前端文件发布机制的研究

2009年8月3日
14:24

前端文件发布机制的研究

一、目的

在开发过程中，
- 有时遇到浏览器缓存问题导致页面不能及时更新
- 有时页面引入了过多的不必要的样式脚本文件
- 有时由于文件太多，字节过大导致页面的性能缓慢

为了解决这些问题，设想了以下解决方案

二、文件目录结构

目录	目录结构	备注
Home/Public/css	- **img** 样式图片 （其他目录不许方式图片） ○ Product 应用目录的样式引用的图片 ○ Common ○ Jquery 插件中引用到的图片 - common 全局样式文件 - product 按逻辑应用分化目录 - jquery 插件样式 - **release** 提供合并压缩文件存放的目录	- 调整： 调整了图片的目录结构，由于在合并压缩文件的过程中，原有图片引用路径多样化，无法确定合并后的引用路径，因此调整成新的目录结构。 注：样式的背景图片引用需要按照这个目录结构来引用
Home/Public/js	- jquery 插件脚本 - lib 公共样式 - product 按逻辑应用分化目录 - **release** 提供合并压缩脚本存放的目录	js中删除了原有的jquery插件引用的样式和图片的目录。

注：CSS涉及背景图片的引用，为了统一img图片的应用方式，
在目录结构中Home/Public/css/一级目录/ ,最好只有一级目录，一级目录下不可以再建文件夹，
目前程序，在合并多个css文件为一个文件的过程中，只将background:(../img/eg.gif)的 ../img 替换相应的路径，对 ../../img 等不做替换

三、解决方案

配置文件：

```
{
    "Version":"v3",//版本
    "Update":"20091015",//根据时间戳更新缓存
    "Compress":"1",//文件输出方案（1，2，3，4）
    "Common":[//全局级文件
      "common/global.css",
      "common/reset.css",
    ],
    "Domain":{//引用的跨域文件（该级别的文件将会直接按照原路径引入，不做压缩处理）
        "d_name_1":["http://www.liba.com/a.css","http://www.liba.com/a2.css"]
    },

    "Module":{//模块级文件
       "m_name_1":["module/s1","module/s2.css"]
       "m_name_2":["module/s3.css","module/s4.css"]
    },
    "Page":{//页面级文件
       "index_php":["","d_name_1","sys/index.css"],
                //第一个元素存储 模块 引用名称，无则留空
                //第二个元素存储 跨域 引用名称，无则留空
                //引用多个模块或跨域文件时，用 “，”隔开，比如： 模块1，模块2
       "search_php":["m_name_1","","product/p1.css","product/p2.css"]

     }
}
```

注：文件的路径相对与资源文件的目录来写，
比如 /home/css下放置所有的css文件，那么：/home/css/product/p1.css在填写配置文件时，
填写为product/p1.css

1、页面的资源级别：
- 全局级（Common）
  全站都会用到的文件
- 模块级（Module）
  按模块划分出的文件
- 页面级（Page）
  当前页面中，除了全站级，模块级文件之外引用到的资源文件

2、优化方案

减少单个文件的字节大小：

- 移除注释
- 移除额外的空格
- 移除换行符

减少文件的个数（即减少http请求数）：
按照资源文件的级别，合并当前级别下多个文件为一个文件

3、缓存问题

采用时间戳后缀

4、压缩方案

采用当前开源的css，js压缩库，我这里选择php版本的

压缩方案	开源类库
CSS	class.csstidy.php 地址: http://csstidy.sourceforge.net/
JS	class.JavaScriptPacker.php 地址：http://dean.edwards.name/packer/

5、文件输出方案

基于资源级别及优化方案，制定以下4种方案：
1、 直接依次引入单个文件（<link/>、<script/>）
不做压缩，不做合并，直接按原有的文件路径引入文件
2、 基于1，进行单个文件压缩，并引入新的文件
比如：原有文件及目录：sys/index.css，压缩后的文件及目录：release/index.pack.css

3、按照资源级别合并该级别下的资源文件为单个文件
比如search_php页面选择文件输出方案3，结果为：
//全局
/css/release/common.comb.css (包含全局下的global.css,reset.css)
//模块
/css/release/m_name_1.comb.css(包含m_name_1下的s1.css,s2.css)
//页面级
/css/release/search_php.comb.css(包换p1.css p2.css)

4、基于3，进行合并后的文件压缩
会先进行文件合并，这步完成后会生成3方案中的新文件，然后再进行压缩
如：3方案中的全局文件合并后为release/common.comb.css，
那么再压缩后变为release/common.comb.pack.css

注：合并后的文件总数=全局级1个文件+模块级n个数+跨域文件m个数+页面级1个文件

四、程序设计

1. 读取json配置文件，转化成php数组
2. 根据每个页面的资源参数，查找该页面拥有的样式资源
3. 根据文件输出方案，进行处理
4. 在页面上输出PHP数组结果

比如采取文件输出方案4：

```
Array (
   [common] => <link type="text/css" href="http://XXXXX/css/release/common.comb.pack.css?20091015" rel="stylesheet" />
     //全局文件 1个
   [module] => <link type="text/css" href="http://XXXXX/css/release/m_name_1.comb.pack.css?20091015" rel="stylesheet" />
               <link type="text/css" href="http://XXXXX/css/release/m_name_2.comb.pack.css?20091015" rel="stylesheet" />
     //模块文件 n个，取决于引用的模块个数，  这里
   [page] =>  <link type="text/css" href="http://xxxxx/css/release/search_php.comb.pack.css?20091015" rel="stylesheet" />
           //页面文件 1个
             <link type="text/css" href="http://xxxxx/css/release/跨域文件.css?20091015" rel="stylesheet" />
           //跨域引用文件 m个，取决与跨域引用的文件个数
)
```

例子：

比如search.php这个页面的样式引用

- 根据页面找到页面级资源： Page["search_php"]
- 根据该数组的第一个值查找所引用的模块样式
- 根据配置文件中的“Compress”值确定文件输出方案，并进行处理
- 在页面上进行输出

```
//全站样式 common
<link type="text/css" href="common/global.pack.css?20091015" rel="stylesheet" />
```

```
//模块样式 module
<link type="text/css" href="product.pack.css?20091015" rel="stylesheet" />
//页面样式 page
<link type="text/css" href="product/search.pack.css?20091015" rel="stylesheet" />
```

注：

1、配置文件没有更新的情况下，只解析一次json为数组，
2、只解析一次资源，并将结果保存，供下次直接使用

五、配置结果如何使用

引用改程序最终输出作用为两部分
1、 根据配置文件及输出方案，优化输出了文件
2、 输出了每个页面需要引用的文件的字符串 (<link/>,<script/>)

输出数组：array("common"=>"", "module"=>"", "page"=>"");

方式一：

CSS资源文件的引用方式：

在页面中设置一个 php变量: $CSS_USE_PATH，并将 输出数组的里面的引用文件资源的字符串按照全局，模块，页面三个级别依次连接，并赋值给改变量。
注：在YAHOO的14条性能优化准则中，css文件需要全部放在头部，因为需要放在头部

JavaScript资源文件的引用方式：

设置对应资源级别需要设置三个php变量：
$JS_COMMON ： 对应 输出数组的 common结构的赋值
$JS_MODULE ： 对应 输出数组的 module的赋值
$JS_PAGE： 对应 输出数组的 page赋值

设置三个变量的原因：
1、 脚本的全局变量$JS_COMMON，有时需要在头部引入，比如jquery.js文件的引用，需要在一开始在头部引入，
因为有时需要在页面中间部分执行某些函数，如果放在底部，jquery中的库函数就无法调用
2、$JS_COMMON，$JS_PAGE按照模块级，页面级的顺序，依次写在页面中。

注：在YAHOO的14条性能优化准则中，js文件建议放在底部，如果没有特殊原因还是全局变量还是建议放在底部的。
总之，按照 全局级，模块级，页面级，依次引入相应资源即可

方式二：
将每个页面的输出结果，写到一个模板文件中，在php页面中根据页面的tag参数，引用改模板文件

比如
Search.php的Tag参数为 search_php，那么引用模板文件Search_php_css.html，其内容为：

```
//全站样式 common
<link type="text/css" href="common/global.pack.css?20091015" rel="stylesheet" />
//模块样式 module
<link type="text/css" href="product.pack.css?20091015" rel="stylesheet" />
//页面样式 page
<link type="text/css" href="product/search.pack.css?20091015" rel="stylesheet" />
```

css可以采用这种方式，JavaScript文件不宜采用，因为，脚本文件有时需要分几个部位输出。

六、若干问题

跨域引用资源文件	考虑到有跨域引用文件的问题，因此提供了跨域文件的一个配置级，但不能对跨域文件进行压缩等等的处理，只能单个引入原文件
配置填写时的资源顺序文件问题： 在传统的直接引入资源文件的时候，是存在顺序性的，采用配置文件，只是将这种顺序性整理到配置文件中	css： css文件的顺序行问题不强，根据css选择器的规则，如果规则设计分得很清楚，那么不要考虑填写的顺序，但是还是需要按照资源级别来填写配置文件的使用。 JavaScript： JavaScript不是编译型的语言，是解释型的语言，它有顺序行。它的变量作用域存在先后的问题， 所以必须按文件的引用顺序来书写
php文件和css，js文件是否在同一台服务器	css，js，包括css中的img，个人觉得没有必要和php文件分开服务器放置，css，js的应该不会超过1M，css引用的img图片的总大小不会很大。 资源网站的业务图片，比如：电子商务网站的产品图片，是有必要在不同服务器放置的。
其他问题?	欢迎大家提出并讨论

七、模块化开发样式和脚本

采用目前这个发布机制的配置方案，可以实现这种样式模块化开发的方案。

理想化的css按需获取的方式

全局样式模块	如： reset重置样式.css 全局的通用.css
模块化样式	如: 历史查看.css 歌曲模块.css
页面级样式	如：search.css
根据页面中的模块，设置改页面的配置文件	如：search搜索页需要 全局样式 模块样式: 1.列表样式.css 2.历史查看.css 3.歌曲模块.css 页面样式： Search.css **按照上面的资源级别，采用JSON的配置方案，填写配置即可完成。**

模块化样式文件，并对模块化的样式进行归类，按照归类的程度，按照JSON配置最终打包成一个文件发布，

比如：
全局样式：在开发的过程中，按模块 reset.css，排版样式.css 创建，最终按照 JSON配置，打包成一个文件发布
以前我是直接写在一个文件中的。

对于如何恰当的对资源文件划分模块是个需要深思熟虑的问题。

八、结论
一方面解决了缓存问题，改善了文件的模块化，在一定上程度了优化文件大小，经过简单配置文件，可以自动进行资源文件的压缩优化处理，并提供给站点使用。