Scratch:一个基于Chrome浏览器的用户操作捕捉与回放工具

陈萧宇 黄 震 刘譞哲 黄 罡 张 颖
(北京大学信息科学技术学院软件研究所 北京 100871)
(高可信软件技术教育部重点实验室 北京 100871)

摘 要 目前,浏览器已经成为人们接入互联网最主要的入口。HTML5、JavaScript和CSS等Web技术的发展,也极大地增强了Web应用的功能并丰富了用户的交互体验。但是,随着Web应用逻辑日益复杂,人们既要经常完成过程繁琐且需重复执行的任务,也要完成操作复杂且难以记忆的任务。因此,基于目前流行的Chrome浏览器,设计实现了一个动作捕捉与回放工具Scratch(Smart Capture-and-Replay at Chrome),以实现对Web应用操作正确、一致的捕捉与回放,并允许其他用户对已有操作记录进行定制。

关键词 捕捉与回放,HTML5,Chrome浏览器,JavaScript
中图法分类号 TP317.1 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2014.11.022

Scratch:Tooling Support for Capture-and-replay of User Actions in Chrome Browser

CHEN Xiao-yu HUANG Zhen LIU Xuan-zhe HUANG Gang ZHANG Ying
(Institute of Software,School of Electronics Engineering and Computer Science,Peking University,Beijing 100871,China)
(Key Laboratory of High Confidence Software Technologies (Peking University),Ministry of Education,Beijing 100871,China)

Abstract Modern browsers,such as Mozilla FireFox and Google Chrome,are equipped with numerous powerful facilities like plug-ins and add-ons,which significantly enrich the user experiences on the Web. However,as Web applications get more complicated day by day,many tedious processes must be performed frequently,while others which are complex or hard to remember are done less frequently. This paper presenbed the design of Scratch (Smart Capture-and-Replay at Chrome),a collaborative Programming-by-Demonstration (PBD) system for capturing,recording,editing,and playing back the user interactions and sharing user experience in Chrome Web browser,which greatly enhances people's efficiency.

Keywords Capture and replay,HTML5,Chrome browser,JavaScript

1 引言

Web应用已经成为互联网环境下最为重要的应用形态。随着HTML5、JavaScript和CSS等Web技术的飞速发展,Web应用的交互体验不断增强,应用功能逻辑更加复杂。而用户操作Web应用时也面临着新的问题,例如很多Web应用,如网络教学系统、客户关系管理系统等,其工作流程通常比较复杂,涉及步骤较多,对于初学者而言难以很快掌握;再如,用户经常会进行一些重复性的操作,如重复添加信息等,这些工作通常比较琐碎,当重复次数较多时容易出错。

一个自然的解决方案是:将用户的操作记录下来进行分析,允许其他用户对这些操作记录进行按需定制(如控制重复次数等)并自动回放,这能够极大地提高工作的效率。

已有很多工作实现了用户操作的捕捉与回放,但是这些工作大多是面向规整的、具有相对固定程序和静态结构的应用,而新型Web应用则基于可动态变化的DOM(文档对象模型),同时具备HTML5的诸多特性(如本地存储、离线模式等),结构和行为更加复杂。同时,如何更好地将这些操作以相对简单和直观的方式展示出来,方便用户定制和复用,也是值得考虑的问题。这就对现有的操作记录和回放工具带来了新的挑战。

针对上述问题,本文实现了一个基于Chrome浏览器的用户操作记录和回放工具Scratch(Smart Capture and Replay at Chrome),实现了面向动态Web应用的捕捉与回放机制。本文的主要技术贡献在于:

- 提出了一种Web应用操作捕捉与回放技术,实现了对动态DOM结构和内容的捕捉,保证回放的正确性和一致性;
- 针对不同回放场景,实现了在线即时回放和离线快照回放两种机制。

本文第2节介绍相关研究工作;第3节介绍Scratch系统

到稿日期:2013-09-01 返修日期:2013-11-01 本文受国家重点基础研究发展规划973项目(2009CB320703),国家高技术研究发展计划863项目(2011AA01A202),国家自然科学基金(61121063,61003010,U1201252)资助。

陈萧宇(1990-),男,硕士生,主要研究方向为Web前端工程、浏览器插件,E-mail:roxasxiaoyunchen@pku.edu.cn;黄 震(1993-),男,主要研究方向为Web应用、浏览器插件;刘譞哲(1980-),男,博士,副教授,主要研究方向为服务计算、Web工程和移动计算等;黄 罡(1975-),男,博士,教授,主要研究方向为软件中间件、操作系统和软件工程;张 颖(1984-),男,博士,主要研究方向为软件工程、软件中间件、自治计算、分布式计算等。

的总体设计思路，分析主要的技术难点；第 4 节介绍 Scratch 系统的实现，包括面向动态 Web 应用的动作捕捉与回放定位、在线和离线回放模式等；第 5 节通过实例验证了 Scratch 的性能，结果表明 Scratch 占用较少的浏览器资源，不影响用户体验；最后总结全文并展望未来工作。

2 相关工作

2.1 动作捕捉与回放

捕捉与回放(Capture & Replay，也称记录与回放(Record & Replay))可将一组序列化的动作记录下来，然后自动化地加以重放。捕捉与回放技术的本质是将人的经验或技巧通过计算机程序的方式固化下来并自动化执行，提高了任务执行的效率，被广泛用于程序调试、协同工作、计算机辅助教学等领域。早期的捕捉与回放主要应用于桌面应用，程序结构相对比较静态和固定。例如，基于 Eclipse 的图形界面测试工具 TPTP[1] 记录 GUI 程序测试员的动作，将动作回放以测试 GUI 的运行是否正常；JTutor[2] 则记录教师逐个创建文件、逐行编写代码的过程，再通过原样回放给学生，让学生观看编程的过程。学生可以将编程过程的脚本下载到自己的机器上，在自己的机器上回放，观看程序运行的效果；SheepDog[5] 则试图从大量的用户对 IDE 的操作序列中挖掘和总结出特定的序列，以生成自动化的编程向导(Wizard)。

随着 Web 应用近年来的迅速发展，面向 Web 应用的捕捉与回放技术也得到了重视。例如，Koala[6] 是一个基于 FireFox 浏览器的插件，可以记录和回放用户在 Web 应用中的操作动作；MugShot 系统[4] 则针对微软 Internet Explorer 浏览器提供了捕捉与回放支持。但是，现有大多数面向 Web 应用的记录回放技术都存在一定的问题。首先是对 Web 应用的变化性考虑不足，当应用结构和内容发生变化时，并不能保证回放的一致性；其次，FireFox 浏览器采用的是单进程多线程架构，Koala 插件运行在浏览器主进程相同的地址空间，一旦发生问题则必须重启整个浏览器，而重启操作会对其他应用产生影响。而 MugShot 则是特定于 IE 浏览器设计，对于 ECMAScript 标准[8] 并不完全兼容，因此该方案难以推广。

2.2 Chrome 浏览器及其插件机制

Chrome 是由 Google 开发的浏览器。截至 2013 年 5 月，Chrome 浏览器已经成为全世界范围内用户数量最多的浏览器，并且其市场份额仅次于 Microsoft 的 IE 浏览器。Chrome 浏览器提供了丰富的扩展机制，允许第三方开发者开发的插件动态安装和卸载。Chrome 插件开发的主要语言是 JavaScript，通过后台的 JavaScript 程序控制插件本身的功能，插件的函数实现体被直接导入网页中，以获得网页中的内容或者修改其内容。需要特别注意，和 Mozilla Firefox 浏览器不同的是，Chrome 采用多进程，这种特性能保证插件的运行对于浏览器主进程的影响较小。

3 Scratch 系统设计

基于上述分析，本文设计的 Scratch 系统将实现为 Chrome 插件，对待记录网页动态注入 JavaScript 代码以实现动作捕捉，插件还负责与浏览器交互、发送请求给服务器等。需要说明的是，Scratch 的核心逻辑(动作捕捉与回放)均以 JavaScript 实现，仅使用了 Chrome 向目标网页注入 JavaScript 文件的 API。其他浏览器如 FireFox 也提供了类似机制，因此 Scratch 经过简单修改后也可以移植到其他浏览器上。

如图 1 所示，Scratch 内部包含如下功能模块：

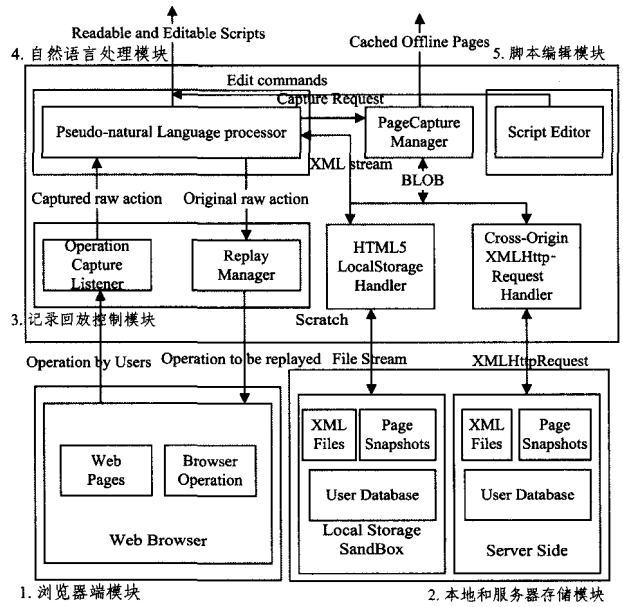


图 1 Scratch 总体架构

1)浏览器端模块负责捕捉用户的交互信息。其主要捕捉两种类型的信息：来自网页内部的操作信息和来自浏览器内部的操作信息。这两类信息需要使用不同的机制捕捉获取，然后在汇总后上传给记录回放控制模块。

2)本地和服务器模块用于存储捕捉到的信息，唯一不同在于本地存储是利用 HTML5 LocalStorage 特性保存在本地，而服务器是利用 HTTP 请求保存在服务器端。这种按需的存储方式可以支持在线和离线两种回放模式。其中，XML 文件记录操作序列和相关的语义信息，是整个捕捉和回放的基础；网页离线缓存文件保存用户当时操作网页的内容，这样做的目的是为了实现离线回放功能；用户数据库负责保存用户的信息和每个用户的脚本信息，用于隐私保护和脚本分享功能。负责给这个模块传输消息的是 XMLHttpRequest Handler，它是插件和服务器/本地沙箱(SandBox)之间的一个沟通桥梁，在它们之间传输消息和文件(如 XML 文件、用户信息、离线网页缓存文件等)。

3)记录回放控制模块是 Scratch 的核心模块，负责高层次的捕捉和回放的控制。Operation Capture Listener 负责捕捉用户的交互信息，根据动作的特征对其进行分类并且把它们转换成原始的文本脚本形式；当进行操作回放时，Replay Manager 模块负责控制和执行进程，并且提供在线回放和离线回放两种功能。

4)自然语言处理模块同样部署在 Scratch 内部，负责将记录到的操作用一种容易理解的类自然语言格式展现出来，以便用户编辑和共享。Pseudo-natural Language Processor 模块得到文本类型的脚本后，进一步把它们翻译成为可读性更好的类自然语言，并且生成一个 XML 文件流来保存被记录下来的动作。

5)脚本编辑模块允许用户利用 script editor 编辑自然语言脚本(设置循环或者断点、改变参数等)。通过这种方式，便可以把一个简单的操作过程进行编辑，成为适用于各种情况、各种用户需求的操作过程。

在上述功能模块中，我们需要重点解决几个技术挑战：1）在记录动态网页（如包含 AJAX 请求）时，如何定位到需要的 DOM 元素；2）如何在操作和可读且可编辑的语言规范之间进行转化；3）应该提供什么样的回放模式来满足不同用户的需求。

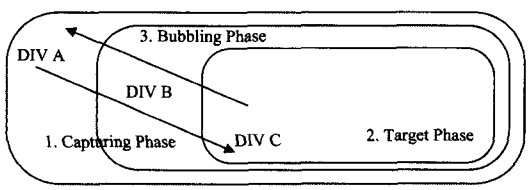
4 Scratch 系统实现

4.1 捕捉用户交互动作

总体上看，Scratch 需要捕捉两种操作行为：网页操作和浏览器操作行为。这两种类型的操作可能同时发生，导致最终得到的序列非常冗长和琐碎。于是我们需要简化整个过程，来增强记录脚本的可读性和用户友好性。在 Scratch 中，只有“源操作”（动作产生的作用源）才会被记录在脚本中。例如，点击一个链接有可能会导致点击网页的动作被捕捉到，也可能同时导致打开一个新的标签页动作被捕捉到，但是最终的脚本只应该显示为：“点击一个 XX 链接”，因为打开新的标签页等操作只是这个源操作的后续附加行为。为了实现这个功能，我们需要停止 DOM 节点的事件冒泡（Bubbling）机制，并且设置一个控制单元来避免网页操作和浏览器操作同时被记录下来。

因此，捕捉 DOM 事件的时机便成为一个关键的问题。不可能等到浏览器完全把网页加载完成后才捕捉，因为某些复杂的网页加载可能会非常耗时，并且用户有可能已经开始在该网页上进行下一个操作行为了；也不能太早地开始捕捉，因为有些 DOM 节点可能还没有被载入，导致记录失效。

Scratch 通过扩展 DOM Level 2 事件模型来解决该问题：每一个 DOM 事件都有 3 个分发的阶段：捕捉阶段（capturing phase）、目标阶段（target phase）和冒泡阶段（bubbling phase）。在捕捉阶段，Window 和 Document 类会首先得到这个事件，然后从〈html〉节点一直往下传送，直到到达目标的 DOM 节点；在目标阶段，事件被移交给目标 DOM 节点并且触发其上的事件处理程序；在冒泡阶段，事件从 DOM 树的叶节点传送回根部。值得注意的是，这 3 个阶段是依次产生的，这意味着在执行相应的处理之前（在目标阶段执行），已经能捕捉到相应的事件（在捕捉阶段）。图 2 展示了这个过程。




```
〈div id='A'〉
〈div id='B'〉
〈div id='C' onclick='handlerC()'〉〈/div〉
〈/div〉
〈/div〉
document. getElementById ('A'). attachEventListener ('click', handlerA, true);
document. getElementById ('B'). attachEventListener ('click', handlerB, false);
```

其中有 3 个 DIV，分别标记为 A、B、C。如上这样定义事件处理函数，handler A 会在捕捉阶段被触发，handler B 会在目标阶段被触发，最后 handler C 会在冒泡阶段被触发

图 2 DOM Level2 事件模型

利用 DOM Level2 事件模型的上述机制，我们可以为 Window 类添加一个事件处理函数，并且用addEventListener() 函数将其声明为捕捉阶段被触发。由于 Window 类从页面加载一开始就已经存在了，无论是 DOM 节点何时被加载，与之相关的所有事件都能被捕捉到。

4.1.1 脚本语言设计

动作序列脚本包括 3 个主要元素：操作动作本身、目标类型和语义信息。由于 HTML 标签属性往往不规范，因此很难探索出一个通用的方法来提取出特定 DOM 节点中所需要的语义信息。图 3 展示了 HTML 标签属性。

```
Button 1：〈button〉OK〈/button〉
Button 2：〈input type="button" value="OK"〉〈/input〉
```

图 3 不确定的 HTML 标签语义位置

尽管这两个按钮功能相同，但是它们在代码上却截然不同。因此当获取这个按钮的内部语义时，也就是找到“OK”这段文本时，这两个按钮必须要区别对待。表 1 列出了常见操作动作和 HTML 标签的语义信息位置。

表 1 常见的 HTML 标签操作语义及位置

操作类型	目标节点类型	语义及位置
鼠标点击	Link	Attribute：InnerText
鼠标点击/鼠标选中	Paragraph/Highlighted Text	Attribute：InnerText
鼠标点击	Image	Attribute：Src
鼠标点击	Button	Attribute：InnerText
键盘编辑＋属性值 鼠标点击＋属性值	Input type＝text Input type＝password/ Input type＝checkbox Input type＝radio/Select/ Option	Function： NearestContext()
鼠标点击	Input type＝button/ Input type＝submit/	Attribute：Value

有时仅从单一的 DOM 节点是无法得到充分的语义信息的，因为描述性的信息刚好在其它的节点中，尤其是那些附近的节点：比如对于一个文本编辑框节点〈*textarea*〉，它内部不包含任何内容，因为它本身就是等待用户输入的一个节点。它的语义信息应该从附近的节点，比如〈*label*〉节点中寻找。更一般的情况是在任何的用户登录网页界面中，常常会发现有输入框来让用户输入用户名和密码信息，但是这些指示性信息并不在输入框中，而在一个标签节点（〈*label*〉）或者段落节点（〈*p*〉）中。它们常常出现在附近（且大多数情况位于左边，因为人们的读写习惯通常是从左往右的）。为此，Scratch 设计了一个函数“NearbyContext”来帮助寻找，其代码如图 4 所示。

```
function NearbyContext(domNode)
{
//maxloop determines the max distance for finding context
  for i<－0 to maxloop
      while (domNode has left or right sibling)
    do
      if left sibling has semantic context t1
        then result <－ t1, return
      else if right sibling has semantic context t2
        then result <－ t2, return
      domNode ＝ domNode. parentNode//get result from parent
  return result;
}
```

图 4 获取邻近节点信息

4.1.2 脚本存储

Scratch 利用 XML 语言格式来格式化保存脚本，其格式图 5 所示。

```
〈footprint〉
  〈tag〉the name of the target object〈/tag〉
  〈operation〉the operation taken〈/operation〉
  〈value〉the semantic value of the target object〈/value〉
  〈url〉the url of the action〈/url〉
  〈keyattr〉the key attribute of the DOM object〈/keyattr〉
  〈domnum〉the DOM sequence in numbers〈/domnum〉
  〈domname〉the DOM sequence in names〈/domname〉
  〈context〉the context of the DOM object〈/context〉
  〈text〉the textual script language〈/text〉
〈/footprint〉
```

图 5 记录脚本格式

其中需要说明的是：(1)“*keyattr*”标签用来保存 DOM 节点的重要属性，比如 ID、Name、Href 等。其所记录的属性很大程度上取决于 DOM 节点的特征，比如 src 属性对于链接标签重要，但是 value 属性对于输入框更为重要。(2)“*domnum*”标签是一个数组，该数组包含着 DOM 树中从顶端到达目标节点的轨迹，以子节点的序号作为唯一的标识。相似地，“*domname*”标签也是一个数组，用来表示 DOM 树中的轨迹，但是只是用标签的名字作为唯一的标识符记录下来。(3)“*context*”标签中的内容是用于提供关于目标节点的语义信息的，也就是上一节中提到的语义信息。所有这些内容都是在回放脚本时用来定位目标节点位置的重要信息。

4.1.3 与服务器的交互

在大多数浏览器中，出于安全性的考虑，用 JavaScript 将一个域名下的信息传送到另外的域名下是被严格禁止的(即所谓的“同源策略”)。Scratch 是以插件形式存在的，因此具备浏览器层的操作权限。换言之，Scratch 有权限访问当前浏览器打开的每一个网页。在 Chrome 的插件 API 中，其提供了相应的机制来支持跨域的访问，只需要在插件的 manifest 文件中声明访问的权限，并且利用其提供的机制把 JavaScript 脚本注入到网页中即可。在声明权限时，需要把主机的名称或者主机名称的匹配符输入进 manifest 文件中，然后插件就可以从该网页申请跨域访问到远程服务器。

在服务器端，Scratch 创建了一个用于用户管理和脚本管理的数据库，以保存所有用户上传的脚本文件。除脚本文件外，服务器端还可能保存有离线快照回放时所需要的相关网页文件。

4.1.4 本地存储机制

很多时候用户可能会面临无法访问到网络(比如在飞机上继续工作)的问题，同样，也有很多情况下用户出于安全性和隐私的考虑，在保存脚本时不想上传到服务器与别人分享，而只是在本地自己使用。因此，Scratch 设计了本地存储以支持离线回放。

在 HTML5 规范中，允许浏览器有限度地访问本地文件系统。Chrome 浏览器在本地文件系统中划分出一个小的存储区域，作为一个本地的沙盒。该沙盒是一个隔离的独立区域，它不能直接地从本地磁盘访问。同样，网页的插件和应用也不能写入到除了该区域外的其他磁盘区域。

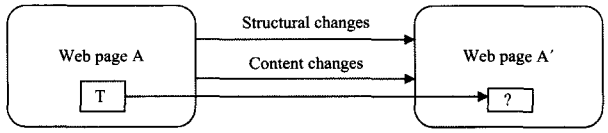


图 6 动态网页中的节点定位

在 Scratch 中，当用户保存脚本时，可以选择保存在本地的存储沙盒中或是保存到在线的服务器上。如果他们选择本地化保存，脚本会被保存到本地 PC 上。尽管没有互联网的访问会导致该离线的网页中的表格无法被提交并且链接无法被打开，但是用户还能使用 Scratch 的回放机制(只要当前的 URL 还是保持不变)。这对于一些复杂并且耗时的单网页操作来说非常有意义：用户可以在网络良好的时候一直记录网页操作并且保存脚本到本地。当网络断开时，如果用户工作还没有完成，只需要通过 Scratch 打开离线缓存好的网页并且把之前的操作回放一遍，就可以完全恢复断网之前的网页状态。此时，用户可以继续在该离线网页上的工作，并且在网络恢复正常的时候提交修改的内容或者打开新的链接。

4.2 回放用户的操作

由于网页的不稳定性和动态变化的特征，在网页上正确并且一致地回放用户的动作是一个需要考虑的问题。经过仔细分析后，我们发现了在网页上的两种主要变化：1)结构性变化；2)内容性变化。结构性变化通常是指改变了 DOM 节点的位置和外观，但同时保持其内部的语义信息基本不变；内容性变化意味着对于语义内容有着重大的改变，甚至整个 DOM 节点都被移除了。

针对这两种变化，Scratch 提供两种不同的回放机制：在线即时回放机制，以克服结构性的变化；离线快照回放机制，以解决内容上的变化。

4.2.1 在线即时回放机制

在线即时回放，顾名思义就是在真实的环境中和即时的网页中回放。由于上面讨论的变化，该网页很有可能与原先捕捉记录阶段的原网页不一样。这些变化可能会给我们的重定位动作所作用的 DOM 节点带来问题。

为了完成定位，需要来自原网页(记为网页 A)的 3 种类型的信息。首先，需要原始的位置信息(节点 T 在网页 A 中的位置信息)。尽管在新的网页(记为网页 A')中，目标节点的位置有可能会改变，但是这一类信息依然是有价值的，前提是位置的改变在一定的范围内。事实上，由于网页的内容导向的划分区域方式，大多数的位置变化都被限制在网页的一个特定区域中。因此新的 DOM 节点就与旧的 DOM 节点共享了很多共同的父节点。这样一来，原始的位置信息就为我们重新定位新的位置提供了重要的参考信息。Scratch 采用从 DOM 树的根节点到目标节点的路径信息来表示位置。其次，需要原始的 DOM 节点内部的信息(节点 T 内部信息)。这些信息包括该节点的重要属性(在“*keyattr*”中保存)和该节点的语义信息(在“*value*”中保存)。这些信息非常重要，因为它们能很好地与别的节点区分开来。Scratch 用该标签的语义信息和重要属性来表征这一类的信息。最后，在网页中有时会存在内部信息一模一样的两个节点，这样若利用第二类信息来匹配，则准确性将会大大降低。于是需要该节点附近的上下文信息(在 T 周围的节点的语义信息)。我们使用从原始节点周围捕获的语义信息来表征这一类信息。

Scratch 设计了一种匹配算法来寻找在新的网页中与原始 DOM 节点最匹配的节点。由于所有的信息都以字符串形式组织，我们采用字符串编辑距离（Levenshtein distance）函数来作为度量函数。

两个字符串 a,b 之间的字符串编辑距离 $Lev_{a,b}(|a|,|b|)$ 的定义如下：

$$lev_{a,b}(i,j)=$$

$$\begin{cases}\max(i,j) & \text{if } \min(i,j)=0 \\ \min\begin{cases}lev_{a,b}(i-1,j)+1 \\ lev_{a,b}(i,j-1)+1 \\ lev_{a,b}(i-1,j-1)+[a_i\neq b_j]\end{cases} & \text{otherwise}\end{cases}$$

因此，最佳的匹配结果定义如下：

$$TarNode=\text{Max}(a*F_1(T,T')+b*F_2(T,T')+c*F_3(T,T'))$$

where $(a+b+c=1)$ and $a,b,c\in(0,1)$

$$F_j(T,T')=1-Lev(T_j,T_j')/\text{Max}(len(T_j),len(T_j'))$$

其中，T' 代表所有可能的待匹配节点。T_1 表示位置信息，T_2 表示语义信息，T_3 表示上下文信息。参数 a,b,c 是为不同种类的信息赋予的权值，经过测试后，较好的 a,b,c 权值应该分别是 0.2,0.5,0.3。

但是，在有限的时间内，筛选算法应该尽量快而且占用更小的内存空间。考虑到最好的方式应该是限制待匹配节点的数量，我们设置了 3 个筛选步骤用于排除不可能的目标节点：

1）排除所有在内部语义内容上存在较大差异的节点（大于某个阈值）；

2）在第一步的基础上，排除在重要标签属性上存在较大差异的节点；

3）在前两步的基础上，再根据公式计算出最终的编辑距离；

4）如果在前面任何一步中有且仅有一个完美匹配的节点（即编辑距离为 0），则可以跳过后续的匹配步骤并认定该节点就是最终的结果。

一旦找到了最终的结果，Scratch 在回放时就会把目标 DOM 节点在新的网页中高亮出来，显示地告诉用户匹配的结果，并执行相应的操作。如果最终的目标节点编辑距离过大（大于某个阈值），则说明即使最优的节点与原始 DOM 节点也存在较大差异。这时我们认为该匹配失败并且抛出错误，让用户尝试离线快照回放机制。

类似播放器，Scratch 中也支持顺序回放（不停止地从脚本开始回放到结尾）、单步回放（每次只执行一个动作）、暂停、恢复以及跳步（任意选择回放的下一个动作）。另外，Scratch 还可以设置循环、参数和断点，以执行重复性工作或由用户自定义新的动作。用户只需要记录一遍动作的序列，并且以脚本的形式将其保存下来，然后编辑脚本，为其设置循环、断点和参数的输入点即可。

4.2.2 离线快照回放机制

如上所述，在回放动态网页的内容时，Scratch 在定位 DOM 节点失败之后会报出错误，并且发出提示性的消息给用户。一般来说，定位失败的原因是网页结构和内容变化过大，匹配等法无法找到对应节点。为了解决这个问题，Scratch 提供了一种补偿性的离线快照回放机制，以提供一些提示性的信息给用户使其手动地定位到需要的信息。

在记录脚本时，Scratch 会自动地把整个页面快照下来，然后伴随着脚本文件一同被保存下来。因此当匹配失效时，这些快照的网页可以被用来展示用户之前的操作信息。利用 Chrome API chrome.pageCapture() 函数可以保存某个标签页中的网页，并且包装成 MHTML 文件的形式。为了解决 MHTML 文件自身的安全性隐患，Scratch 强制要求该文件只能从文件系统中读入，而且只能被载入到顶层的 frame 中。同理，Scratch 不允许直接打开一个在服务器上或者在 HTML5 本地存储沙盒上的 MHTML 网页。因此，在目前的实现中，Scratch 不通过 URL 链接打开该网页，而是生成一个下载的链接，让用户手动地下载之后保存在本地磁盘中再打开。这样一来，在打开后脚本就可以继续在标签中回放了。需要注意的是，从本地沙盒中“下载”并不需要网络链接，因为网页文件就保存在本地。这样，就可以在完全离线的状态下回放单个页面。

4.3 用户界面

图 7 显示了 Scratch 的用户界面。

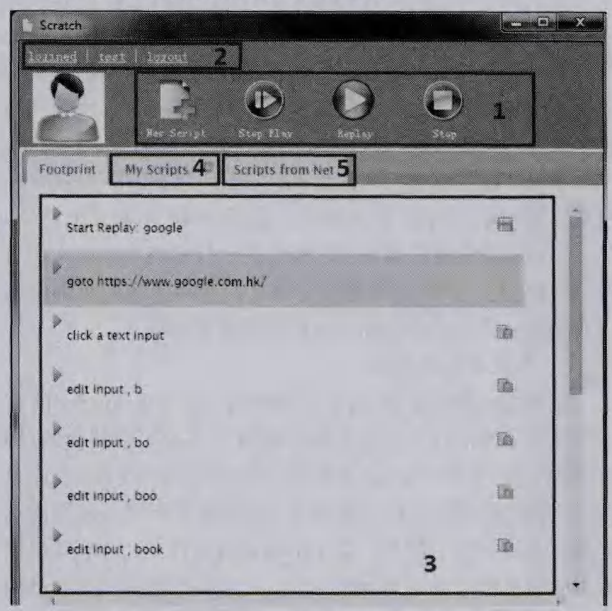


图 7 Scratch 用户界面

其中，框 1 中是 Scratch 的操作按钮，用来控制 Scratch 的记录回放功能。框 2 中是用户管理的信息，用户可以登录和查看自己发布的脚本以及别人分享的脚本等。框 3 中是 Scratch 的主要显示模块，用于展示信息和编辑脚本。框 4 是一个标签按钮，点击不同的标签会切换框 3 中展示的信息。其中可以展示用户自己创建的脚本或者从别人分享的脚本中下载到本地沙盒的脚本。框 5 也是一个标签按钮，点击后框内将显示所有用户创建的所有可用的脚本信息。用户还可以根据自己的需要进行脚本搜索或者编辑脚本。

5 Scratch 系统性能评估

Scratch 作为 Chrome 浏览器的插件形式存在，当用户上网开启 Scratch 记录其动作时，如果造成性能上较大的负载，那么将严重影响用户体验和工具的可用性。为了验证 Scratch 系统在运行时的性能，我们选取了百度（http://www.baidu.com）和雅虎（http://www.yahoo.com）两个网站作为测试对象。前者是表单型应用，后者是门户型应用，具

备较好的代表性。

由于用户进行每步操作时，浏览器都会和 Scratch 发生交互，因此用户在网页上的操作次数便成为我们需要考察的因素。本实验中，选取点击(click)这一典型的浏览器事件进行测试，分别对两个网站进行 20 次和 60 次点击后(百度的测试包含输入关键词搜索后对结果页的点击)，记录内存消耗情况和 CPU 线程数，如表 2 所列。

表 2　Scratch 性能测试

测试网站	是否开启 Scratch	CPU 线程数	内存	Js 使用内存
百度(点击 20 次)	不开启 Scratch	1.5	25400k	3635k
http://www.baidu.com	开启 Scratch	1.5	25980k	6400k
百度(点击 60 次)	不开启 Scratch	1.5	26500k	3028k
http://www.baidu.com	开启 Scratch	1.5	27500k	6500k
雅虎(点击 20 次)	不开启 Scratch	2	80000k	14521k
http://www.yahoo.com	开启 Scratch	2	82000k	26400k
雅虎(点击 60 次)	不开启 Scratch	2.5	74500k	16000k
http://www.yahoo.com	开启 Scratch	3	84000k	26700k

表 2 中，内存表示该 Web 应用进程(Chrome 中每个 Web 应用分配一个进程)所占用的全部内存；由于 Scratch 以 JavaScript 实现并在运行时注入到被测应用中，因此我们单独记录了 Js 所占用的内存(需要注意的是，这部分还包含网页自身的 Js 文件解析所需的内存)。可以发现，在百度的测试结果中，Scratch 开启与否对内存和 CPU 几乎没有影响，而随着点击次数上升，使用 Scratch 后 Js 使用内存则有小幅上升(最多也仅有 1M 左右)，这是因为 Scratch 会对每次操作记录。但总体上看，开启 Scratch 后对用户浏览网页的影响很小。

在性能测试过程中，我们也对网页变化性进行了简单实验。在百度和雅虎的所有 160 次操作中，每次回放时页面都会发生一定的变化，而 Scratch 都能够准确定位到变化后的节点(如果存在的话)，初步验证了本文所提节点匹配算法的有效性。

结束语　本文实现了一个基于 Chrome 浏览器的用户记录捕捉与回放工具，解决了网页动态结构与内容变化对捕捉回放技术带来的挑战，通过在线和离线两种模式保证回放的一致性和正确性。

目前 Scratch 在用户脚本上的编辑还不太完善，不便于用户共享和复用。未来将结合自然语言处理技术，提高记录脚本的可读性、可编辑性和可定制性。同时，在更广的范围内如 Web 应用自动化测试、基于 Web 的在线协同等尝试应用 Scratch。

参 考 文 献

[1] Eclipse. org. TPTP[OL]. http://www.eclipse.org/tptp

[2] Kojouharov C, Solodovnik A, Naumovich G. JTutor: an Eclipse plug-in suite for creation and replay of code-based tutorials[C]// Proceedings of the 2004 OOPSLA Workshop on Eclipse Technology Exchange(Eclipse'04). ACM, New York, NY, 2004: 27-31

[3] Leshed G, Haber E M, Matthews T, et al. CoScripter: Automating & Sharing How-To Knowledge in the Enterprise[C]// CHI. 2008

[4] Mickens J, Elson J, Howell J. Mugshot: Deterministic Capture and Replay for JavaScript Applications[C]//NSDI. 2010

[5] Lau T, Bergman L, Castelli V, et al. Sheepdog: Learning Procedures for Technical Support[C]//IUI. 2004: 109-116

[6] Little G, Lau T A, Cypher A, et al. Koala: Capture, Share, Automate, PersonalizeBusiness Processes on the Web[C]// ACM. 2007: 109-116

[7] Chrome Extension Structure [OL]. http://developer.chrome.com/extensions/getstarted.html

[8] ECMAScript Standard [OL]. http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf

(上接第 111 页)

[21] Robinson J T. The KDB Tree: A Search structure for large multidimensional dynamic indexes[C]// Proceedings of the 1981 ACM SIGMOD International Conference on Management of Data. Ann Arbor, 1981: 10-18

[22] Samet H. The quad tree and related hierarchical data structures [J]. ACM Computing Surveys, 1984, 16(2): 187-260

[23] Finkel R A, Bentley J L. Quad Trees a Data Structure for Retrieval on Composite Keys[J]. Acta Informatica, 1974, 4(1): 1-9

[24] 于戈，谷峪，鲍玉斌，等. 云计算环境下的大规模图数据处理技术[J]. 计算机学报，2011，34(10)：1753-1767

[25] 王结臣，张辉，吴文周，等. 一种平面散乱点集的自适应空间划分算法[J]. 武汉大学学报，2012，37(007)：770-774

[26] Yang B, Ma Q, Qian W, et al. TRUSTER: Trajectory data processing on clusters[C]// Proceeding of the 14th International Conference on Database Systems for Advanced Applications. Brisbane, 2009: 768-771

[27] Ma Q, Yang B, Qian W, et al. Query processing of massive trajectory database on MapReduce[C]// International Conference on Information and Knowledge Management. Hong Kong, China, 2009: 9-16

[28] Ye Xiang-long, Huang Meng-xing, Zhu Dong-hai, et al. A novel blocks placement strategy for Hadoop [C]// Proceedings of the 11th International Conference on Computer and Information Science. Washington D C: IEEE, 2012: 3-7

[29] 林伟伟，刘波. 基于动态带宽分配的 Hadoop 数据负载均衡方法[J]. 华南理工大学学报：自然科学版，2012，40(9)：42-47

[30] 刘琨，肖琳，赵海燕. Hadoop 中云数据负载均衡算法的研究及优化[J]. 微电子学与计算机，2012，29(9)：18-22

[31] Borthakur D. The Hadoop Distributed File System: Architecture and Design[EB/OL]. [2012-01-03]. http://hadoop.Apache.org/common/docs/stable/hdfs_design.html

[32] Apache Hadoop. Rebalance Data Blocks [EB/OL]. http://issues.Apache.org/HADOOP1652, 2009

[33] White T. Hadoop: The Definitive Guide [M]. Beijing: Tsinghua University Press, 2010

[34] GitHub Inc. Building Footprints in Chicago [EB/OL]. https://github.com/Chicago/osd-building-footprints, 2013