

本科生学士学位论文

视觉和惯性定位结合的目标追踪算法研究

Object Tracing Based On Combination of Vision and Sensor Localization

姓 名： 杜大有

学 号： 1300012870

院 系： 信息科学技术学院

专 业： 计算机科学与技术

指导老师： 梁云

二〇一七年五月

## 摘 要

运动目标检测及跟踪是机器视觉领域备受关注的前沿课题之一，同时需要保证机器人控制的实时性和目标定位的准确性。一方面，可以使用基于计算机视觉的方法，在摄像机采集的图像中搜索特定目标，提取目标区域，标定目标位置。另一方面，也可以使用惯性定位的方法，结合移动设备上的惯性测量单元，从而精确地定位目标的室内位置。

然而，基于计算机视觉的定位方法通常会由于短暂无法检测到目标（如目标被遮挡，或超出摄像机范围等）而失去目标的匹配，尤其是在多目标检测时；另一方面，单纯地使用移动设备上的传感器（加速度计、地磁计、陀螺仪等）进行惯性定位，会造成由两重积分带来的累计误差。因此，本文设计并实现了一种基于扩展卡尔曼滤波器的位移融合算法，旨在消除惯性定位的累计误差的同时，增强视觉定位的稳定性。实验表明，融合后的轨迹能与目标的实际轨迹较好地吻合；在摄像头因为某种原因暂时无法检测到目标时，惯性定位能够持续反馈出较为准确的位置，从而重新匹配目标。

本文的主要工作如下：

* 分析了当前基于计算机视觉
* 设计并实现了一种视觉和惯性定位结合的传感器融合系统。该系统
* 通过，验证了本文的传感器融合系统的有效性。

**关键词：传感器融合，运动目标追踪，扩展卡尔曼滤波，惯性测量单元**

## Abstract

On the mobile devices like smartphones and tablet computers, Web browser is one of the main entrance of the Internet. However, the existing mobile browsers have not considered user experience very well. The page load speed is slow. The data usage is wasteful. And the energy consumption is too much.

The cache mechanism is an effective way to solve the problems above, but the existing browser’s cache performance is not satisfying. It has a huge space to improve. Therefore, without the change of user browsing habits and the configuration of Web applications, this thesis designs and implements a cache optimization system. With checksum comparison, data compression and website prefetching methods. This system can reduce the data usage and increase the load speed. The experiment shows that this system can save data usage by 17% and load time by 29% on average when users load a Web page at the first time. When users load the same page immediately, this system can save data usage by 66% and load time by 9% for this second load.

The contributions of this thesis are as follows.

* This thesis introduces existing browser’s cache mechanism and potential defects, analyses of the upper bound of cache performance and the cause of the gap between actual and ideal performance, raises the optimization scheme.
* With the optimization scheme, this thesis designs and implements the proxy-based cache optimization system for mobile Web browser. This system consists of a system service on mobile devices and a personal proxy server, which records browser’s requests, selectively pushes resources and optimizes the transition.
* This thesis measures the Web page load time and data usage with and without optimization, and proves that the proposed system can both speed up the page load and reduce the data usage for mobile Web browsing.

**Key Words: Sensor Fusion, object tracking, Extended Kalman Filter(EKF), Inertial Measurement Unit(IMU)**

## 目 录

[摘 要 2](#_Toc482652280)

[Abstract 3](#_Toc482652281)

[目 录 4](#_Toc482652282)

[第一章 引言 1](#_Toc482652283)

[1.1 运动目标检测及跟踪的应用现状 1](#_Toc482652284)

[1.2 现有目标追踪算法的分析与局限 2](#_Toc482652285)

[1.2.1 基于计算机视觉的目标检测与追踪 2](#_Toc482652286)

[1.2.2 基于IMU的惯性定位 3](#_Toc482652287)

[1.3 本文的主要工作与结构 4](#_Toc482652288)

[1.3.1 本文的主要工作 4](#_Toc482652289)

[1.3.2 本文的结构 5](#_Toc482652290)

[第二章 理论推导与算法框架 6](#_Toc482652291)

[3.1 基于9轴传感器的手机姿态测算 6](#_Toc482652292)

[3.1.1 概述 6](#_Toc482652293)

[3.1.2 手机传感器坐标定义 6](#_Toc482652294)

[3.1.3 9轴传感器姿态融合算法 8](#_Toc482652295)

[3.1.4 姿态初始化算法 9](#_Toc482652296)

[3.1.5 融合加速度计和地磁计的姿态测算原理 11](#_Toc482652297)

[3.2 惯性定位与视觉定位结合的位移测算 13](#_Toc482652298)

[3.2.1 概述 13](#_Toc482652299)

[3.2.2 视觉定位系统简述 14](#_Toc482652300)

[3.2.3 惯性定位与视觉定位融合滤波原理 15](#_Toc482652301)

[3.2.4 惯性定位与视觉定位融合算法 16](#_Toc482652302)

[第三章 算法的具体实现 17](#_Toc482652303)

[4.1 获取手机传感器数据 17](#_Toc482652304)

[4.2 手机姿态解算 19](#_Toc482652305)

[4.2.1 姿态初始化 19](#_Toc482652306)

[4.2.2 EKF预测阶段 20](#_Toc482652307)

[4.2.3 EKF更新阶段 21](#_Toc482652308)

[4.3 位移融合滤波 22](#_Toc482652309)

[4.3.1 未收到视觉信号时的惯性定位 22](#_Toc482652310)

[4.3.2 收到视觉信号后融合滤波 22](#_Toc482652311)

[第四章 实验与效果分析 27](#_Toc482652312)

[5.1 姿态测算实验 27](#_Toc482652313)

[5.1.1 实验方案 27](#_Toc482652314)

[5.1.2 实验结果 27](#_Toc482652315)

[5.1.3 实验结果讨论 27](#_Toc482652316)

[5.2 位移融合滤波实验 27](#_Toc482652317)

[5.2.1 实验方案 28](#_Toc482652318)

[5.2.2 实验结果 28](#_Toc482652319)

[5.2.3 实验结果讨论 29](#_Toc482652320)

[5.3 特殊事件下的系统稳定性实验 30](#_Toc482652321)

[5.3.1 实验方案 30](#_Toc482652322)

[5.3.2 实验结果 32](#_Toc482652323)

[5.4 小结 33](#_Toc482652324)

[第五章 工作总结与展望 36](#_Toc482652325)

[6.1 本文工作总结 36](#_Toc482652326)

[6.2 未来工作展望 36](#_Toc482652327)

[参考文献 38](#_Toc482652328)

[致 谢 39](#_Toc482652329)

## 第一章 引言

1.1 运动目标检测及跟踪的应用现状

根据中国互联网络信息中心于2015年1月发布的《第35次中国互联网络发展状况统计报告》，截至2014年12月，我国手机网民规模达5.57亿，较2013年增加5672万人。网民中使用手机上网的人群占比由2013年的81.0%提升至85.8%。连续几年来，手机网民规模保持稳定增长，手机也早已成为第一大上网终端。

越来越多的人们正在利用手机或者平板电脑来通过Wi-Fi或3G网络上网，随之而来的，Web浏览器成为了移动设备上最常用的应用之一。根据德克萨斯大学2014年的研究，浏览器占据了移动设备63%的显示时间。但是，现有的这些浏览器带来的用户体验却远不尽如人意。资源加载是影响浏览器性能重要因素之一，有实验表明，网页加载65%的时间都花在了资源加载上。由于移动设备的网络状况通常并不理想，尤其是可能出现网络拥挤或是用户正在移动的状况，这个问题就变得更为严重了。此外，下载资源的过程本身会带来非常严重的电量消耗，而电量恰恰是移动设备上最为稀缺的资源。

为了加速Web资源加载，缓存机制被广泛运用。通过将已访问网站的资源存储到本地（内存或磁盘），当用户再次访问这些网站时，就可以尝试从本地取这些资源，而不用再从远程Web服务器上请求了。通过上述缓存的方法，可以为用户减少资源下载量，从而提高Web浏览性能。这种缓存机制给移动设备带来的好处更大，因为在移动设备上，该机制不单可以减少网页加载时间，还可以减少网络流量和电量消耗。

然而，虽然缓存机制已经被详细定义在了HTTP协议中，在实际运用中，缓存却并未被很好地配置与实现。此前的研究表明，缓存机制的理论优化空间很大，而测得的实际性能却并不如人意。因此当前有这样一个需求，在不改变现有浏览器与服务器实现的前提下，对现有的缓存机制进行优化，提高缓存的利用效果。希望可以以此进一步提高移动设备用户在浏览网页时的体验，减少用户的流量与电量消耗。

1.2 现有目标追踪算法的分析与局限

1.2.1 基于计算机视觉的目标检测与追踪

本文只分析HTTP协议中规定的缓存以及浏览器对其的实现。

浏览器为了加载一个网页，需要向服务器请求每一个资源，而请求每一个资源需要向指定服务器发出请求资源的HTTP头，服务器会根据这一请求返回相应的HTTP响应头以及资源体，浏览器接收到资源并进行渲染，同时可能将资源进行缓存。浏览器是否该缓存此资源，以及何时可以调用缓存的信息就包含在HTTP响应头中。下面来用一个HTTP响应头的例子来说明服务器是如何通过HTTP响应头配置缓存的。

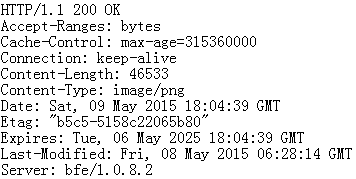
  
图1：2015年5月10日请求百度首页Logo资源的响应头

图1是响应头的示例，此处关注响应头中的以下部分：

**响应状态行：**样例中的第一行。当浏览器有过期的缓存时，可以向服务器验证缓存的有效性，此处为304 Not Modified时表示验证通过，缓存有效，不返回资源本体，浏览器使用缓存的资源渲染网页；此处为200 OK表示验证不通过，服务器会返回完整的资源。

**Cache-Control域：**这里标示了缓存策略，如是否应该缓存。可以用max-age标示了缓存的有效时间。

**Cache-Control的max-age设置与Expires域：**均标示缓存的有效期限，max-age含义为多少秒后过期，Expires标明过期的时间点。（当响应头的Cache-Control中设定了max-age时，优先级高于Expires）浏览器不能直接使用过期的缓存渲染页面。

**Etag域：**此处内容为该资源在服务器上的唯一标示符，通常为文件信息的hash值，不同的资源有不同的Etag信息，在后面提到的缓存验证中有作用。

**Last-Modified域：**此处标示该资源在服务器上最近一次更新的时间，在后面提到的缓存验证中有作用。

需要注意的是，这些缓存相关的信息并不是网站必须提供的，好的网站设计者会通过配置这些参数来提高用户加载网站的速度，当没有这些配置信息时，浏览器可以通过自身的启发式算法来配置缓存的过期时限。上面提到的是服务器对于资源的缓存设置，浏览器在接收到这些信息之后会记录并采用这些设置。浏览器判断是否存在缓存的唯一依据是资源的URL，在第二次访问请求此URL时，浏览器是使用缓存的方法是这样的，如图2所示。

  
图2：第二次请求同一URL，浏览器使用缓存的流程图

1.2.2 基于IMU的惯性定位

根据上一节的分析，可以发现，有一些情况下，缓存不能被很好地利用：

* **URL变化的相同资源：**由于URL是缓存利用的唯一索引，在某些情况下，当同一资源在服务器上的URL发生改变时，可能此前浏览器曾经缓存过资源本体，当前的缓存机制还不能解决由此带来的额外流量开销。
* **不必要的资源传输：**服务器端会有一些资源没有设置Last-Modified或Etag，在这种情况下，如果缓存过期，浏览器将不得不从服务器请求一份新资源，即使内容与原缓存一致。当缓存有效期较短，用户访问较频繁时，会给用户带来大量不必要的流量开销。
* **错误缓存问题：**当缓存未过期时，浏览器从本地缓存中读取资源并渲染网页，然而，有一些资源已经在服务器上更新了，浏览器却错误地使用了过期的资源，而没有从服务器上重新请求这些新资源，本文称这种情况为错误缓存。错误缓存可能会极大地影响用户的浏览体验，尤其是当如CSS或JavaScript这种功能型资源发生错误缓存时，相关的网站可能会产生完全错误的显示。
* **缓存时限设置不当：**网站的开发者设置一些资源的有效期可能是不准确的，还有些网站的开发者不设置有效期，此时浏览器会通过自身算法“假设”资源的有效期。这种并不可靠的有效期可能会带来冗余传输、错误缓存和不必要的资源验证。

以上是根据缓存机制可以分析出的现有缓存的潜在缺陷。下一章中将分析每一种缺陷的成因和对缓存性能的影响程度，以及改良这些缺陷后缓存性能的提升量。

1.3 本文的主要工作与结构

1.3.1 本文的主要工作

本文的主要工作包括：

* 介绍了当前浏览器的缓存机制与潜在的缺陷，分析了缓存的理论性能上限与当前性能间的差距及造成的原因，验证了实现浏览器的缓存优化是有意义而且可行的，并提出了优化方案。
* 针对提出的优化方案，本文设计并实现了基于代理的移动浏览器缓存优化系统，该系统有三个子系统组成：手机端后台服务、服务器端在线服务、服务器端离线资源抓取程序。该系统通过移动设备端的后台服务和云端服务器对浏览器请求资源的记录、选择性推送与优化传输，实现了对现有缓存机制的优化。
* 设计实验验证了本文的缓存优化系统对提升网页加载速度和节省流量的有效性。实验表明，该系统在用户第一次访问网页时，平均可以节省流量17%，提高加载速度29%；立刻再次访问同一网页，平均可以为第二次访问节省流量66%，提高加载速度9%。

1.3.2 本文的结构

本文的结构如下：

第一章呈现了移动浏览器的应用现状，对现有移动浏览器的缓存机制进行了分析，并指出了现有缓存的潜在缺陷。

第二章分析了缓存的理论上限与当前性能间的差距及造成的原因，验证了实现浏览器的缓存优化是有意义而且可行的，并对可行的缓存优化手段进行了分析，提出了优化方案。

第三章基于面向对象的方法对本文的系统进行了分析与设计。

第四章给出了移动浏览器缓存优化系统的具体实现细节。

第五章对本系统进行了实验设计，并描述了自动化测试系统的设计与实现，展现了实验的结果，最后分析了本系统带来的额外开销。

第六章对全文的工作进行了回顾和总结，并提出了未来工作的三个方向。

## 第二章 理论推导与算法框架

本章对传感器和视觉结合的定位系统的各个部分进行了详细说明和理论推导，同时给出了相应的算法流程以在整体上说明各个部分的工作方式。

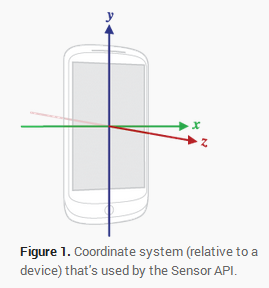
3.1 基于9轴传感器的手机姿态测算

3.1.1 概述

总体而言，本文参照[\*\*\*]中的理论推导，使用四元数作为姿态的状态表示，以避免使用旋转矩阵表示造成的冗余或欧拉角表示产生的奇异点；使用扩展卡尔曼滤波器将角速度传感器数据和外源观测数据融合，消除角速度计的漂移。与[\*\*\*]中不同的是，本文中用以融合的数据为加速度计和地磁传感器数据，而不是恒星计的数据。比较而言，由于基于加速度计和地磁传感器的姿态测算原理利用静止状态下的地球重力场和地磁场，因此在手机运动时产生的误差更大，得到较为精确姿态的难度更高。

下文中将略去共性较大的部分，即姿态的四元数表示、四元数的相关运算、陀螺仪的误差模型、卡尔曼滤波器的状态方程和预测方程等，着重介绍与其不同的部分。

3.1.2 手机传感器坐标定义



图\*\*\*手机传感器坐标轴图示

本文中一共读取了三种手机传感器数据：平动加速度计，角速度计和地磁传感器。这三类传感器数据均有三个维度，其对应定义如下：

平动加速度计：如图\*\*\*所示，其x坐标平行于手机短边，面向手机朝向右侧为正；其y坐标平行于手机长边，面向手机朝向上方为正；其z坐标穿过手机屏幕向外为正。

地磁传感器：和平动加速度计坐标定义相同。如图\*\*\*所示，其x坐标平行于手机短边，面向手机朝向右侧为正；其y坐标平行于手机长边，面向手机朝向上方为正；其z坐标穿过手机屏幕向外为正。

角速度计：绕平动加速度计x轴旋转为第一坐标（Pitch），顺时针旋转为正；绕平动加速度计y轴旋转为第二坐标（Roll），顺时针旋转为正；绕平动加速度计z轴旋转为第三坐标（Azimuth），顺时针旋转为正。

3.1.3 9轴传感器姿态融合算法

否

姿态初始化

读入传感器数据

使用角速度计数据更新先验姿态

计算线性加速度模长和磁感应强度模长

加速度模长小于阈值？

地磁计模长小于阈值？

使用加速计增量更新姿态

使用地磁计增量更新姿态

否

是

是

否

一次姿态计算循环

图\*\*\* 姿态融合算法流程图

如图\*\*\*所示，姿态融合算法将在开始时进行初始化，并在之后的时间内不断进行姿态计算循环。

姿态初始化：

在初始化期间采集一定时间的数据（约2s左右），并认为此时间内手机静止，主要目的是使用加速度计和地磁计计算初始姿态，并且消除角速度计的初始漂移。这一部分将在\*\*\*进行详细介绍。

姿态计算循环：

首先，在读取传感器数据后，对角速度进行积分得到手机先验姿态，并计算其相应的协方差矩阵，以同其余传感器进行扩展卡尔曼滤波消除陀螺仪漂移。

然后，计算加速度模长和磁感应强度模长。其理由是加速度和地磁计姿态解算的原理为认为手机近似静止，利用地球的重力场和地磁场来构建三维笛卡尔坐标系从而解算姿态（详见\*\*\*），而如果手机不在匀速运动（即带有加速度）或者外部有磁场干扰的条件下，姿态计算误差就会变大（见实验\*\*\*）。因此，本文中为加速度模长和磁感应强度模长分别设定了一个阈值。当真实读数小于对应的阈值时，认为其相对于地球重力场或磁力场的影响很小，可以使用相应读数来更新姿态；否则，本次循环中不使用该组数据来更新姿态。

每完成一个姿态计算循环后，等待下一组传感器信号输入，进行下一次循环。该扩展卡尔曼滤波器的计算量较小，其计算时间远小于传感器信号采样周期（约1~2ms），完全可以达到实时性的要求。

3.1.4 姿态初始化算法

读入传感器数据

分别累计角速度向量、加速度向量、地磁计向量

达到设定的初始化时长？

记录初始linux时间戳

对角速度计累计值取平均，存储作为陀螺仪偏移

对加速度累计值和地磁计累计值取平均

根据加速度向量和地磁计向量平均值计算初始姿态

是

否

初始数据采集阶段

初始姿态计算阶段

图\*\*\* 姿态初始化算法流程图

如图\*\*\*所示，姿态初始化算法分为初始数据采集阶段和初始姿态计算阶段。

在初始数据采集阶段，程序分别累计设定时间内（2s）的角速度向量，加速度向量和地磁计向量。以试验中约700HZ计算，可以累计约1400组数据，并记录起始linux时间戳和结束linux时间戳。在此期间，假设手机完全静止，其姿态不发生变化。

在初始姿态计算阶段，首先，对角速度计累计值取平均，存储作为陀螺仪偏移。在之后使用任何陀螺仪读数前，均减去此偏移。然后对加速度累计值和地磁计累计值取平均，并根据加速度向量和地磁计向量平均值计算初始姿态（公式推导见\*\*\*），并初始化相应的协方差矩阵。在之后的姿态预测和更新过程中，此姿态（转换为四元数表示）作为初始值进入扩展卡尔曼滤波算法，使用角速度数据进行预测，使用加速度计和地磁计读数进行更新。

3.1.5 融合加速度计和地磁计的姿态测算原理

加速度计读数和地磁计读数有两部分作用：建立初始姿态和在卡尔曼滤波器中更新后验姿态。

1. 建立初始姿态的公式推导：

输入：初始数据采集阶段的加速计读数均值，地磁计读数均值，它们均为手机自然坐标系中的**三维列向量**，方向和正负见\*\*\*中定义。

输出：初始旋转矩阵

1. 对和进行归一化，即
2. 计算旋转矩阵

1. 验证旋转矩阵

令地球重力场单位向量为（地面系下）

令地磁场单位向量为（地面系下）

由旋转矩阵定义，将地球重力场和地磁场向量转换至手机自然坐标系有

即理论上等于手机测量值

1. 使用初始旋转矩阵初始化卡尔曼滤波器的公式推导：

输入：初始旋转矩阵，陀螺仪（角速度计）采样周期

参数：陀螺仪方差（variance of the gyro's output per Hz）陀螺仪偏误方差

输出：扩展卡尔曼滤波器初始状态量（姿态四元数***q***）和噪声协方差矩阵

1. 将旋转矩阵表示转换为四元数表示
2. 计算矩阵Q

1. 计算噪声协方差矩阵

令

则噪声协方差矩阵为

1. 使用加速度计或者地磁计更新后验姿态公式推导：

输入：某一时间点加速计读数，地磁计读数，先验姿态（四元数表示）和协方差矩阵

输出：更新后的后验姿态和协方差矩阵

1. 将

3.2 惯性定位与视觉定位结合的位移测算

3.2.1 概述

面向对象的设计是对分析的细化，同时需要结合具体实现的环境对分析的结论进行调整，本系统实现的环境如下：

* **通信协议：**为了提高系统效率，本文需要实现服务器推送（server push），TCP连接复用等功能，因此本文的手机端与服务器端通信采用SPDY协议。
* **手机端：**浏览器使用Android平台可以配置HTTP代理的Firefox浏览器。手机端需要在后台启动服务，为浏览器提供HTTP代理。因此手机端需要实现Android上的App，该应用可以作为后台服务运行。编程语言为Java。
* **服务器在线端：**服务器端需要处理手机端的连接请求，同时通信使用SPDY协议。因此本文将Jetty的SPDY模块内嵌到服务器端的程序中，并由此搭建服务器，编程语言为Java。
* **服务器离线端：**该程序需要能抓取网站的所有资源，此前的研究表明，为了获得网站资源列表，只能使用真实浏览器渲染。本文将JCEF实现的浏览器内嵌到该离线端，用来获得网站资源列表。程序的编程语言为Java。
* **数据库：**为了实现在服务器端实现离线抓取网站、对缓存长久保存、同步手机缓存资源等功能，需要在服务器端搭建数据库，本文使用MySQL数据库。

在第四章中，将对所涉及的具体工具的进行更为详尽的介绍。

3.2.2 视觉定位系统简述

  
图11：系统架构图

本文的系统由三个子系统组成：

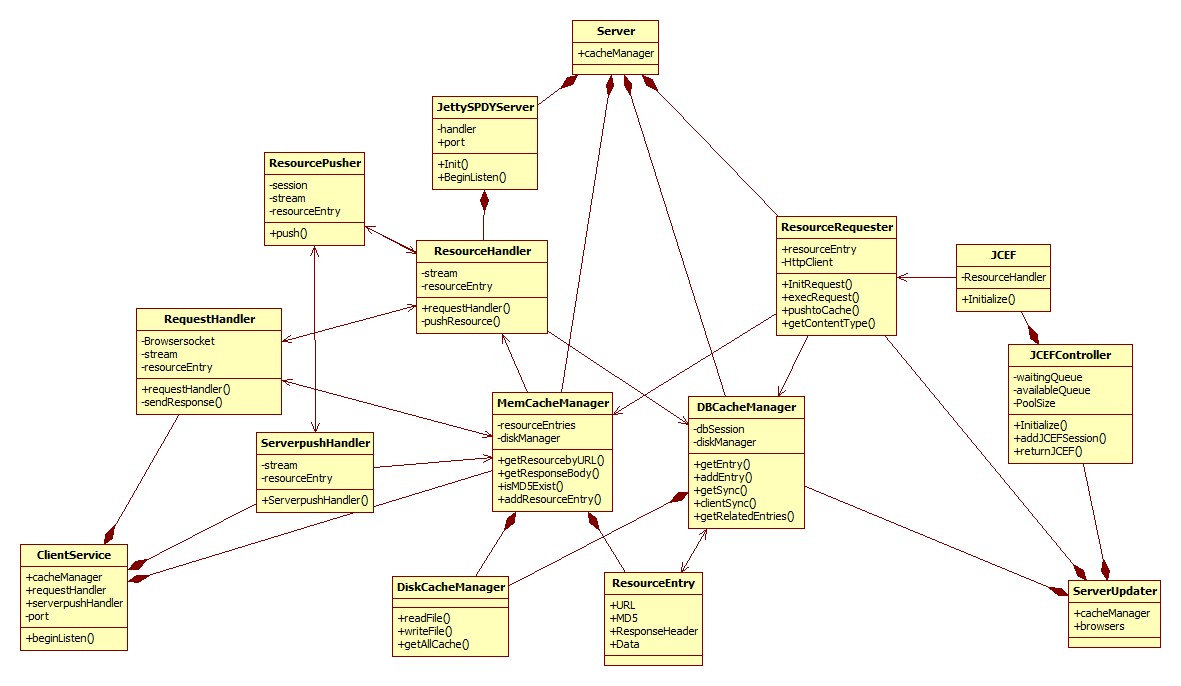
**手机端后台服务：**通过HTTP协议接收Firefox浏览器的请求并发送响应，通过SPDY协议与服务器在线系统进行通信，发送资源请求，并接收服务器推送的资源。系统内部由3个子模块组成，JettySPDYClient是与服务器通信的模块，同时处理服务器推送的数据；MemCacheManager为缓存控制模块，用来处理缓存条目的查询与添改，同时控制磁盘上的缓存存储；RequestHandler用于处理浏览器的请求，通过缓存管理模块检索缓存并通过通信模块向服务器发送资源请求与接受响应。

**服务器端在线程序：**通过SPDY协议与手机端通信，处理手机端发来的资源请求，推送手机端需要的资源。JettySPDYServer是与手机端的通信模块，同时负责开启端口让手机端接入；ResourceHandler用来实际处理手机端的请求，访问数据库检索资源，向手机端返回资源，该模块内含两个子模块，ResourcePusher用于向服务器推送资源体，ResourceRequester基于HTTPClient完成Web资源获取的工作。

**服务器端离线程序：**读取用户可能访问的资源列表，并从网上抓取访问网站相关的所有资源。共由3个子模块组成，ResourceRequester获取Web资源，CEFPool控制浏览器加载网站并驱动ResourceRequester请求资源，Prefetcher负责数据库的读取与离线程序的控制。

3.2.3 惯性定位与视觉定位融合滤波原理

根据系统环境与架构的设计，本文在面向对象分析类图的基础之上，对类图进行了修改与细化。

  
图12：面向对象设计类图

相比面向对象分析中的类图，主要有以下三个改动：

* 细化缓存管理类，将原CacheManager拆分为3个类，分别负责内存、磁盘、数据库中的缓存。手机端程序不涉及数据库，因此没有数据库管理类；离线端程序不需要在内存记录缓存，因此没有内存缓存管理。
* 在离线资源抓取系统中，针对JCEF设计控制类，重写JCEF的资源获取方法，从而可以用ResourceRequester取资源并记录数据
* 细化手机端与服务器端的通信事件处理类，手机端细化为手机资源请求类和服务器推送处理类，服务器端细化为服务器推送类和手机资源请求处理类。

3.2.4 惯性定位与视觉定位融合算法

本系统共建立了两个数据库表：resource与userpreference。

resource表用于存储服务器的缓存条目与手机端的同步情况，该表共含5个资源，细节如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 类型 | 设置 | 说明 |
| URL | varchar | 主键 | 资源的URL |
| MD5 | varchar | 非空 | 该资源体的MD5，长为32的字符串，资源的实际内容存储在磁盘中，名称为MD5的值 |
| Header | varchar | 非空 | 该资源最新的响应头 |
| Father\_URL | varchar |  | 表示该资源在哪一个URL的资源列表中 |
| Sync | integer | 非空 | 表示该资源体是否在手机上有储存，0表示未储存，1表示已储存 |

userpreference表用来存储用户可能访问的网站，即服务器离线取资源的预取列表，只有一个字段用来保存用户访问网站的URL。

## 第三章 算法的具体实现

本章的主要内容是缓存优化系统的实现部分，首先介绍实现过程中涉及的相关工具与协议，此后介绍系统的各个子系统与各个模块的具体实现与伪代码。

4.1 获取手机传感器数据

SPDY是由Google创造的Web内容传输协议，SPDY协议是基于TCP协议的，在应用层上对原HTTP协议进行的优化，目标是降低网络延迟，提高安全性。下一代通信协议HTTP/2正是在SPDY的基础之上修改的。在本文的应用中，由于HTTP/2协议相对较新，目前支持HTTP/2协议的开源项目非常少，而且还远不成熟，而SPDY已经具有3年的历史，相关Web应用与开发资料比较丰富。因此本文使用SPDY作为手机端与服务器端的通信协议。

SPDY在安全套接层（SSL）之上创建了会话层（Session Layer），本文主要利用SPDY会话层的功能来实现客户端对资源的请求与服务器端对资源的推送。这个会话层是允许连接的多路并发的，可以在单个TCP连接上通过流（Stream）的方式进行同时双向同时传输，而且客户端与服务器端均可以在会话层上创建多个流。当会话创建后，可以用每一个流来模拟HTTP短连接，从而解决TCP连接建立慢的问题。此外，在同一会话中，不同的流可以设置不同的优先级，这样可以将不同的资源以不同的优先级进行传输。

在流上，数据以帧（Frame）的方式进行传输，服务器与客户端均可以发送帧，帧可以用来传送控制信息并可以附带数据。帧分为控制帧（Control Frame）和数据帧（Data Frame）两种，常用的控制帧有以下两种：

* SYN\_STREAM帧：用来主动创建流，后文简称SYN帧。本文的系统中，手机端发送此帧，表示手机向服务器请求资源；服务器端发送此帧，表示进行一个资源的Server push。
* SYN\_REPLY 帧：在接收到SYN\_STREAM帧后，发回此帧表示连接成功建立，后文简称REPLY帧。本文的系统中，手机端发送此帧，表示手机端已准备好接收server push；服务器端发送此帧，会附带资源信息，表明这个资源手机是否已有缓存。

数据帧没有控制流的作用，其功能是在流建立完成后，在流上互相发送数据，后文称DATA帧。本文的系统中，只有服务器端可以发送此帧，用来传输Server push的资源内容。

以上SPDY协议在本系统中的具体应用方式详见后文伪代码。

Jetty项目为开发者提供了一个易用的Web服务器与Servlet容器，同时还有大量的网络功能，如SPDY、WebSocket等。Jetty的所有构件都是开源的并可以用作商业用途。Jetty是由Java语言编写的，其API以多个Jar包的形式进行发布。Jetty的特点是可以把Web服务器实例化成一个对象，从而可以很容易地嵌入Java应用，来提供Web服务器的功能。

本文在服务器端使用Jetty搭建了基于SPDY协议的服务器，在手机端使用Jetty的SPDYClient库与服务器端进行SPDY协议的通信。使用Jetty开发对JDK的最低版本要求是1.7，嵌入Android时，对Android系统的最低版本要求是4.4 (API Level 19)。这也是本文系统运行的最低Java版本与Android版本。

实验证明，静态分析HTML的方式不能分析出加载一个网站所需要的全部资源，如果想要得到加载一个网站对应的资源列表，最好的方式是使用浏览器引擎将其加载一遍。CEF是基于Google Chromium项目的开源浏览器项目，但CEF与Chromium并不相同，Chromium关注的是开发Google Chrome应用，着眼于优化渲染网页的方式；CEF关注浏览器在第三方应用上的嵌入式应用，使开发者可以在不了解浏览器内核的情况下，通过CEF提供的API来开发浏览器应用。大量的软件使用CEF为应用提供内嵌浏览器，如Evernote，QQ等

CEF是C++语言的项目，基于CEF项目，开发小组启动了JCEF项目，即CEF的Java语言项目。本文系统实现的语言为Java，因此本文使用JCEF作为内嵌的浏览器。

本文使用JCEF来在应用中嵌入浏览器并通过接口创建它们。在离线采集系统中，当系统可以预先判断出用户喜好的资源列表时，就可以在服务器上通过让JCEF浏览器通过渲染网页的方式得到加载到用户喜好网站的资源列表，从而完成预取的功能。

4.2 手机姿态解算

4.2.1 姿态初始化

RequestHandler类主要负责在手机端处理浏览器发出的资源加载请求，包括RequestHandler1和RequestHandler2两个方法，分别为接收浏览器请求的处理函数和接收服务器响应的处理函数，下图给出其伪代码。

|  |
| --- |
| // 浏览器通过代理服务器加载资源的事件处理函数  // 客户端新建一个线程执行该函数  function RequestHandler1(BrowserSocket)  // 读取浏览器请求  browserRequest <- getRequestFromSocket(BrowserSocket)  // 对于不支持的协议或方法返回404  if browserRequest not supported  BrowserSocket.return404()  return  // 从请求中提取目标URL与请求头  URL <- parseURL(browserRequest)  // 从CacheManager中尝试检索资源  resourceEntry <- MemCacheManager.getEntry(URL)    // 资源条目如果在缓存中存在  if resourceEntry exists  // 如果资源正在传输过程中，线程挂起  if resourceEntry is transmitting  // 等待在该资源上  resourceEntry.wait()  // 资源准备好后通过按照HTTP响应结构发给浏览器  BrowserSocket.sendResponse(resourceEntry)  else  // 找到当前可用的SPDY会话，如果过期则重新创建  session <- getSPDYSession()  // 创建发送给服务器的请求头  header <- newSPDYHeader()  header.add("URL", URL)  header.add("header", browserRequest)  // 通过session发送SYN帧  // 创建stream并将访问相关信息发服务器，保持流畅通  // 注册RequestHandler2函数作为服务器返回信息的事件处理函数  session.syn(header, STREAM\_NOT\_CLOSE,  RequestHandler2(URL, BrowserSocket))  // 服务器发回响应时的处理函数，服务器返回资源的相关信息  function RequestHandler2(URL, BrowserSocket, ReplyInfo)  // 从服务器的响应中提取响应类型、响应头、MD5  replyType <- getReplyType(ReplyInfo)  responseHeader <- getResponseHeader(ReplyInfo)  MD5 <- getResponseMD5(ReplyInfo)    // 判断响应的内容是否已存在  if replyType is "MD5Exist"  // 向浏览器发送响应  BrowserSocket.sendResponse(responseHeader,  MemCacheManager.getData(MD5))  else  // 等待服务器推送该URL的资源  MemCacheManager.WaitOnURL(URL)  // 等待后线程被重启，向浏览器发送资源  resourceEntry <- MemCacheManager.getEntry(URL)  BrowserSocket.sendResponse(resourceEntry) |

图13：RequestHandler的伪代码

4.2.2 EKF预测阶段

ServerpushHandler类主要负责处理服务器端主动通过SPDY流发起的资源推送，仅包括ServerpushHandler方法，为接收到服务器端主动发起SPDY流后的处理函数，下图给出其伪代码。

|  |
| --- |
| // 用于监听服务器通过SYN帧发起的stream的处理函数  function ServerpushHandler(stream, synInfo)  // 从传来的头文件中得到推送资源的：  // URL, MD5, 在本地是否存在，响应头，数据大小，压缩后大小  URL <- synInfo.getURL()  MD5 <- synInfo.getMD5()  MD5Exist <- synInfo.getMD5Exist()  responseHeader <- synInfo.getResponseHeader()  dataSize <- synInfo.getDataSize()  zipSize <- synInfo.getZipSize()  // 如果该资源已存在，则保留响应头，无需传递资源体  if MD5Exist  MemCacheManager.addNewEntryFromExist(URL,  MD5, responseHeader)  return  // 向服务器使用REPLY帧发回响应，服务器接下来传数据  stream.reply(STREAM\_NOT\_CLOSE)  // 即使资源尚未传输完成，也先在缓存中增加该URL项  // 浏览器请求该资源时，会等待该资源准备完成  resourceEntry <- MemCacheManager.addNewEntry(URL,  MD5, responseHeader, DataSize)  // 监听DATA帧，服务器会通过DATA帧传回资源  stream.addDataListener(DataListener(resourceEntry))  // 用于监听服务器通过DATA帧发送资源的处理函数  // DATA帧的内容已在服务器经过GZip静态压缩  function DataListener(resourceEntry, stream)  //读取服务器通过DATA帧推送的所有资源，并放在zipArray中  while stream is not close  zipArray.append(stream.data)  //对数据解压缩，并写回到MemCacheManager中  resourceEntry.data <- GZipUtils.decompress(zipArray)  //向MemCacheManager发送消息，表明资源准备好  //回执行数据往磁盘的写入并继续所有挂起在该URL上的线程  resourceEntry.dataReady() |

图14：ServerpushHandler的伪代码

4.2.3 EKF更新阶段

MemCacheManager主要负责管理内存中的缓存，除此之外，同时也通过DiskCacheManager来管理磁盘中对资源的缓存，该类实现对内存中缓存条目获取与添加修改。需要实现的方法包括：添加资源条目、根据URL获得资源条目、根据MD5查询资源的存在情况、根据MD5获得资源体。具体的实现伪代码在此不再赘述。

4.3 位移融合滤波

4.3.1 未收到视觉信号时的惯性定位

该对象的作用是打开端口监听SPDY的会话层连接，默认端口为8181。手机端通过SPDY的会话层连接连上此端口后，当手机端发送SYN帧建立流时，触发事件并调用ResourceHandler处理函数。

4.3.2 收到视觉信号后融合滤波

ResourceHandler类的主要负责在手机发送SYN帧建立流时处理手机的资源请求，其中实现了ResourceHandler方法为接收SYN帧时的处理函数，下图给出其伪代码。

|  |
| --- |
| // 当手机端执行RequestHandler1时，发来SYN帧时触发事件执行该处理函数  // 此函数返回响应时出发手机端的RequestHandler2函数  function ResourceHandler(stream, synInfo)  // 初始化传回手机端的响应  replyHeader <- newSPDYHeader()  // 读取手机端传来的资源请求，包括URL与请求头  URL <- synInfo.getURL()  requestHeader <- synInfo.getResquestHeader()    // 从内存缓存中根据URL读取资源  // 注：服务器端的MemCacheManager的内容与手机是同步的  resourceEntry <- MemCacheManager.getEntry(URL)  // 如果存在则表明手机端请求的资源在传送中  if resourceEntry exists  // 此时应让手机端等待  replyHeader.add("ReplyType", "Waiting")  // 发回回复，并将流关闭  stream.reply(replyHeader, STREAM\_CLOSE)  else  // 从数据库尝试读取URL，查看服务器端已缓存本资源  resourceEntry <- DBCacheManager.getEntry(URL)  // 数据库不存在此URL，从网络在线取资源  if resourceEntry is null  resourceRequest <- newRequest(URL, requestHeader)  resourceEntry <- resourceRequest.execute()  // 查询手机端是否已缓存该资源体  isSync <- DBCacheManager.getSync(resourceEntry.MD5)  // 同步时，让手机端调用本地资源体，否则需要发送资源  if isSync  replyHeader.add("ReplyType", "MD5Exist")  else  replyHeader.add("ReplyType", "New")  ResourcePusher.push(stream, resourceEntry)  // 附上相关响应头与MD5，向手机端返回  replyHeader.add("ResponseHeader",  resourceEntry.header)  replyHeader.add("MD5", resourceEntry.MD5)  stream.reply(replyHeader, STREAM\_CLOSE)    // URL可能是数据库已缓存的网站HTML（即Father\_URL）  // 此时打包返回所有该网站相关的未同步的资源  relatedEntries <- DBCacheManager.  getRelatedEntries(URL)  for each relatedEntry in relatedEntires  ResourcePusher.push(stream, relatedEntry) |

图15：ResourceHandler的伪代码

ResourceRequester类的实例负责处理每一个服务器对Web资源的请求。该类基于HTTPClient 3.1对网络资源进行请求，该类可以读取浏览器的请求头并将请求头整合到HTTPClient的请求里面。在HTTPClient默认配置之上，本系统做了以下修改：

* 关闭资源重定向，原样返回状态为302的响应
* 为了提高浏览器的加载速度，将资源连接超时设定为2秒，并关闭资源请求重试的功能

ResourcePusher类负责服务器端主动推送资源，由ResourceRequester视需求创建并调用。实现的方法有两个，push用于向手机发送SYN帧并创建流，datapushHandler是接收到手机端接收数据的确认后的调用的处理函数，下图给出其伪代码。

|  |
| --- |
| // 此函数的功能是服务器端的对资源的主动推送  // 注意：此处的流是手机端发起的资源请求流，  // 主动推送需要在会话层上需要重新创建流  function push(stream, resourceEntry)  // 得到SPDY的会话层，可以在会话层上主动发起流  session <- stream.getsession()  // 压缩资源文件  zipData <- GZipUtils.compress(resourceEntry.data)    // 检查资源体是否和手机端同步  isSync <- DBCacheManager.getSync(resourceEntry.MD5)  // 创建推送的头信息，包含信息：  // URL, MD5, 在本地是否存在，响应头，数据大小，压缩后大小  pushHeader <- newSPDYHeader()  pushHeader.add("URL", resourceEntry.URL)  pushHeader.add("MD5", resourceEntry.MD5)  pushHeader.add("ResponseHeader",  resourceEntry.responseHeader)  pushHeader.add("DataSize", resourceEntry.data.getSize())  pushHeader.add("zipSize", zipData.getSize())    // 如果资源体已经在手机上，则只发送推送头信息即可  if isSync  // 向手机端发送头信息后，关闭此流  session.syn(pushHeader, STREAM\_CLOSE)  else  // 向手机端发送头信息，等待手机端REPLY响应，  // 此后调用处理函数向手机端推送数据  session.syn(pushHeader, datapushHandler(zipData, MD5), STREAM\_NOT\_CLOSE)  // 手机端接受到头信息后，会发回REPLY帧，此时的处理函数  function datapushHandler(zipData, MD5, stream)  // 更新数据库，此资源体已同步  DBCacheManager.ClientMD5Sync(MD5)  stream.data(zipData, STREAM\_CLOSE) |

图16：ResourcePusher的伪代码

对于JCEF，本文进行了针对性的配置，首先修改了浏览器的请求头，将User-Agent修改为手机Firefox浏览器的User-Agent，并根据目标手机的浏览器分辨率进行了分辨率的配置，用来模拟手机的访问。其次重写了浏览器资源请求的方法，将所需请求资源的信息发送到ResourceRequester类，并由该类对Web资源进行请求与记录。

JCEFController用于控制JCEF池，系统在启动时会启动多个JCEF实例，同时按照需求给JCEF分配加载任务，当任务过多时排队等待。在有JCEF加载完成后接收新的任务。需要实现的方法有三个，Initialize方法用于初始化多个JCEF实例，addJCEFSession方法用于给JCEF池分配新的任务，returnJCEF方法被JCEF实例在网站加载完成后调用，释放JCEF接受新的任务，下图给出其伪代码。

|  |
| --- |
| // 初始化JCEF池  function Initialize(PoolSize)  // 初始化等待队列与可用队列  WaitingQueue <- newQueue()  AvailableQueue <- newQueue()  // 初始化JCEF浏览器，并加入到可用队列中  for i=1 to PoolSize  AvailableQueue.push(newJCEF())  // 在用户通过JCEF抓取资源时使用此函数加载URL  function addJCEFSession(URL)  // 从池中找到可用的JCEF，否则等待  JCEF <- JCEFAvailableQueue.pop()  if JCEF exists  // 如有JCEF可用，直接加载URL  JCEF.loadnewURL(URL)  else  // 将URL插入等待队列  WaitingQueue.push(URL)  // 这是浏览器加载完成事件的处理函数  // 当JCEF加载资源完毕时，通过此函数返还原先占用的JCEF  function returnJCEF(JCEF)  // 查询有没有等待中的访问  if WaitingQueue is not empty  URL <- WaitingQueue.pop()  JCEF.loadnewURL(URL)  else  AvailableQueue.push(JCEF) |

图17：JCEFController的伪代码

ServerUpdater类是离线网站抓取系统中的主动类，其中实现了Prefetch函数用于抓取用户所有可能访问的网站中的所有相关资源。

|  |
| --- |
| // 默认JCEF池大小为10  JCEFPoolSize <- 10  // 从网上抓取资源  function Prefetch()  // 从数据库读取用户的网站列表  websites <- DBManager.getUserPreferences()  // 初始化JCEF池  JCEFController.Initalize(JCEFPoolSize)  // 通过JCEF加载所有资源  for each website in websites  JCEFController.addCEFSession(website) |

图18：Prefetcher的伪代码

## 第四章 实验与效果分析

在缓存优化系统实现完成后，本文接下来需要验证通过本文缓存优化系统相比原浏览器直接加载网页存在性能提升。为此，本文设计并进行了如下实验。

5.1 姿态测算实验

5.1.1 实验方案

在本文的实验中，采取以下方案：使用两个相同的手机，连接到同一个无线路由器。一个手机上的浏览器通过配置代理的方式使用本文的系统，另一个手机的浏览器使用默认配置。服务器端首先通过离线网站采集系统采集将要测试的网站，在浏览器缓存清空的情况下，同时使用着两个手机加载测试网站。此后，在手机端有缓存的情况下，立刻再次加载网站。测量两次网站加载的时间与加载网站的手机端传入流量，进行分析与对比。

后文提到的第一次访问的冷启动实验简称为Cold实验，第一次加载后立刻进行得第二次访问的热启动实验简称为Warm实验。

5.1.2 实验结果

本文在Alexa排名前500网站中选取网站，选取原则如下：

* 网站在实现环境下可以访问，列表中域名对应的内容不重复
* 网站在原浏览器和采用本文系统后的浏览器中均可以顺利访问，标志为浏览器的进度条读满，显示正常
* 访问网站均为专为移动设备设计的移动版网站

遵循以上原则，本文选择出50个网站。为了避免某一次访问的结果有较大的偏差，本文对每一个列表中的网站进行10次试验，采集优化前后的各10个数据。因此，本文共得到使用优化前后共2000条加载网站的加载时间与流量数据。

5.1.3 实验结果讨论

5.2 位移融合滤波实验

5.2.1 实验方案

以下为单个手机进行测试的流程图。

  
图19：自动化测试系统流程图

5.2.2 实验结果

实验中共需要收集两个数据，加载时间与加载流量，以下是数据收集的方案。

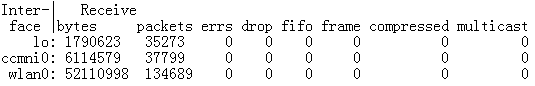
关于网站加载时间，本文定义，网站加载的时间为从网站开始加载到网页触发onload事件的时间间隔。本文的系统中，开始时间为通过Firefox插件控制浏览器加载某一网站的时间，结束时间为该网站触发onload事件，并执行Firefox插件重写的onload处理函数的时间，这两个时间点的差即网页加载所需时间。

关于网站加载流量，本文定义，网站加载的流量为从网站开始加载到触发onload事件5秒之后的网卡传入流量，定一个5秒的空闲时间主要是因为onload的时刻并不一定代表网站所有资源的加载完成，onload事件触发后，有一些需要通过JavaScript加载的资源仍在加载，如果此时就停止加载或者记录流量数据是不符合实际使用情况的，因此本文增加了此延时。

为了获取网卡的传入流量，本文将手机通过USB连接到PC，通过Android Debug Bridge (ADB)的shell命令读取记录Android设备当前网络传输流量的文件，在PC端执行如下命令：

> adb shell cat proc/net/dev

下图为执行该命令后得到的结果示例（对传入部分和没有使用的网络接口进行了省略）：

  
图20：adb shell cat proc/net/dev的结果示例

图中可以看到ADB返回的在执行命令时刻各网络接口的传入数据，其中，lo表示回环接口，ccmni0表示手机移动网络的接口（在未来的3G网络实验中将用到），wlan0表示Wi-Fi网络的接口，是本实验所关心的。在实际应用中，本文在网页加载前后两次执行此命令，对两次wlan0采集到的传入字节数做差，即为加载网页所需流量。

5.2.3 实验结果讨论

如果想控制网页的加载并获得onload事件的触发时间，就需要对浏览器进行一定修改，来控制浏览器的行为，本文是通过编写Firefox插件的方式来实现的。

由于需要调用PC端的命令来获取流量信息，此处无法仅通过Firefox插件实现，因此需要在PC端搭建程序，并让Firefox插件与PC端程序协同工作。针对浏览器本身易于使用HTTP方式发送并接受信息的特点，本文设计了以下架构：

在PC端搭建Web服务器，浏览器加载此服务器上的Web网页（即通过HTTP的GET方法加载），以参数的方式向服务器报告网页加载时长与时刻，同时可以从服务器请求下一个加载的网站与是否清除缓存的指令。为了实现多设备同时测试，本文在每个设备发送的参数中添加一个设备编号，即可以在服务器端区分每个设备的测试。

手机端可以传入的参数有以下三种：

* devid: 表示设备编号
* type: 值为begin表示测试开始，为end表示测试结束
* time: 当type=end时，用来表示这一次加载所需的毫秒数

5.3 特殊事件下的系统稳定性实验

5.3.1 实验方案

Firefox插件可以为Firefox浏览器添加新功能，例如管理标签页、更改Web内容、提高安全性等。Firefox插件的开发语言是JavaScript，其实现的效果也相当于是在浏览器里执行JavaScript代码。本文需要开发的是Android平台上Firefox的插件，其开发细节与开发普通Firefox浏览器的插件略有差异。

根据设计，在用户点击开始测试后，浏览器监听每一个页面的onload事件，浏览器总共会加载三种网页：测试网页、开始网页、结束网页。其中开始网页和结束网页是用来和服务器通信，发送测试结果并取得测试控制信息的。以下给出了Firefox插件的伪代码。

|  |
| --- |
| // 获取测试服务器提供的ServletURL  // 如：http://162.105.51.254/TestServlet  ServletURL <- getServletURL()  // 记录加载页面开始的时间戳  StartTime <- null  // 记录当前设备编号  DeviceID <- getDeviceID()  //点击浏览器的“开始测试”触发的函数  function ClickStart()  {  // 监听浏览器的onload事件，事件处理函数为onPageLoad()  Browser.addEventListener("load", onPageLoad)  // 向测试服务器发送开始信息，请求下一个网站  loadBegin()  }  //onload事件处理函数  function onPageLoad()  {  // 获取onload完成的网站URL  URL <- Browser.currentURL  // 判断加载完成的是目标网站还是服务器指令  if URL.contains(ServletAddr)  // 判断载入的是加载开始还是加载结束  if URL.contains("begin")  // 读取网页内容，即下一个要载入的URL  nextURL <- Browser.getDocument.getBody()  // 记录开始加载的时间  StartTime <- getTime()  // 加载下一个网站  Browser.loadURL(nextURL)  else if URL.contains("end")  // 读取是否需要清理缓存，如果需要则清理  CacheControl <- Browser.getDocument.getBody()  if CacheControl = "clear"  Browser.cleanCache()  // 开始下一次测试  loadBegin()  else  // 计算出加载所需时间  loadTime = getTime() - StartTime  // 等待5秒钟  setTimeout(5000)  // 测试结束，传输加载时间  loadEnd(loadTime)  }  // 测试开始  function loadBegin()  {  Browser.loadURL(ServletURL +  "?type=begin&devid=" + DeviceID);  }  // 测试结束  function loadEnd(loadTime)  {  Browser.loadURL(ServletURL + "?type=end&time=" +  loadTime + "&devid=" + DeviceID);  } |

图21：Firefox插件的伪代码

5.3.2 实验结果

本文采用Servlet搭建测试控制Web服务器，Servlet在Tomcat服务器上运行。服务器的主要工作是在测试开始和结束时调用ADB采集流量信息，在测试开始时提供加载的网站，在结束时收集访问时间，计算访问流量并将测试结果写入到文件中。以下给出Servlet的伪代码。

|  |
| --- |
| // 通过GET方法访问Servlet时的调用函数  function doGet(request, reponse)  // 获取设备编号  devID <- request.getParameter("devid")  // 以字符串读取到相应设备的adb传入的全部流量数据  wlanData <- getNetStatFromADB(DevID)  // 对数据进行文本分析，提取出Wi-Fi的传入流量  rx <- Extract\_wlanRx(wlanData)  // 判断此次GET是开始测试或者结束测试  if request.getParameter("type") = "begin"  // 得到该设备下一个访问的URL  nextURL <- getNextURL(devID)  // 向手机端以HTML内容的方式发送URL  response.sendHTMLbody(nextURL)  // 记录设备访问前的传入数据量  RecordRx(devID, rx)  else if request.getParameter("type") = "end"  // 从参数获得加载时间  loadTime <- request.getParameter("time")  // 将此时的数值与开始时的数值做差得到数据量  loadRx <- getLoadRx(devID, rx)  // 向设备发送是否清理缓存的指令  response.sendHTMLbody(isCleanCache(DevID))  // 记录本次访问的数据  output(devID, loadTime, loadRx) |

图22：自动化测试Servlet的伪代码

5.4 小结

本节首先给出实验数据的处理方法，定义了优化率作为评价优化程度的指标。之后从各个角度描述实验结果，最后讨论实验见过，对实验中优化程度很低的测试进行更深一步的具体分析

本文首先定义优化率，用来优化后相比于优化前的性能提升水平。下式表示的是时间优化率的计算，将加载时间变为加载流量即可得到流量优化率。

在实验中，本文测量加载时间与加载流量，这两个数值越低，说明访问效果越好。根据公式，优化后的测试结果越低，优化率越高。因此采用该优化率评价系统的优化能力是合理的。

为了避免外在因素造成偏差较大的结果，本文对于每一个网站的第一二次访问实验均重复了10次，对于这10次测量结果，本文首先计算出在每一次测量中时间与流量上的优化率，再对这10个优化率取中位数得到最终该网站在某一次访问实验中的时间或流量优化率。对于50个网站的实验，可以得到200个优化率结果。

通过上一节的方法，本文得到针对每个网站的四个实验的优化率：

* Cold, Time：第一次访问的加载时间优化率
* Cold, Data：第一次访问的传入流量优化率
* Warm, Time：第二次访问的加载时间优化率
* Warm, Data：第二次访问的传入流量优化率

针对每个实验所测的50个网站的优化率，本文绘制了优化率的累积分布函数图，即CDF图，如图23所示。

图23：优化率的累积分布函数（CDF）图

除此之外，本文对数据进行了初步统计，每个实验的50个结果的平均值与中位数见下表：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 优化率 | 平均值 | | 中位数 | |
| Time | Data | Time | Data |
| Cold | 28.9% | 17.1% | 29.6% | 11.2% |
| Warm | 9.0% | 66.4% | 10.9% | 74.5% |

图24中，直接对实验中所测得网站的消耗流量与加载时间绘制了CDF图，可以更直观的看出，采用了本文的缓存优化系统后，访问网站的加载时间有明显的减少，传输的流量也有明显的减少。

|  |  |
| --- | --- |
|  |  |
|  |  |

图24：加载时间与加载流量的CDF图

虽然整体结果表明，本系统整体上对访问网站有明显优化，但在某一些网站加载的测试结果中，出现了负优化率，有一些数值较大，以下给出原因的分析。

第一种情况是本身加载时间非常短。如m.360.cn网站，在第二次访问中，原网站加载的时间为0.18秒，而优化后加载时间为0.34秒。用户无法在访问中感受到这两个时间的任何差异，然而计算出的优化率却为-90%左右，对实验结果有较大影响。

第二种情况是由于HttpClient加载失败引起的。在cn.bing.com网站的第二次访问实验中，原加载方法的时间为0.4秒，优化后加载时间为2.1秒左右，此时，计算得到流量优化率为-400%左右（使第二次访问平均速度优化率降低了8%）。由此对第二次加载的时间平均值造成非常大的影响。通过反复回放发现，原因在于，有一些资源无法通过HttpClient正常获取，而通过浏览器本身的资源加载方式则可以获取。这种情况发生时，HttpClient会在2秒的超时时间经过后返回资源获取失败，而此时计算得到该网站的加载时间就在2秒以上了。但Firefox浏览器的原生请求中，该资源可以在短时间内正常获取，这就造成了实验中的情况。而为什么HttpClient无法正常获取的问题根源目前还没有发现。

## 第五章 工作总结与展望

6.1 本文工作总结

本文的主要工作包括：

* 介绍了当前浏览器的缓存机制与潜在的缺陷，分析了缓存的理论性能上限与当前性能间的差距及造成的原因，验证了实现浏览器的缓存优化是有意义而且可行的，并提出了优化方案。
* 针对提出的优化方案，本文设计并实现了基于代理的移动浏览器缓存优化系统，该系统有三个子系统组成：手机端后台服务、服务器端在线服务、服务器端离线资源抓取程序。该系统通过移动设备端的后台服务和云端服务器对浏览器请求资源的记录、选择性推送与优化传输，实现了对现有缓存机制的优化。
* 设计实验验证了本文的缓存优化系统对提升网页加载速度和节省流量的有效性。实验表明，该系统在用户第一次访问网页时，平均可以节省流量17%，提高加载速度29%；立刻再次访问同一网页，平均可以为第二次访问节省流量66%，提高加载速度9%。

6.2 未来工作展望

在本研究的基础之上，未来可以从三个方面继续进行研究：

1. 以此研究成果作为平台，继续优化浏览体验

本文所实现的系统在开发的过程中一直本着高扩展性的原则，该系统今后可以作为基于资源的移动浏览器优化实验平台，可以在工作的基础之上再进一步地丰富功能，比如说：

* **更细粒度缓存的实现：**现有的缓存是以资源为粒度的，研究表明，可以通过更细粒度的缓存来进一步提高流量的节省，如对HTML中的每一行进行缓存。
* **资源推送优先级的实现：**目前系统推送资源为随机顺序推送，然而，良好的推送顺序能使浏览器更快地加载资源。在资源列表的构造过程中，可以构建出访问一个网站的资源树。在浏览器加载的过程中，只有浏览器加载了根节点资源，其子节点的加载才变为可能。由此可以优先推送位于根节点的资源，提高加载速度。
* **网站预取的实现：**当用户处于Wi-Fi环境下，服务器可以将用户可能接下来访问到的资源自动推送到手机端，在用户下一次访问这些资源时可以进行无流量的浏览。

1. 继续进行试验，验证缓存优化的效果

还需要测量不同时间间隔的缓存优化效果来验证系统的性能。目前实验中的第二次访问是在第一次访问后立刻执行的，接下来可以做相隔半小时、一小时、两小时等不同市场的优化效果。这个实验更符合用户的使用行为，更有意义，然而进行此实验存在很大的技术难题尚未解决。

1. 利用此前制作的应用商店App，开展用户实验

此前，作者制作了应用商店App用来开展用户实验，其功能是将常用网站的快捷方式放在手机的桌面上，便于用户快速打开这些网站。本文的系统可以与该App简单地结合到一起，让用户体验到本系统优化浏览的效果，同时收集用户的访问数据，并进一步发现系统的可提升之处。

## 参考文献

[1]中国互联网络信息中心. 第35次中国互联网络发展状况统计报告[R].2015

[2] Y. Zhu, V.J. Reddi. WebCore: Architectural support for mobile Web browsing[C]. ISCA2014

[] N. Trawny, S.I. Roumeliotis: Indirect Kalman Filter for 3D Attitude Estimation. : <http://www-users.cs.umn.edu/~trawny/Publications/Quaternions_3D.pdf>

[]

## 致 谢

感谢杨芙清院士、梅宏教授和北京大学软件工程研究所，为我提供了宽松自由的学习与研究环境，使我有机会接触并参与软件工程领域的研究项目，让我学习到很多书本上学不到的内容，为我未来的工作打下了坚实的理论基础，并积累了极富价值的工程经验。

感谢网构中间件小组的黄罡老师和刘譞哲老师。在黄罡老师开设的软件工程实验班上，我参与设计并实现了第一版缓存优化系统，在此过程中，两位老师给了我悉心的指导，使我对这个项目产生了浓厚的新区与深厚的感情。非常感谢他们给了我机会，让我可以在毕业设计中继续进行移动浏览器改进的项目。特别感谢我的导师刘譞哲老师，在百忙之中为我的研究指引方向，关心我的工作进度，在我感到懈怠时督促并鼓励我。在撰写论文与准备答辩的过程中，刘老师为我提供了大量有价值的写作建议，并亲自为我修改论文，付出了大量的时间与精力。

感谢网构中间件小组的师兄师姐，为我提供了良好的学术氛围与研究环境。特别感谢马郓师兄，他之前所做的研究是本文工作的重要前提与依据。在我毕业设计的工作中，他不遗余力地对我的工作进行指导，帮助我解决学习与生活上的各种问题，为我提供很多宝贵的研究思路和技术思路，每天花费大量时间和我讨论工作，时常忙碌到深夜。即使师兄远赴国外开会也一直挂念着我的工作，和我保持交流。在撰写论文的过程中，他帮我耐心修改论文的整体结构，逐字逐句地找问题。没有师兄的帮助，我的工作将很难达到现有的高度。感谢陈镇鹏师弟，在忙碌的期中考试周，他利用宝贵的时间帮我完成了部分实验。

感谢大学四年来知道的我的所有老师。老师们的言传身教，让我拓宽了眼界，使我在计算机领域打下了良好的基础。

最后，衷心感谢我的家人，尤其是我的父母。你们一直在背后默默支持我、鼓励我，给我无微不至的关怀，是我大学四年工作与生活中最坚实的后盾。