大型网站之网站静态化（综合篇） - Robin Hu的专栏 - 博客频道 - CSDN

原文：http://blog.jobbole.com/84200/
http://blog.jobbole.com/84328/

一、简介

网站的web前端要实现高效，第一个要解决的短板就是网络的延迟性对网站的加载效率的影响，当然很多人会说网速快不快这是网络运营商的问题，不是网站的问题，但是大家肯定也见过就算我们用上了千兆宽带也会有些网站加载速度慢的让人无法忍受，网站本身的确是没法控制网络速度的能力，但是如果我们不降低网络对页面加载效率的影响，其他任何优化网站的手段也就无从谈起，原因就是网络效率对于网页加载效率的影响是起到大头作用的，只有这个大头被解决了，那么解决其他的小头才能发挥作用。

大型动态网站之所以可以做到能快速响应高并发，它们都是尽量让自己的网站静态化，当然这种静态化绝不是把网站就做成静态网站，而是在充分理解了静态网站在提升网站响应速度的基础上对动态网站进行改良。网站静态化的关键点是动静分离，解决这个关键点的本质就是为了降低网速对网站加载效率的影响/

静态网站非常简单，它就是通过一个url访问web服务器上的一个网页，web服务器接收到请求后在网络上使用http协议将网页返回给浏览器，浏览器通过解析http协议最终将页面展示在浏览器里，有时这个网页会比较复杂点，里面包含了一些额外的资源例如：图片、外部的css文件、外部的js文件以及一些flash之类的多媒体资源，这些资源会单独使用http协议把信息返回给浏览器，浏览器从页面里的src，href、Object这样的标签将这些资源和页面组合在一起，最终在浏览器里展示页面。但是不管什么类型的资源，这些资源如果我们不是手动的改变它们，那么我们每次请求获得结果都是一样的。这就说明静态网页的一个特点：静态网页的资源基本是不会发生变化的。因此我们第一次访问一个静态网页和我们以后访问这个静态网页都是一个重复的请求，这种网站加载的速度基本都是由网络传输的速度，以及每个资源请求的大小所决定，既然访问的资源基本不会发生变化，那么我们重复请求这些资源，自己在那里空等不是很浪费时间吗？如是乎，浏览器出现了缓存技术，我们开发时候可以对那些不变的资源在http协议上编写相应指令，这些指令会让浏览器第一次访问到静态资源后缓存起这些静态资源，用户第二次访问这个网页时候就不再需要重复请求了，因为请求资源本地缓存，那么获取它的效率就变得异常高效。

二、CDN技术

由于静态网站的请求资源是不会经常发生变化的，那么这种资源其实很容易被迁移，我们都知道网络传输的效率是和距离长短有关系的，既然静态资源很容易被迁移那么我们就可以把静态资源服务器按地域分布在多个服务节点上，当用户请求网站时候根据一个路由算法将请求落地在离用户最近的节点上，这样就可以减少网络传输的距离从而提升访问的效率，这就

是我们长提的大名鼎鼎的CDN技术，内容分发网络技术。

CDN技术应该由三个步骤组成，首先是解析DNS，找到离用户最近的CDN服务器，接下来CDN要做一下负载均衡，根据负载均衡策略将请求落地到最合适的一个服务器上，如果CDN服务器上就有用户所需要的静态资源，那么这个资源就会直接返回给浏览器，如果没有CDN服务器会请求远端的服务器，拉取资源再把资源返回给浏览器，如此同时拉取的资源也被缓存在CDN服务器上，下次访问就不需要在请求远端的服务器了，CDN存储资源的方式使用的是缓存，这个缓存的载体是和apache，nginx类似的服务器，我们一般称之为http加速器，之所以成为http加速器是为了和传统静态web服务器区别开来，传统的静态资源服务器一般都是从持久化设备上读取文件，而http加速器则是从内存里读取，不过具体存储的计算模型会根据硬件特点做优化使得资源读取的效率更高，常见的http加速器有varnish，squid。Ngnix加上缓存模块也是可以当做http加速器使用的，不管使用什么技术CDN的服务器基本都是做一个就近的缓存操作

三、减少http请求

网络传输效率还和我们传输资源的大小有关，因此我们在资源传输前将其压缩，减小资源的大小从而达到提升传输效率的目的；另外，每个http请求其实都是一个tcp的请求，这些请求在建立连接和释放连接都会消耗很多系统资源，这些性能的消耗时常会比传输内容本身还要大，因此我们会尽力减少http请求的个数来达到提升传输效率的目的或者使用http长连接来消除建立连接和释放连接的开销。

四、动静分离

我常常认为最佳的性能优化手段就是使用缓存了，但是缓存的数据一般都是那些不会经常变化的数据，上文里说到的浏览器缓存，CDN其实都是可以当做缓存手段来理解，它们也是提升网站性能最为有效的方式之一，但是这些缓存技术到了动态网站却变得异常不好实施，这到底是怎么回事了？

首先动态网站和静态网站有何不同呢？我觉得动态网站和静态网站的区别就是动态网站网页虽然也有一个url，但是动态网站网页的内容根据条件不同是会发生改变的，而且这些变化的内容却是同一个url，url在静态网站里就是一个资源的地址，那么在动态网站里一个地址指向的资源其实是不同的。因为这种不同所以我们没法把动态的网页进行有效的缓存，而且不恰当的使用缓存还会引发错误，所以在动态网页里我们会在meta设定页面不会被浏览器缓存。

如果每次访问动态的网页该网页的内容都是完全不同的，也许我们就没有必要写网站静态化的主题了，现实中的动态网页往往只是其中一部分会发生变化，例如电商网站的菜单、页面头部、页面尾部这些其实都不会经常发生变化，如果我们只是因为网页一小部分经常变化让用户每次请求都要重复访问这些重复的资源，这其实是非常消耗计算资源了，我们来做个计算吧，假如一个动态页面这些不变的内容有10k，该网页一天有1000万次的访问量，那么

每天将消耗掉1亿kb的网络资源，这个其实很不划算的，而且这些重复消耗的宽带资源并没有为网站的用户体验带来好处，相反还拖慢了网页加载的效率。那么我们就得考虑拆分网页了，把网页做一个动静分离，让静态的部分当做不变的静态资源进行处理，动态的内容还是动态处理，然后在合适的地方将动静内容合并在一起。

五、动静合并

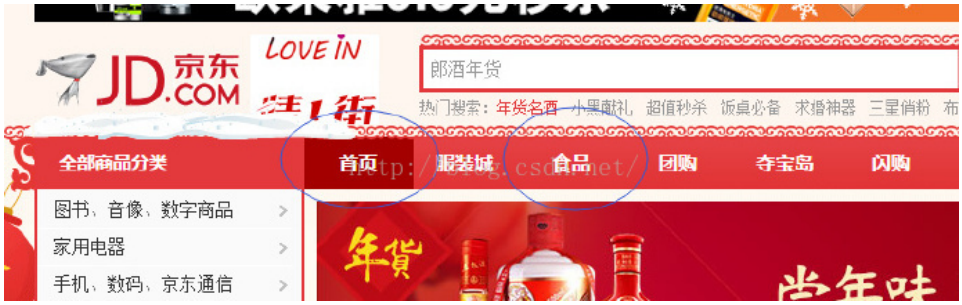
内容动静分离后，需要把他组合起来，即动静合并。动静合并的位置非常的关键，这个位置的选择会直接导致我们整个web前端的架构设计。

在Java的web开发里，页面技术jsp本身就包含了将页面动静分离的手段，例如下面的代码：

```
<%@ include file="header.jsp" %>
<body>
……
<%@ include file="footer.jsp" %>
```

一般一个网站的头部和尾部都是一样，因此我们把头部的代码单独放置在一个header.jsp页面里，页面尾部的代码放置下footer.jsp页面里，这样技术人员在开发页面时候就不再需要重复编写这些重复的代码，只需要引用即可，这个做法最大的好处就是可以避免不同页面在相同代码这块的不一致性，假如没有这个统一引用的话，手动编写或者复制和粘贴，出错的概率是非常的高的。

但是这个做法有一个问题，问题就是这种动静分离其实都是作用于单个页面的，也就是说每个页面都要手动的重复这个动静分离的操作，大多数情况这种做法都不会有什么问题，但是对于一个大型网站而言这种做法就有可能会制造不必要的麻烦，这里我截取了一张京东的首页，如下图所示：



讲述前我要事先声明下，京东网站可能不存在我要讲述的问题，我这里只是使用京东网站的首页做例子来说明，看图里的首页和食品两个条目，有些公司做这样的网站时候这些导航进入的页面会是一个独立的工程，每个工程都是由独立的项目组开发维护的，虽然项目组不同但是他们页面的整体结构会是一致的，如果按照上面的动静分离手段，那么每个项目组都要独立维护一份相同的头部尾部资源，这个时候麻烦来了，如果该公司要新增个新的条目，那么每个项目组都要更新自己不变的资源，如果该企业一共分了5个项目组，现在又做了一个新的条目，那么其他与之无关的项目组都得折腾一次更改统一引用文件的工作，要是做的不仔细就有可能出现页面展示不一致的问题，为了解决这个问题，java的web开发里就会考虑使用模板语言替代jsp页面技术，例如模板语言velocity，这些模板语言都包含一个布局的功能，例如velocity就有这样的功能，我们看看velocity的布局模板实例，如下所示：

```
<html>
<head>
<metahttp-equiv="Content-Type"content="text/html;charset=utf-8" />
<title>#springMessage("page_upop_title")title>
<metahttp-equiv="X-UA-Compatible"content="requiresActiveX=true" />
<metaname="keywords"content='#springMessage("page_upop_keywords")'/>
<metacontent='#springMessage("page_upop_description")'name="description"/>
head>
<bodyoncontextmenu="return false"onselectstart="return false">
#if($pageHead)
#parse($pageHead)
#end
$screen_content
#parse($page_footer)
body>
html>
```

页面里我们可以引入这个布局格式，这个布局文件其实就是页面里不变的东西抽取了出来，它完成了页面动静分离，页面只要应用这个布局文件即可，到了这里这个布局文件和前面的include方式区别不大，那么我们再看看下面的代码：

```
1 <property name="layoutUrl" value="layout/default.vm"/>
```

这是布局文件的引用方式，我们可以把布局文件放置在网络上，项目里应用这个文件所在地址即可，这样我们就把项目里不变的静态资源抽取在同一个地方，如果在碰到布局要做修改，那么我们只需要改一个地方即可。

不管服务端采取何种动静分离，动静资源的整合都是有服务端完成，按照上文提到网站静态化的思想，这些做法不会给网站性能提升带来任何好处，它们只是给开发，运维提供了便利而已，我们要把动静分离往前移，服务端往前移碰到的第一个点就是静态的web服务器例如apache或ngnix。

（1）web服务器的动静合并

java的web开发里我们一般使用jsp来编写页面，当然也可以使用先进点的模板引擎开发页面例如velocity，freemark等，不管我们页面使用的是jsp还是模板引擎，这些类似html的文件其实并不是真正的html，例如jsp本质其实是个servlet也就是一个java程序，所以它们的本质是服务端语言和html的一个整合技术，在实际运行中web容器会根据服务端的返回数据将jsp或模板引擎解析成浏览器能解析的html，然后传输这个html到浏览器进行解析。由此可见服务端语言提供的开发页面的技术其实是动静无法分离的源头，但是这些技术可以很好的完成动静资源中的动的内容，因此我们想做动静分离那么首先就要把静的资源从jsp或者模板语言里抽取出来，抽取出来的静态资源当然就要交给静态的web服务器来处理，我们常用的静态资源服务器一般是apache或ngnix，所以这些静态资源应该放置在这样的服务器上，那么我们是否可以在这些静态web服务器上做动静结合呢？答案是还真行，例如apache服务器有个模块就可以将它自身存储的静态资源和服务端传输的资源整合在一起，这种技术叫做ESI、SSI，这个时候我们可以把不变的静态内容制作成模板放置在静态服务器上，动态内容达到静态资源服务器时候，使用ESI或者SSI的标签，把动静内容结合在一起，这就完成了一个动静结合操作。

为什么我们要在服务端前面加个静态web服务器的道理。我个人觉得在每个服务端之前都布置一个静态web服务器，该服务器起到一个反向代理的作用，而且我觉得不管我们是否使用CDN，最好都这么做，这么做有如下好处：

好处一：方便日志的记录。

好处二：在服务端之前设立了一个安全屏障，即静态web服务器可以在必要时候过滤有害的请求。

好处三：可以控制流入到服务端的请求个数，当并发很高时候，可以利用静态web服务器能承担更高并发的能力来缓冲服务端的压力，这里我补充一些实践技巧，以java里常用的web容器tomcat为例，一般官方给出它的最大并发数应该不会超过200，如果我们在tomcat前放置了一个apache服务器，那么我们可以把tomcat的最大并发数设置为无效大，把并发数的控制放置在apache这边控制，这么做会给我们系统运维带来很大的好处，tomcat虽然有一个建议最大并发数，但是实际运行里java的web容器到底能承受多大并发其实要看具体场景了，因此我们如果可以动态控制apache的并发数，这个操作很方便的，那么我们就可以动态的调整tomcat这样容器的承载能力。

好处四：可以便于我们做动静分离。

SSI

这里我们以apache为例子讲解将动静分离前移到apache的一些做法，apache有一个功能叫做SSI，英文全称是Server Side Include，页面上我们一般这样使用SSI，SSI有一种标签，例如：

```
1 
```

页面一般使用注释的方式引入，这个和jsp的引入有点区别的，SSI的做法其实和服务端的引入类似，只不过使用SSI将本来服务端做的动静整合交由了apache完成了，我们可以把静态文件直接放置在Apache这里，如果这个静

态web服务器上升到CDN，那么这些静态资源就可以在靠近用户的地方使用，SSI说白了就是像apache这样的静态资源服务器接收到服务端返回后，将一部分内容插入到页面了，然后将完整页面返回至浏览器。这个做法如果优化的得当，可以很好的提升网站的加载效率。

ESI

Apache这样的静态资源服务器还支持一种动静整合的技术，这个技术就是ESi，它的英文全称叫做Edge Side Includes，它和SSI功能类似，它的用法如下所示:

```
1 <esi:includesrc="test.vm.esi?id=100"max-age="45"/>
```

它和SSI区别，使用esi标签获取的资源来自于缓存服务器，它和SSI相比有明显的性能优势，其实网页特别是一个复杂的网页我们做了动静分离后静态的资源本身还可以拆分，有的部分缓存的时间会长点，有点会短点，其实网页里某些动态内容本身在一定时间里有些资源也是不会发生变化的，那么这些内容我们可以将其存入到缓存服务器上，这些缓存服务器可以根据页esi传来的命令将各个不同的缓存内容整合在一起，由此我们可以发现使用esi我们会享受如下优点：

优点一：静态资源会存放在缓存里，那么获取静态资源的效率会更高。

优点二：根据静态资源的时效性，我们可以对不同的静态资源设置不同的缓存策略，这就增加了动静分离方案的灵活性。

优点三：缓存的文件的合并交由缓存服务器完成，这样就减少了web服务器本身抓取文件的开销，从而达到提升web服务器的并发处理能力，从而达到提升网站访问效率的目的。

关于ESI的更多内容请参考《大型网站之网站静态化（ESI）》

（2）CDN的动静合并

CDN其实也是一组静态的web服务器，那么我们是否可以把这些事情放到CDN做了？理论上是可以做到，但是现实却是不太好做，因为除了一些超有钱的互联网公司，大部分公司使用的CDN都是第三方提供的，第三方的CDN往往是一个通用方案，再加上人家毕竟不是自己人，而且CDN的主要目的也不是为了做动静分离，因此大部分情况下在CDN上完成这类操作并不是那么顺利，因此我们常常会在服务端的web容器前加上一个静态web服务器，这个静态服务器起到一个反向代理的作用，它可以做很多事情，其中一件事情就是可以完成这个动静结合的问题。

（3）ajax的动静合并

SSI和ESI是静态web服务器处理动静资源整合的手段,那么我们把这个动静结合点再往前推，推到浏览器，浏览器能做到这件事情吗？如果浏览器可以，那么静态资源也就可以缓存在客户端了，这比缓存在CDN效率还要高，其实浏览器还真的可以做到这点，特别是ajax技术出现后，浏览器来整合这个动静资源也就变得更加容易了。浏览器端的动静整合的技术称之为CSI，英文全称叫做Client Side Includes，这个技术就是时下javascriptMVC、MVVM以及MVP技术采取的手段，实现CSI一般是采用异步请求的方式进行，在ajax技术还没出现的年代我们一般采取iframe的方式，不过使用CSI技术页面加载就会被人为分成两次，一次是加载静态资源，等静态资源加载完毕，启动异步请求加载动态资源，这么一做的确会发生有朋友提到的一种加载延迟的问题，这个延迟我们可以使用适当的策略来解决的.

ajax是CSI的一种技术手段。不过一般而言，我们使用ajax做动静分离都是都是从服务端

请求一个html片段，到了浏览器后，使用dom技术将这个片段整合到页面里.

（4）javascriptMVC架构

虽然在浏览器使用ajax来进行动静分离和合并已经比全页面返回高效很多，但是他还是有问题的，服务端处理完请求最终返回结果其实都是很纯粹的数据，可是这些数据我们不得不转化为页面片段返回给浏览器，这本质是为纯粹的数据上加入了很多与服务端无用的结构，之所以说无用是因为浏览器自身也可以完成这些结构，为什么我们一定要让服务端做这个事情了？如是乎JavaScript的模板技术出现了，这些模板技术和jsp，velocity类似，只不过它们是通过javascript设计的模板语言，有了javascript模板语言，服务端可以完全不用考虑对页面的处理，它只需要将有效的数据返回到页面就行了，使用了javascript模板技术，可以让我们动静资源分离做的更加彻底，基本上所有的浏览器相关的东西都被静态化了，服务端只需要把最原始的数据传输到浏览器即可。讲到这里我们就说到了web前端最前沿的技术了：javascriptMVC架构了。关于此的更多内容请参考《大型网站之网站静态化（javascriptMVC架构）》

六、Web前端优化

关于Web前端优化的请参考《Web前端优化最佳实践及工具集锦》，《探真无阻塞加载javascript脚本技术》，《【Web优化】Yslow优化法则（汇总篇）》

七、总结

在web开发里，除了需要浏览器处理的，其他技术都可以当做服务端来理解，如果我们网站使用到了CDN，使用到了静态web服务器例如apache，以及服务端的web容器例如jboss，那么按请求的行进路径，我们结果处理越早那么网站响应效率也就越高，所以当请求在CDN返回了，那么肯定比在apache返回效率高，在apache就返回了肯定比jboss返回的效率高，再则服务端的web容器本身因为服务端程序运行要消耗部分系统资源，所以它在处理请求的效率会比CDN和apache差很多，所以当我们按照动静分离策略拆分出了静态资源后，这个资源能不放在最底层的服务端的web容器处理就不要放在服务端的web容器里处理。