__id__, __mod__) {
    var a = __mod__["a"];
    var b = __mod__["b"];
    __mod__[__id__] = a + b;
})("c", (_MC_["c"] = {}) && _MC_);

(function(__id__, __mod__) {
    var b = __mod__["b"];
    var c = __mod__["c"];
    var ret = b * c;
    __mod__[__id__] = ret;
})("main", (_MC_["main"] = {}) && _MC_);

if (typeof define !== "undefined") {
    define("main", [], function(require, exports, module) {
        module.exports = _MC_["main"];
    });
}
```

合并压缩之后，备份/显示变动列表

- 部署之后，整站的页面都会请求新上线的文件
- 确定上线影响的页面范围

- 实现方法：
  - svn diff 显示变动过的小文件列表+合并环节计算出的依赖关系
  - 入口文件依赖的任何小文件有改动，合并后的大文件也会变动

上传至内网源服务器

- CDN更新机制
- 如何让外网CDN尽快生效？purge
- 把上传到内网源服务器的文件，备份到svn online目录

Jenkins一键部署/回滚

Jenkins部署sample

Next，模板工程化——http://aui.github.io/tmodjs/

- 把HTML模板片段，编译成符合CMD规范的JavaScript文件
- 模板支持嵌套，提供依赖管理
- JS通过require/seajs.use来使用模板
- 模板自动化部署

模板工程化的好处

- 实现前端模板按文件与目录组织，脱离页面
- 在开发阶段将编译模板为高性能的 js 文件
- 支持模板之间支持引入外部模板
- 使用同步模板加载接口
- 支持本地调试模板
- 进入JS自动化部署流水线

存放方式	书写友好	CDN 部署	本地调试	代码复用	按需加载
内嵌业务页中	✓	✗	✗	✗	✗
Ajax 远程加载	✓	✗	✓	✓	✓
嵌入 js 文件	✗	✓	✓	✓	✓

Thanks!

开启全新视界