**摘要**

 随着多媒体技术普及和迅速发展，人机交互的研究重点放在了虚拟交互、智能化交互和人机协同交互等以人为在中心的人机交互技术方面。

基于图像处理的运动侦测、纹理映射等理论，设计出能通过手指触摸或激光笔照射来操作计算机的新型投影系统，实现低成本、高性能的多点触摸人机交互技术。

**Abstract**

With the popularization and rapid development of multimedia technology, human-computer interaction research focuses on the virtual interaction, intelligent interaction and human-computer interaction and so on.

Interactive projection system based on the current use of technology, often have very high cost, slow reaction speed positioning, and projection surface material shortcomings such as limited, and most of the products can only use a pen or pencil to infrared electromagnetic control, can not be achieved can be multi touch control by hand, and therefore can not bring good experience to the user. The invention can solve the problems above.

The idea of works: this work is the optical principle of multi touch technology, multi touch tracking, gesture recognition based on the theory, design a new projection system can operate the computer through the touch of a finger or pen laser irradiation, low cost and high performance multi touch human-computer interaction technology. At the same time, it is applied to the augmented reality, which is a new field of virtual reality technology, and is connected with the computer network to realize the information exchange function of the Internet of things.

Compared with the traditional projection system, this work has many functions such as multi touch, remote operation, augmented reality and Internet of things. At the same time, the equipment and materials used in this work are more selective and the system is easy to construct. This work can be widely used in home entertainment, advertising, education and so on many occasions the meeting, according to the needs of different occasions, the flexibility to choose the type and size of equipment and materials to build the system, and change their relative position. The market analysis and economic benefits of this work: 1 create a new game based on the touch screen in the entertainment, to guide the new wave of electronic games. 2 in the application of advertising, so that the audience can independently choose to learn more information for the propagandist to add more customers. 3 in the meeting of the application of education, so that the whole process of communication between people more convenient, improve work efficiency. 4 in the Internet of things, the use of augmented reality technology will be superimposed on the virtual information projection on the real object, through the computer network connection, the realization of the Internet of things, "human" information interaction.

目录

[1 绪论 4](#_Toc479420751)

[1.1 课题背景 5](#_Toc479420752)

[1.2课题意义 5](#_Toc479420753)

[2系统的原理 5](#_Toc479420754)

[2.1 opencv 5](#_Toc479420755)

[2.2 运动检测 6](#_Toc479420756)

[2.3 纹理映射 6](#_Toc479420757)

[3 运动检测 6](#_Toc479420758)

[3.1背景减除 6](#_Toc479420759)

[3.2 时间差分 6](#_Toc479420760)

[3.3光流 7](#_Toc479420761)

[4 纹理映射 7](#_Toc479420762)

[5 系统实现 11](#_Toc479420763)

[5.1系 11](#_Toc479420764)

[5.2后台逻辑设计及功能模块实现 11](#_Toc479420765)

[6 系统测试与运行结果 11](#_Toc479420766)

[6.1 客户模块 11](#_Toc479420767)

[6.2 后台模块 11](#_Toc479420768)

[致谢辞 11](#_Toc479420769)

[参考文献 12](#_Toc479420770)

[附录1：英文文献原文 13](#_Toc479420771)

[附录2：英文翻译原文 18](#_Toc479420772)

# 1 绪论

显示屏的触摸交互可以说是科技史上最伟大的发明之一，特别是进入移动互联网时代之后，[智能手机](http://www.leiphone.com/tag/%E6%99%BA%E8%83%BD%E6%89%8B%E6%9C%BA)、平板电脑等触控设备带有的触摸界面交互给人们的生活带来了无数便利，改变了人们的生活方式。然而，无论是[智能](http://www.leiphone.com/)手机、平板电脑还是车载屏幕等其他的触屏设备，在发展过程中都出现了瓶颈——绕不开「屏幕」这层硬件。所以，人们又开始对新的界面交互方式进行了探索，目前已经有几方面有了新的进展，比如虚拟现实和增强现实，再比如投影交互。投影界面交互已经不是什么特别新鲜的技术了，很多做投影仪的厂商都已经将其融入到现有的产品中推向市场，并做出了一些可观的成绩，但是其中很多产品的用户体验并不是很好。

随着科技的发展，人们对高新科技设备的要求越来越高，液晶触摸屏已经不能满足人们多种多样的需求，所以类似VR/AR和投影交互等成为了人们探索下一代界面触控交互的方式，而这几种方式都各有优势，短期的发展来看并不能互相取代。一个拥有不错用户体验的投影交互设备会被应用到诸多领域，成为下一代新式界面交互的主力军。

多媒体互动投影系统采用先进的计算机视觉技术和投影显示技术实现了参与者与投射在地面上的影像的真实互动，产生奇幻动感的交互体验。互动投影系统打破了传统静态和影视广告毫无互动性的传统风格，将互动游戏与广告完美结合，在吸引广大受众参与互动或参观的同时，起到更好的广告宣传效果。互动投影系统可在影像区内产生各种特效，参与者可以直接使用身体动作与影像区内的图像及虚拟场景进行交互，各种互动效果就会随着动作产生相应的变幻，让参与者进入虚实融合、亦真亦幻的奇妙世界。例如：鱼儿随着参与者的脚步在虚拟水面上泛起的波纹或四处游散或游聚一起；花朵随着参与者的手势在虚拟的草地上绽放；海龟被参与者的脚步踢到四处游开等，都能带给观众一种全新趣味性的互动体验。

本项目的优点：1.在娱乐方面创作出基于触摸投影的新型游戏，引导电子游戏的新风潮。 2.在广告宣传的应用，使观众能自主有选择地了解更多信息，为宣传者增加更多的顾客。 3.在会议教育的应用，使整个过程中人与人之间的交流更为便捷，提高工作效率。 4.在物联网方面，利用增强现实技术将虚拟信息投影叠加到真实物体上，通过计算机网络连接，实现物联网的“人—物”信息交互功能。

**关键字**：交互、物联网、多媒体

## 1.1 课题背景

互动投影系统根据目前所使用的技术，往往有成本太高、定位不准、反应速度慢、投影面材质局限等缺点，同时大多数产品只能利用红外笔或电磁笔去控制，无法实现用手进行可多点触摸控制，因此无法给用户带来很好的体验。

## 1.2课题意义

与传统投影系统相比，本项目具有多点触摸、远距操作、增强现实、物联网的强大功能，具有崭新的人机交互理念。同时本项目所用到的设备、材料选择性多，系统搭建容易。 本项目可广泛应用于家庭娱乐、广告宣传、会议教育等众多场合，根据各个场合的需要，可灵活选择搭建本系统的设备、材料的种类和型号，并改变它们的相对位置。

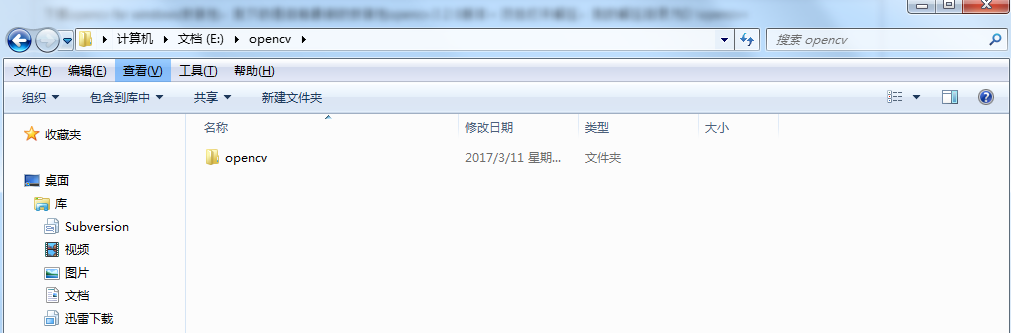
# 2系统的准备

## 2.1 opencv

OpenCV的[全称](http://baike.baidu.com/item/%E5%85%A8%E7%A7%B0" \t "_blank)是：Open Source Computer Vision Library。OpenCV是一个基于BSD许可（开源）发行的跨平台计算机视觉库，可以运行在Linux、Windows、Android和Mac OS操作系统上。它轻量级而且高效——由一系列 C 函数和少量 C++ 类[构成](http://baike.baidu.com/item/%E6%9E%84%E6%88%90/103686" \t "_blank)，同时提供了Python、Ruby、MATLAB等语言的接口，实现了[图像处理](http://baike.baidu.com/item/%E5%9B%BE%E5%83%8F%E5%A4%84%E7%90%86" \t "_blank)和计算机视觉方面的很多通用算法。

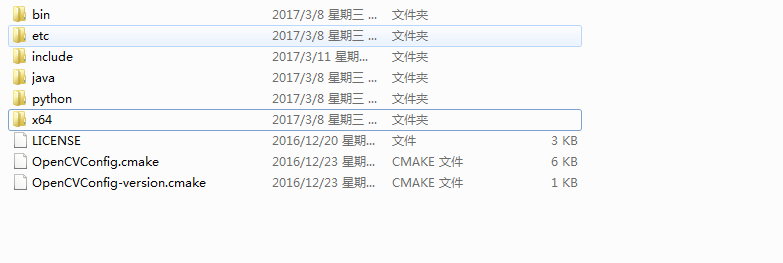
     OpenCV（Open Source Computer Vision Library）。OpenCV是一款跨平台计算机视觉库，可以运行在Linux、Windows和Mac OS操作系统上。

下载opencv for windows安装包，我下的是目前最新的安装包opencv-3.2.0版本。双击打开解压，我的解压目录为D:\opencv。



文件中有build和sources两个文件夹

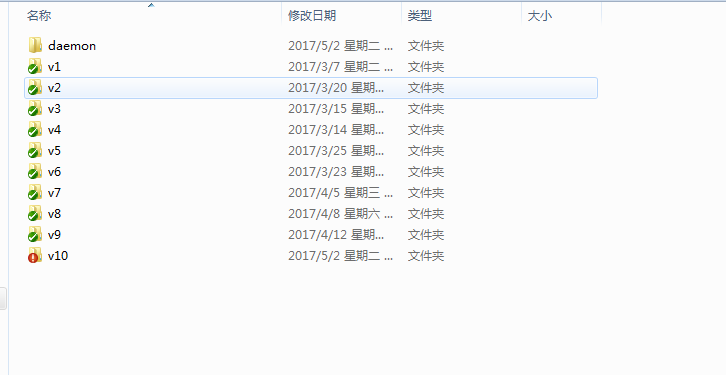
build中是编译后的依赖库，source中放有程序的源代码



现在的opencv支持java开发c开发还有python27版本的开发。

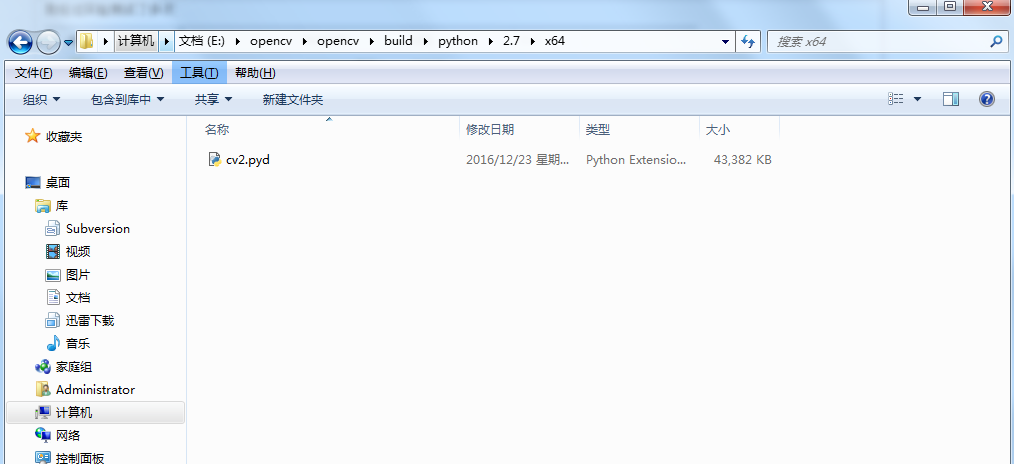
所以opencv完全可以在常见的平台都可以进行开发，我之前采用python的版本进行开发，

我经过实验测试了多次



最终用c++开发出最后版本。

在python开发中需要知道numpy这个类型，numpy是python一个快速处理矩阵运算的库，配合opencv的cv2.pyd



cv2.pyd应该放到python安装目录里的

### [sit-packages](http://undefined/)

### 里这样可以保证python程序在编译的时候可以找到cv2这个类库，python是个强大的脚本语言，开发迅速，但是安装麻烦，所以最后我还是用vc重写了下。

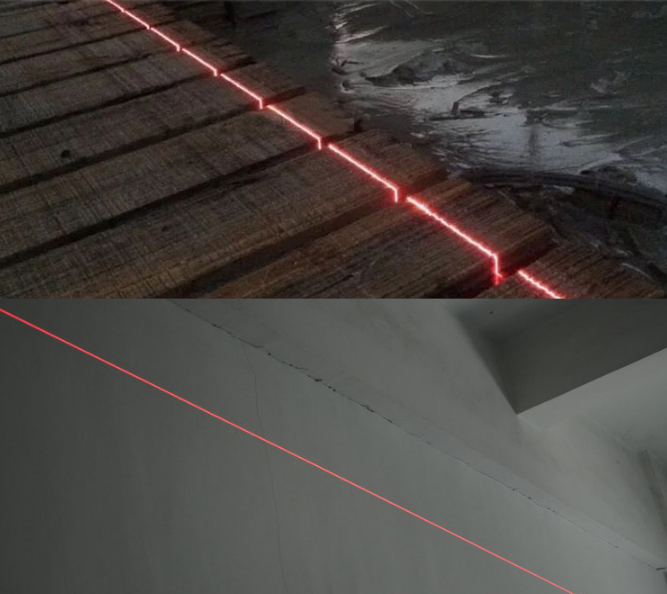
同样opencv也支持java的开发，这样他就可以在安卓等设备上供人开发使用。

## 2.2 运动检测

### 这些都是我初期做的版本，初期的反应并不灵敏，但是可以产生出粗糙的效果于是我想怎么把产生的位置转化成响应。第一个想到的是进程通讯，服务器端产生位置坐标并向连接服务器的客户端发送坐标，然后客户端将受到的数据通过win32的api转化成鼠标事件，产生响应，

初期的图像处理识别比较简单，依靠的是人体可以反射红外信号的特点，我只要将红外摄像头采集的图像作为我的图像输入源。然后逐帧分析图像，经过灰度化阈值处理后提取位置坐标，只是这种方法受投影机的的影响太大。强行调节阈值可能什么都获取不到。

于是我在网上看了下别人的产品，他们基本用的用一字线性激光打到墙面上，



安装就是平行打到墙上，一旦有物体在墙上阻挡了光线就会产生红光，这个光用肉眼看不到在红外摄像头下会特比明显，这样我将图像进行直方图均衡化，增强对比度，然后设置一个阈值可以比较准确的找到投影中手的位置。这种方案如果只打一道红外是不能多点响应的，可能前面的点会挡住后面的点，所以可以多打几道一字线性激光，但是这个方案的成本不便宜，一个一字线性激光300多，觉 的并不合适。

最后想到的是运动侦测，相比光流法，背景减除，最后决定用时间差分的方法，也就是侦差法，

时间差分（又称相邻帧差）方法是在连续的图像序列中两个或三个相邻帧间采用基于像素的时间差分并且阈值化来提取出图像中的运动区域。时间差分运动检测方法对于动态环境具有较强的自适应性，但一般不能完全提取出所有相关的特征像素点，在运动实体内部容易产生空洞现象。

不过空洞的现象基本不会出现（不能排除摄像头自身的原因），在时间差设置的足够小的情况下会有比较好的效果。差分的结果产生一个新的矩阵通过阈值处理可以找出人手的大体位置，在经过轮廓寻找就可以找到物体的大致位置。这种方法可以省掉昂贵的设备成本，只是依靠运动去侦测。

最后的方法采用的是三联帧差的方法，逻辑和时间差分类似，设所有帧都经过灰度化二值化后，把当前帧设为上一帧，当前帧与之前的帧差分出a ，下一帧与当前帧差分出b，然后a与b相交（即：矩阵每个位置对应进行且运算）最后的结果在进行轮廓寻找找到图像的位置。

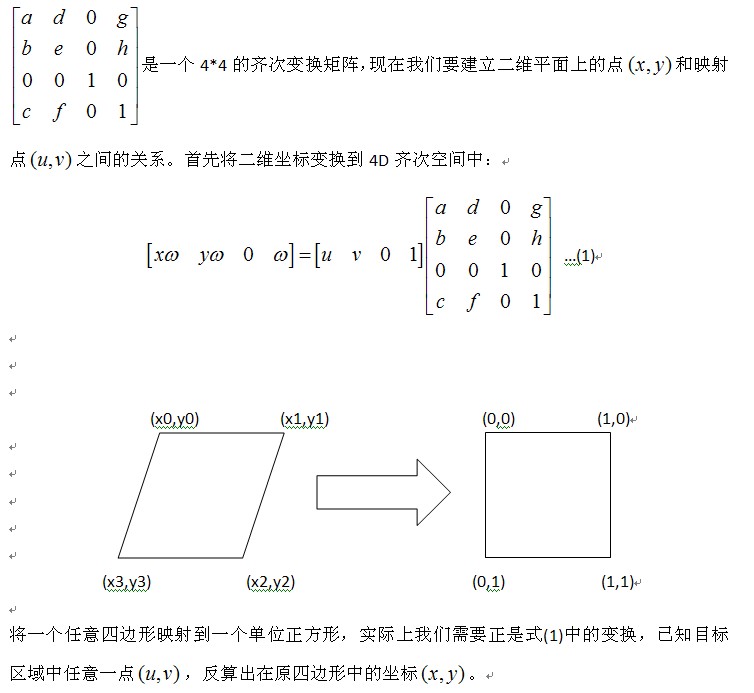
## 2.3 纹理映射

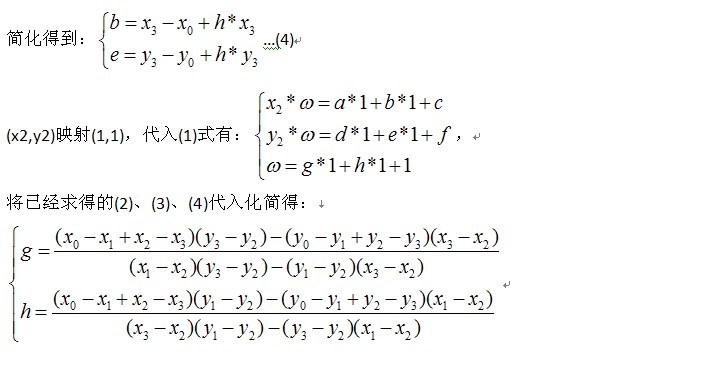
纹理映射（Texture Mapping）是将纹理空间中的纹理像素映射到屏幕空间中的像素的过程。

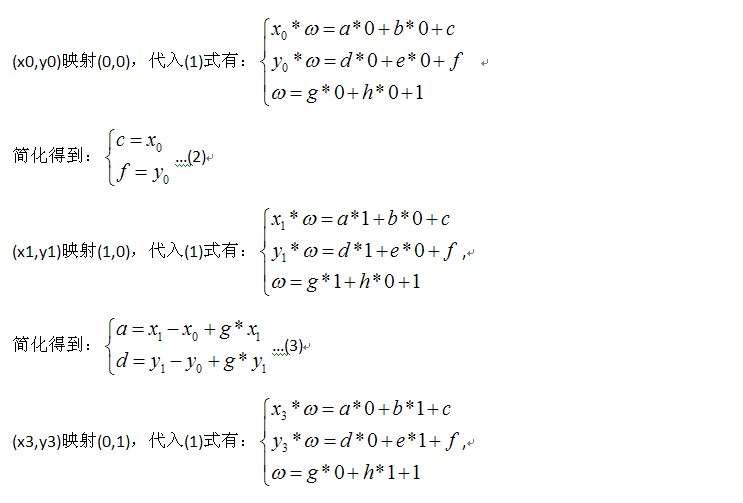
# 用反向映射来解这个问题

u = (ax + by + c)/(gx + hy + 1);  
v = (dx + ey + f)/(gx + hy + 1);

# 可以将一个单位矩阵映射到不规则四边形，然后根据四边形的点求出他在单位矩阵的位置。

建立8元8次方程求出abcdefgh  




**求逆结果**

y=[(fg-d)u+(a-cg)v+cd-af]/[(hd-eg)u+(bg-ah)v+ae-bd]  
x=[(e-fh)u+(ch-b)v-ec+bf]/[(hd-eg)u+(bg-ah)v+ae-bd]

# 3 运动检测

## 3.1背景减除

## 3.2 时间差分

## 3.3光流

# 

# 4 纹理映射

纹理映射（Texture Mapping）是将纹理空间中的纹理像素映射到屏幕空间中的像素的过程。

c++不能自己判断头文件是否引入，所以使用ifndef 判断文件重复引入，这个类是的二维纹理映射的代码，主要逻辑就是

将一个不规则的四边形在一个二维空间单位化，然后在已知屏幕的尺寸或者自定义尺寸后可以实现单位坐标到最终坐标的转化。

#ifndef I\_AM\_NOT\_REDEFINITION

#define I\_AM\_NOT\_REDEFINITION

class AKWPosition {

public:

double a;

double b;

double c;

double d;

double e;

double f;

double g;

double h;

double width;

double height;

//坐标处理

void init(double x0, double y0, double x\_1, double y\_1, double x\_2, double y\_2, double x\_3, double y\_3, double ww, double hh) {

width = ww;

height = hh;

double tmp\_h, tmp\_g;

x0 = x0\*1.0;

y0 = y0\*1.0;

x\_1 = x\_1\*1.0;

y\_1 = y\_1\*1.0;

x\_2 = x\_2\*1.0;

y\_2 = y\_2\*1.0;

x\_3 = x\_3\*1.0;

y\_3 = y\_3\*1.0;

c = x0;

f = y0;

g = (x0 - x\_1 + x\_2 - x\_3)\*(y\_3 - y\_2) - (y0 - y\_1 + y\_2 - y\_3)\*(x\_3 - x\_2);

tmp\_g = (x\_1 - x\_2)\*(y\_3 - y\_2) - (y\_1 - y\_2)\*(x\_3 - x\_2);

g = g / tmp\_g;

h = (x0 - x\_1 + x\_2 - x\_3)\*(y\_1 - y\_2) - (y0 - y\_1 + y\_2 - y\_3)\*(x\_1 - x\_2);

tmp\_h = (x\_3 - x\_2)\*(y\_1 - y\_2) - (y\_3 - y\_2)\*(x\_1 - x\_2);

h = h / tmp\_h;

b = x\_3 - x0 + h\*x\_3;

e = y\_3 - y0 + h\*y\_3;

a = x\_1 - x0 + g\*x\_1;

d = y\_1 - y0 + g\*y\_1;

}

//返回单位坐标

void poi(double &u, double &v) {

double x, y;

y = ((f\*g - d)\*u + (a - c\*g)\*v + c\*d - a\*f) / ((h\*d - e\*g)\*u + (b\*g - a\*h)\*v + a\*e - b\*d);

x = ((e - f\*h)\*u + (c\*h - b)\*v - e\*c + b\*f) / ((h\*d - e\*g)\*u + (b\*g - a\*h)\*v + a\*e - b\*d);

if (x < 0 || y < 0 || x>1 || y>1) {

u = 0;

v = 0;

}

u = x;

v = y;

}

void z\_poi(double &u, double &v,int x0,int y0,int ww,int hh) {

u = int(x0 + ww\*u);

v = int(y0 + hh\*v);

}

};

#endif

python代码

def textture(x0,y0,x1,y1,x2,y2,x3,y3):

global a

global b

global c

global d

global e

global f

global g

global h

x0 = x0\*1.0

y0 = y0\*1.0

x1 = x1\*1.0

y1 = y1\*1.0

x2 = x2\*1.0

y2 = y2\*1.0

x3 = x3\*1.0

y3 = y3\*1.0

c = x0

f = y0

g = (x0-x1+x2-x3)\*(y3-y2)-(y0-y1+y2-y3)\*(x3-x2)

tmp\_g = (x1-x2)\*(y3-y2)-(y1-y2)\*(x3-x2)

print x1,x2,y3,y2,y1,x3,x2

g = g/tmp\_g

h = (x0-x1+x2-x3)\*(y1-y2)-(y0-y1+y2-y3)\*(x1-x2)

tmp\_h = (x3-x2)\*(y1-y2)-(y3-y2)\*(x1-x2)

h = h/tmp\_h

b = x3-x0+h\*x3

e = y3-y0+h\*y3

a = x1-x0+g\*x1

d = y1-y0+g\*y1

print a,b,c,d,e,f,g,h

def position(u,v):

y=((f\*g-d)\*u+(a-c\*g)\*v+c\*d-a\*f)/((h\*d-e\*g)\*u+(b\*g-a\*h)\*v+a\*e-b\*d)

x=((e-f\*h)\*u+(c\*h-b)\*v-e\*c+b\*f)/((h\*d-e\*g)\*u+(b\*g-a\*h)\*v+a\*e-b\*d)

if(x<0 or y<0 or x>1 or y>1):

return None

return (x,y)

def view\_position(a0,b0,x\_m,y\_m,x,y):

x = a0+x\_m\*x

y = b0+y\_m\*y

return (x,y)

if \_\_name\_\_ == "\_\_main\_\_":

textture(10,10,20,10,20,20,10,30);

print position(18,19)

# 5 系统实现

## 5.1系

## 5.2后台逻辑设计及功能模块实现

# 6 系统测试与运行结果

## 6.1 客户模块

## 6.2 后台模块

# 致谢辞

本论文是在孙海滨老师的悉心指导下完成的。孙老师严肃的科学态度，严谨的治学精神，精益求精的工作作风，深深地感染和激励着我。从课题的选择到项目的最终完成，孙老师都始终给予我细心的指导和不懈的支持。导师不仅给我提供了一个做毕业设计的良好环境，而且在做毕业设计的过程中，导师无时无刻不在指导我学习，让我受益匪浅。经过这次毕业设计，我感到自己无论在专业知识方面，还是在动手能力方面都有了很大的收获。不仅接触到了许多新的技术和知识，而且通过亲手实践，了解了如何把书本上所学的东西应用到实践中去。

在本次毕业设计过程中我除了要衷心的感谢孙老师的外，还要感谢同组同学们的支持与帮助。感谢同学们在大学期间对于我的关怀，以及对我的支持，让我顺利完成大学四年的学习生活。感谢许多同学在百忙之中对于我毕业设计的点拨和指导，帮助我开拓研究思路。在他们的帮助下，我的毕业设计才能顺利完成，毕业论文结构更加完善，内容更加充实。在此我衷心的感谢你们！

# 参考文献

[1] Aaron.Professional PHP Design Patterns Saray[M].WROX PR/PEER INFORMATION INC ,2009.8

[2] Buttle, F. (2004). Customer Relationship Management concepts and tools.

[3] 李宇.JavaScript网页特效实例解析[M].机械工业出版社,2013.4

[4]（美）赞德斯彻.深入PHP：面向对象、模式与实践(第3版)[M].人民邮电出版社,2011.7

[5]张丽萍,陈晓娟等.基于PHP技术的酒店管理系统应用研究[J].兰州交通大学学报,2010(1):3-5

[6] 林晓霞 刘太安．数据库技术及应用．中国石油大学出版社．2009.1

[7] 胡菘.Dreamweaver完美网页设计[M].中国青年电子出版社,2010.5

[8] 赵启志.PHP4+MySQL完整自学方案[M].中国铁道出版社,2011.9

# 附录1：英文文献原文

It’s hard, if not impossible, to deny that performance is by far one of the most critical aspects of any decent web project, be it a small portfolio site, a mobile-first web app, right through to a full-scale ecommerce project. Studies, articles and personal experience all tell us that fast is best.

Performance is not only hugely important, it is incredibly interesting, and something I am getting more and more involved in at both work (I’m forever pestering our Lead Performance Engineer) and in side projects and CSS Wizardry (I’m forever pesteringAndy Davies).

I’m going to share – in this massive article – a load of quick, simple and downright intriguing bits of perf knowledge to act as a primer for web designers and front-end developers alike; hopefully this article will serve as a decent introduction for anyone wanting to start learning about perf, and making their front-ends blazingly fast. These tips are all things you can implement by yourself very easily. It just takes a bit of cunning and some basic knowledge of how browsers work and you’re ready to game the system!

This huge post won’t have loads of confusing graphs and numbers to crunch, but instead concerns itself with theory and first-hand performance techniques that I have arrived at as a result of reading, monitoring, collaborating and tinkering (I spend a lot of time glued to CSS Wizardry’s waterfall charts). I will also link to other articles on similar topics to help reinforce any key points. Enjoy!

N.B. This article does require a small amount of basic performance knowledge up-front, but anything covered that you aren’t familiar with should be just a Google search away.

There are a few things all designers and front-end developers will likely know about performance, things like making as few requests as possible, optimising images, putting stylesheets in the <head>, putting JS before the </body>, minifying JS and CSS and so on. These fundamentals will already get you on your way to faster experiences for users, but there’s more… much more.

It is also very important to remember that – for all they give us headaches every day of our working lives – browsers are very clever; they do a lot to optimise performance for you, so a lot of perf knowledge combines knowing where the browser is at work, and knowledge of how best to exploit that. A lot of perf know-how is merely understanding, exploiting and manipulating what a browser does for us already.

Styles at the top, scripts at the bottom

This is a really basic rule, and one that should be super easy to follow most of the time, but why does it matter? Put very shortly:

CSS blocks rendering because of a browsers desire to render pages progressively; they want to render things as they get to them, and in order. If styles are a long way down the page the browser can’t render that CSS until it gets to it. This is so that the browser can avoid redraws of styles if they alter something that was previously rendered further up the document. A browser won’t render a page until it has all the available style information, and if you put that style information at the bottom of the document you’re making the browser wait, blocking rendering.

So, you put your CSS at the top of the page so that the browser can start rendering right away.

JavaScript blocks downloads for a number of reasons (this is the browser being clever again) but firstly, we need to know how downloading assets in browsers actually happens; simply put, a browser will download as many assets as it can from a single domain in parallel. The more domains it is pulling from, the more assets can be downloaded, in parallel, at once.

JavaScript interrupts this process, blocking parallel downloads from any and all domains, because:

So, because browsers stop all other downloads whilst JavaScript is being fetched, it is usually a good idea to put your JavaScript as late in the document as possible. I’m sure you’ve all seen blank sections of pages where a third party piece of JS is taking ages to load and blocking the fetching and rendering of the rest of the page’s assets; this is JavaScript’s blocking in action.

How I helped the NHS rapidly build a brand new product.

Read case study…

Apparently, however, modern browsers get smarter still. I’m going to give you an excerpt from an email from Andy Davies to me, because he explains far better than I can:

Modern browsers will download JS in parallel and only rendering is blocked until the script has been executed (it obviously has to be downloaded too).

Downloading of the script will often be done by the browser’s look ahead pre-loader.

When a browser is blocked from rendering page e.g. waiting for CSS, or JS to execute, the look ahead pre-parser scans the rest of the page looking for resources it could download.

Some browsers e.g. Chrome, will prioritise the download of assets e.g. if scripts and images are both waiting to be downloaded it will download the script first.

Smart stuff!

So, to allow a page to begin rendering as fast as possible, put your styles at the top. To prevent JS’ blocking affecting your rendering, put scripts at the bottom.

Make fewer requests

The other really obvious and basic performance optimisation is simply downloading less. Every asset a page requires is an extra HTTP request; the browser has to go off and get every single asset required to render a page. Each of these requests can incur DNS lookups, redirects, 404s etc. Every HTTP request you make, whether it is for a stylesheet, an image, a web font, a JS file, you name it, is a potentially very expensive operation. Minimising these requests is one of the quickest performance optimisations you can make.

Going back to browsers and parallelisation; most browsers will only download a handfull of assets from each referenced domain at a time, and JS, remember, will block these downloads anyway. Every HTTP request you make should be well justified, and not taken lightly.

Maximising parallelisation

In order to get the browser to download more assets in parallel, you can serve them from different domains. If a browser can only fetch, say, two assets at once from a domain, then serving content from two domains means it can fetch four assets at once; serving from three domains means six parallel downloads.

A lot of sites have static/asset domains; Twitter, you can see, use si0.twimg.com to serve static assets:

<link rel="stylesheet" href="https://si0.twimg.com/a/1358386289/t1/css/t1\_core.bundle.css" type="text/css" media="screen">

Facebook use fbstatic-a.akamaihd.net:

<link rel="stylesheet" href="https://fbstatic-a.akamaihd.net/rsrc.php/v2/yi/r/76f893pcD3j.css">

Using these static, asset domains, Twitter and Facebook can serve more assets in parallel; assets from twitter.com and si0.twimg.com can be downloaded in tandem. This is a really simple way to get more concurrent downloads happening on your page, and even better when coupled with actual CDN technology that can help decrease latency by serving assets from a more suitable physical location.

This is all well and good, but later we’ll discuss how serving from subdomains can actually, in certain circumstances, be detrimental to performance.

So, these are our performance basics out of the way:

Every time you request an asset from any domain, out goes an HTTP request with the relevant headers, the resource is reached, and a response is sent back. This is a vast over-simplification of the process, but it’s about as much as you really need to know. This is an HTTP request, and all assets you reference are subject to this round trip. These requests are the main bottleneck when it comes to front-end performance because, as we covered, browsers are limited by how many of these requests can happen in parallel. This is why we often want to use subdomains; to allow these request to happen on several domains, allowing a greater number of requests to happen at the same time.

A problem with this, however, is DNS lookup. Each time (from a cold cache) a newdomain is referenced, the HTTP request is subject to a time-consuming DNS lookup (anywhere between 20 and 120 milliseconds) in which the outgoing request looks up where the asset actually lives; the internet is tied together by IP addresses which are referenced by hostnames which are managed by DNS.

If each new domain you reference has the upfront cost of a DNS lookup, you have to be sure that it’s actually going to be worth it. If you are a small site (like CSS Wizardry, for example) then serving assets from a subdomain will likely not be worth it; the browser can probably fetch several under-parallelised assets from one domain quicker than it can perform DNS lookups across multiple domains and parallelise those.

If you have perhaps a dozen assets, you might want to consider serving them fromone subdomain; an extra DNS lookup is probably worth it in order to better parallelise that amount of assets. If you have, say, 40 assets, it might be worth sharding those assets across two subdomains; two extra DNS lookups will be worth it in order to serve your site from a total of three domains.

DNS lookups are expensive, so you need to determine which is more suitable for your site; the overhead of lookups or just serving everything from one domain.

It is important to remember that as soon as the HTML is requested from, say,foo.com, that DNS lookup for that host has already happened, so subsequent requests to anything on foo.com are not subject to DNS lookups.

DNS prefetching

If you, like me, want to have a Twitter widget on your site, and Analytics, and maybe some web fonts, then you will have to link to some other domains which means you’llhave to incur DNS lookups. My advice would always be not to use any and every widget without properly considering its performance impact first, but for any you do deem necessary, the following is useful…

Because these things are on other domains it does mean that, for example, your web font CSS will download in parallel to your own CSS, which is a benefit in a way, but scripts will still block (unless they’re async).

The problem here, really, is the DNS lookups involved with third party domains. Fortunately, there is a super quick and easy way to speed this process up: DNS prefetching.

DNS prefetching does exactly what is says on the tin, and could not be simpler to implement. If you need to request assets from, say, widget.foo.com, then you can prefetch that hostname’s DNS by simply adding this early on in the <head> of your page:

<head>

...

<link rel="dns-prefetch" href="//widget.foo.com">

...

</head>

That simple line will tell supportive browsers to start prefetching the DNS for that domain a fraction before it’s actually needed. This means that the DNS lookup process will already be underway by the time the browser hits the <script> element that actually requests the widget. It just gives the browser a small head start.

This simple link element (which I use on CSS Wizardry) is totally backward compatible and will not negatively impact performance. Think of it like performance progressive enhancement!

# 附录2：英文翻译原文

这很难，如果不是不可能的，否认性能是迄今为止任何像样的web项目中最重要的一个方面，是它的一小组合的网站，移动优先的Web应用程序，再到全面的电子商务项目。研究报告，文章和个人经验都告诉我们，快是最好的。

性能不仅非常重要的，这是令人难以置信的有趣，什么我都在工作中越来越多地参与（我永远缠着我们的领先性能工程师）和侧项目和CSS巫术。

我想分享 - 这个巨大的文章 - PERF的知识快速，简单，有趣透顶位作为网页设计师和前端开发人员提供引物的负荷;希望这篇文章将作为任何人想要开始学习PERF，并且使他们的前端极快一个像样的介绍。这些技巧，你可以自己很轻松地实现所有的东西。它只是需要一点狡猾和的浏览器是如何工作的一些基本知识，你准备到游戏系统！

这个巨大的职位不会有混乱的图表和数字紧缩的负荷，而是关注的是，我在到达时的读出结果的理论和第一手的表现手法，监控，协作和修补（我花了很多紧盯CSS巫术的瀑布图的时间）。我也将链接到类似主题的其他文章，以帮助加强所有关键点。请享用！

N.B.这篇文章确实需要的基本性能知识，前期量小，但任何涉及你不熟悉的应该只是谷歌搜索了！

样式的顶部，脚本在底部

基础

有几件事情所有的设计师和前端开发人员可能会知道有关性能，像使尽可能少的要求成为可能，优化图像，将样式表中的<head>，把JS在</ body>之前，与缩小JS和CSS等。这些基本面将得到已经在你自己的方式，为用户提供更快速的经验，但还有更多。

这也是非常重要的是要记住， - 所有他们给我们头疼我们的工作中的每一天的生活 - 浏览器是非常聪明;他们做了很多为你优化性能，所以很多PERF的知识结合知道哪里是浏览器在工作，以及如何最好地利用这些知识。很多PERF诀窍仅仅是认识，利用和操纵是什么浏览器为我们做了。

样式的顶部，脚本在底部

这是一个非常基本的规则，一个应该是超级容易遵循的大部分时间，但它为什么重要？把很快：

CSS渲染块，所以你需要马上处理它（即在文档的顶部，在你的<head>）。

JS块下载，所以你需要对付这些最后，以确保它们不持有任何东西在页​​面上的其他人。

CSS块，因为浏览器的愿望，呈现逐步页面渲染;他们要渲染的东西，因为他们得到它们，而为了。如果样式是很长的路要走下来的页面，直到它得到它的浏览器无法呈现CSS。这使浏览器能避免重画的风格，如果他们改变以前进一步渲染了文件的东西。浏览器将不会呈现一个页面，直到它具有所有可用样式信息，如果你把这种风格在信息你让浏览器等待，堵渲染文件的底部。

所以，你把你的CSS在网页的顶部，这样浏览器就可以开始渲染的时候了。

JavaScript的块有许多原因下载（这是浏览器再次被聪明），但首先，我们需要知道如何在浏览器中实际发生的下载资产;简单地说，浏览器将并行单个域下载尽可能多的资产，因为它可以。所述多个结构域是从拉时，更多的资产可以下载，并联，在一次。

JavaScript的中断这个过程中，阻止任何和所有域并行下载，这是因为：

该脚本被称为可能会改变页面，这意味着浏览器将不得不面对它面前可以移动到任何东西。为了使它处理这一不测事件将停止下载任何东西，以soleley专注于这一点。

脚本通常需要以某种顺序加载为他们工作，例如，装载jQuery的加载一个插件之前。阻止浏览器使用JavaScript并行下载，以便它不启动在同一时间下载jQuery和你的插件;它的jQuery会之前应该是很明显的，如果你要开始下载两个并行，你的插件会到达。

所以，因为浏览器停止所有其他的下载，而JavaScript是被提取，它通常是一个好主意，把你的JavaScript作为后期文档作为可能的。我敢肯定，你都看到哪里第三方一块JS走的是年龄的负荷和阻塞的抓取和页面的资产，其余的渲染页面的空白部分;这是JavaScript的行动阻止。

精选案例：NHS

显然，然而，现代的浏览器变得更加聪明还。我要给你从安迪·戴维斯的电子邮件的摘录给我，因为他说远远超过我可以更好地：

现代浏览器会下载JS在并列的，唯一的渲染被阻塞，直到脚本已执行（这显然有过于下载）。

脚本的下载往往会被浏览器的外观超前预加载器来完成。

当浏览器从页面呈现受阻例如等待CSS，JS还是要执行，该预预解析器扫描页面的其他部分寻找它可以下载资源。

有些浏览器例如Chrome浏览器，例如将资产的优先下载如果脚本和图像都等待下载它首先会下载脚本。

所以，让一个页面开始渲染尽可能快，把你的风格在顶部。为了防止JS'影响你的渲染封锁，把脚本在底部。

其他的真的很明显和基本性能优化，简单地下载少。每一个资产页需要一个额外的HTTP请求;浏览器已熄灭，并得到渲染页面所需的每一个资产。所有这些请求都可以承担的DNS查找，重定向，404等。每个HTTP请求您，无论是样式表，图像，Web字体，JS文件，你的名字，是一个潜在的非常昂贵的操作。尽量减少这些请求是最快的性能优化技术可以使之一。

让我们回到浏览器和并行化;大多数浏览器只会在每次下载然后从每个引用域资产的福，和JS，记住了，反正会阻止这些下载。你让每个HTTP请求应该是天经地义的，并不敢掉以轻心。

最大化并行化

为了获取浏览器的并行下载更多的资产，你可以从不同的领域为他们服务。如果浏览器只能读取，也就是说，在从一个域，一旦两种资产，从两个域再服内容意味着它可以一次提取4个资产;从三个领域服务意味着6个分下载。

很多网站都有静态/资产域; Twitter的，你可以看到，使用si0.twimg.com服务静态资产：

<链接rel =“stylesheet”属性HREF =“htt​​ps://si0.twimg.com/a/1358386289/t1/css/t1\_core.bundle.css”类型=“文/ CSS”媒体=“屏幕”>

Facebook的使用fbstatic-a.akamaihd.net：

<链接rel =“stylesheet”属性HREF =“htt​​ps://fbstatic-a.akamaihd.net/rsrc.php/v2/yi/r/76f893pcD3j.css”>

使用这些静态的，资产领域，Twitter和Facebook可以成为并行更多资产;从twitter.com和si0.twimg.com资产可串联下载。这是一个非常简单的方法来获得更多的并发下载发生在网页上，甚至更好时，实际的CDN技术，可以从一个更适合的物理位置服务资产有助于降低延迟耦合的。

这是一切都很好，但后来我们将讨论如何从子域服务实际上，在某些情况下，会降低性能。

每次请求任何域中的资产的时候，出云相关的头一个HTTP请求，达到了资源，响应送回。这是一个巨大的过于简单的过程，但它是关于就像你真正需要知道的。这是一个HTTP请求，您引用的所有资产都受到这种往返。这些请求是主要的瓶颈，当涉及到前端的性能，因为，正如我们所覆盖，浏览器是由有多少这样的请求，可以并行发生的限制。这就是为什么我们经常要使用子域名;让这些请求发生在若干域，允许请求的更大数量的同时发生。

与这是一个问题，但是，是DNS查找。每次（从冷缓存）一个新的领域被引用，HTTP请求是受一个耗时的DNS查询（之间的任何地方20至120毫秒），其中传出请求查找那里的资产实际居住;互联网是由是由这是由DNS主机名管理引用IP地址捆绑在一起。

如果引用的每个新域都有一个DNS查询的前期费用，你必须确保它实际上将是值得的。如果你是一个小网站（如CSS巫术，例如），然后从子服务的资产将可能不会是值得的;浏览器或许可以从一个域​​快速获取几下，并行化的资产比它可以进行跨多个域的DNS查找和parallelise的。

如果你有可能一打的资产，你可能要考虑从一个子域名为他们服务;一个额外的DNS查找可能是值得的，以parallelise资产组合，其量更好。如果你有，比如说，40资产，它可能是值得分片跨两个子域的资产;两个额外的DNS查询将是值得为了从一共有三个领域的服务您的网站。

DNS查找是昂贵的，所以你需要确定哪个更适合您的网站;查找的开销或者只是从一个域提供的一切。

一定要记住，只要HTML是，比如说，从foo.com要求，对于主机的DNS查找已经发生了，对foo.com因此后续请求任何东西重要的是不受DNS查找。

如果你和我一样，希望有您的网站上一个Twitter的部件，和分析，也许有些网页字体，那么你将有链接到这意味着你必须承担DNS查找一些其他领域。我的建议始终不使用任何和每一个部件没有正确地先考虑它的性能的影响，但对任何你认为有必要的，下面是有用的...

因为这些东西都是在其他领域它意味着，例如，您的Web字体CSS将并行下载到自己的CSS，这在某种程度上是一个好处，但脚本仍然会阻塞（除非他们是异步）。

这里的问题，真的，是涉及与第三方域的DNS查找。幸运的是，一个超级快速简便的方法来加速这一进程：DNS预取。

DNS预取不正是是在锡说，并不能简单实现。如果需要，比如说，从widget.foo.com要求的资产，那么你可以通过简单地在你的页面中的<head>添加此早期预取该主机名的DNS：

<HEAD>

    ...

    <链接rel =“DNS预取的”href =“// widget.foo.com”>

    ...

</ HEAD>

这个简单的行会向浏览器支持的实际需要之前就开始预取的DNS域的一小部分。这意味着，DNS查询过程将已经由时间正在进行浏览器打<script>元素，实际上请求插件。它只是给浏览器中的小头开始。

这个简单的链接元素（这是我的CSS巫术使用）是完全向后兼容，并且不会对性能产生负面影响。认为它像性能逐步增强！

加快你的网站上使用预取

资源预取

以及DNS预取，它可以方便地预取你的网站需要以及任何资源。为了知道我们可能要预取，我们首先必须了解如何以及何时浏览器通常会请求的资源。

网页字体和图像在大致相同的方式的CSS的行为引用;浏览器将开始下载他们一旦它击中HTML的一部分，需要他们。就像我前面提到的，浏览器是很聪明的，这是另一个很好的例子。试想一下，如果浏览器下载尽快在CSS中引用的。

如果浏览器没有等待，直到它遇到需要这些图像，然后打的主页将下载他们四个的HTML。这是一种浪费，所以浏览器确保他们绝对需要的图像，他们开始下载之前。这里的问题是，下载的，因此，不会发生，直到漂亮晚。

如果我们可以完全肯定，我们总是希望在每一页上使用的CSS一定的形象，那么我们就可以欺骗浏览器进入初下载它，它遇到需要它的HTML之前。要做到这一点是令人难以置信的简单，但可以是一个有点脏，这取决于你如何去做。

肮脏的方式，并且很可能最防弹，是每个里面有CSS的图像与空的alt属性的<img>元素的网页有一个隐藏的<div>。我这样做的CSS巫术的精灵;因为我知道它需要使用在每一页上我可以自信地在我的HTML引用它预取它。该浏览器的方式处理内嵌的<img> s是，它预取他们的相当不错，并抓住他们的超级早，所以通过浏览器载入我的精灵作为一个<img>在我的标记，就可以开始CSS需要它之前下载它。我可以先在我的HTML引用它（隐藏）取得上下载获得领先地位。

周围有第二个，'清洁'的方式，它看起来很像DNS预取例如一些困惑：

<链接rel =“预取”HREF =“sprite.png”>

这明确地告诉我的浏览器开始预取我的精灵，不管评估我的CSS后可能作出任何决定。

困惑在于围绕两篇文章之间的差距看似;在此基础上，从MDN，似乎预取是一个提示的浏览器可能开始预取的资产，如果它是空闲的。相反，但是，这篇文章从地球的性能，这似乎表明，浏览器总是会预取的资产，如果它支持的rel =“预取”，而没有提到的空闲时间。我看着瀑布似乎表明后者是真实的，但一个奇怪的WebKit怪癖，让你可以在行动，如果你有开发工具开放（谈薛定谔的逆足...）没有观察到预取意味着我不能100％肯定。在这个任何澄清，将不胜感激。

我提到，字体和图像的工作非常大致相同的方式;上述规则适用完全相同的字体文件，但你不能在加载字体隐藏的<DIV>（你需要使用预取链接）。

<链接rel =“预取”HREF =“webfont.woff”>

因此，基本上，我们这里所做的是'欺骗'浏览器进入的时间提前下载的资产，所以，到时候它来应用我们的CSS，它已经下载的资源（或者至少在它的方式） 。漂亮！

加快你的网站上使用预取

CSS和性能

很多建议指出，如果你正在使用的资产领域，你应当成为从他们所有的静态资产;这包括CSS，JS，图像等

还记得早些时候，当我们讨论如何CSS渲染块？浏览器希望尽快，因为它可以让你的CSS保持，如果不是越早; CSS是你的关键路径上。你的关键路径是用户请求的网页，然后实际看到的东西之间​​的必要旅程。因为它可以阻止渲染，CSS是关键路径上，JS和图像都没有。你要保持沿尽可能快的关键路径这一旅程，这意味着没有任何DNS查找。

在工作​​中，我们正在建立一个网站，其临时环境服务的所有资产相同的主机（如foo.com），但是当它恍然使临时环境更多的活一样，我们开始从s1.foo服务于我们的所有资产.com和s2.foo.com。这意味着所有图像，JS，CSS，字体等都是来自不同的领域来了，因而导致DNS查询。这里的问题是，从冷高速缓存，以启动敛CSS文件实际上减慢关键路径右下所需的DNS查找。我们提供的图表大规模飙升，表明在理论上不应该发生的延迟;最好的做法决定了你应该分片大量资产超过子域，对不对？不是CSS。所需的DNS查找占用了时间，这延迟了页面渲染大量。