



(12)发明专利

(10)授权公告号 CN 103927253 B

(45)授权公告日 2017.04.19

(21)申请号 201310012020.6

(22)申请日 2013.01.11

(65)同一申请的已公布的文献号

申请公布号 CN 103927253 A

(43)申请公布日 2014.07.16

(73)专利权人 阿里巴巴集团控股有限公司

地址 英属开曼群岛大开曼资本大厦一座四层847号邮箱

(72)发明人 金茜茜

(74)专利代理机构 北京润泽恒知识产权代理有限公司 11319

代理人 苏培华

(51)Int.Cl.

G06F 11/36(2006.01)

(56)对比文件

CN 102546273 A, 2012.07.04,

EP 2487595 A2, 2012.08.15,

CN 102033803 A, 2011.04.27,

审查员 陈沙沙

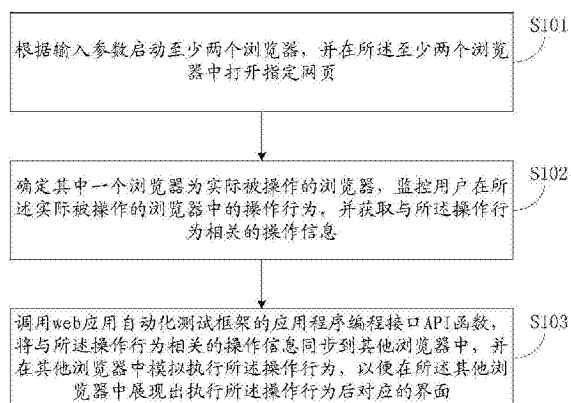
权利要求书2页 说明书13页 附图4页

(54)发明名称

多浏览器兼容性测试方法及系统

(57)摘要

本申请公开了多浏览器兼容性测试方法及系统,其中,所述方法包括:根据输入参数启动至少两个浏览器,并在所述至少两个浏览器中打开指定网页;确定其中一个浏览器为实际被操作的浏览器,监控用户在所述实际被操作的浏览器中的操作行为,并获取与所述操作行为相关的操作信息;调用web应用自动化测试框架的应用程序编程接口API函数,将与所述操作行为相关的操作信息同步到其他浏览器中,并在其他浏览器中模拟执行所述操作行为,以便在所述其他浏览器中展现出执行所述操作行为后对应的界面。通过本申请,能够有效地提升测试效率,并且能够直观、有效的监控到各浏览器页面的兼容性。



1. 一种多浏览器兼容性测试方法,其特征在于,包括:

根据输入参数启动至少两个浏览器,并在所述至少两个浏览器中打开指定网页;

确定其中一个浏览器为实际被操作的浏览器,监控用户在所述实际被操作的浏览器中的操作行为,并获取与所述操作行为相关的操作信息;

调用web应用自动化测试框架的应用程序编程接口API函数,将与所述操作行为相关的操作信息同步到其他浏览器中,并在其他浏览器中模拟执行所述操作行为,以便在所述其他浏览器中展现出执行所述操作行为后对应的界面;

其中,所述调用web应用自动化测试框架的API函数包括:

创建服务端;

通过Socket通信机制,将与所述操作行为相关的操作信息发送到所述服务端;

由所述服务端调用web应用自动化测试框架的API函数;

或,所述调用web应用自动化测试框架的API函数包括:

在所述实际被操作的浏览器的当前页面中创建一隐藏元素;

将与所述操作行为相关的操作信息作为所述隐藏元素的属性信息进行记录;

通过读取所述隐藏元素的属性值获取与所述操作行为相关的操作信息,并调用web应用自动化测试框架的API函数。

2. 根据权利要求1所述的方法,其特征在于,所述通过Socket通信机制,将与所述操作行为相关的操作信息发送到所述服务端包括:

在每次获取到与所述操作行为相关的操作信息之后,为待建立连接分配端口号,并创建XMLHttpRequest对象,通过所述XMLHttpRequest对象向所述服务端发送连接请求;

所述服务端基于所述端口号为每次收到的连接请求创建Socket实例,建立起相应端口上的连接,通过所述连接接收与所述操作行为相关的操作信息。

3. 根据权利要求2所述的方法,其特征在于,还包括:

在每次接收完所述操作信息之后,将针对此次连接请求创建的Socket实例关闭。

4. 根据权利要求1所述的方法,其特征在于,还包括:

在接收到结束此次测试的指示后,将所述创建的服务端关闭。

5. 根据权利要求1所述的方法,其特征在于,所述监控用户在所述实际被操作的浏览器中的操作行为包括:

通过注册钩子函数,监听用户在所述实际被操作的浏览器中的鼠标及键盘操作;

当监控到鼠标操作时,获取被操作页面元素的类型及路径,并在所述页面元素为输入框类型时,获取输入到输入框中的内容;

当监控到键盘操作时,获取被操作按键对应的keycode;

所述将与所述操作行为相关的操作信息同步到其他浏览器中包括:

将监控到的操作类型、被操作页面元素的类型及路径、输入到输入框中的内容、被操作按键对应的keycode同步到其他浏览器中。

6. 一种多浏览器兼容性测试系统,其特征在于,包括:

启动单元,用于根据输入参数启动至少两个浏览器,并在所述至少两个浏览器中打开指定网页;

监控单元,用于确定其中一个浏览器为实际被操作的浏览器,监控用户在所述实际被

操作的浏览器中的操作行为,并获取与所述操作行为相关的操作信息;

调用单元,用于调用web应用自动化测试框架的API函数,将与所述操作行为相关的操作信息同步到其他浏览器中,并在其他浏览器中模拟执行所述操作行为,以便在所述其他浏览器中展现出执行所述操作行为后对应的界面;

其中,所述调用单元包括:

服务端创建子单元,用于创建服务端;

通信子单元,用于通过Socket通信机制,将与所述操作行为相关的操作信息发送到所述服务端;

间接调用子单元,用于由所述服务端调用web应用自动化测试框架的API函数;

或,所述调用单元包括:

隐藏元素创建子单元,用于在所述实际被操作的浏览器的当前页面中创建一隐藏元素;

属性信息记录子单元,用于将与所述操作行为相关的操作信息作为所述隐藏元素的属性信息进行记录;

直接调用子单元,用于通过读取所述隐藏元素的属性值获取与所述操作行为相关的操作信息,并调用web应用自动化测试框架的API函数。

多浏览器兼容性测试方法及系统

技术领域

[0001] 本申请涉及浏览器兼容性测试技术领域,特别是涉及多浏览器兼容性测试方法及系统。

背景技术

[0002] 浏览器是互联网产品客户端的核心软件,也是网站访问的必备软件。不同厂商的浏览器对Java、JS、ActiveX、plugin-ins的支持也各有差异。即使是同一厂家的浏览器,也会存在不同的版本的问题。例如,ActiveX是Microsoft的产品,是为Internet Explore (IE) 而设计的;JS是Netscape的产品等。另外,框架和层次结构风格在不同的浏览器中也有不同的显示。目前最为常用的浏览器为IE,但由于操作习惯等问题,还有相当一部分用户喜欢Firefox浏览器、谷歌浏览器等等。

[0003] 针对这种情况,网页或者web服务开发商也应制定有针对性的测试策略来进行网站的浏览器兼容测试,从而保证网站在各种不同的浏览器中有能得到良好的展现。对一些特殊项目,可以指定某一类型的浏览器(包括版本),这些都必须需求规格说明书指明,针对这些指明的浏览器必须进行兼容性测试。但是,考虑到大部分项目是不能指定浏览器的,针对这样的项目,就需要对各种浏览器(含版本)的兼容性进行测试,尽量保证测试的完整性,最好是能够兼容多种内核的浏览器(比如IE内核、谷歌浏览器内核、火狐浏览器内核等)。

[0004] 在浏览器的兼容性测试中,为了测试兼容性,一般需要测试人员手工在各个浏览器中分别、重复的进行相同操作,既效率低下,又让测试人员感到枯燥。自动化测试可以让测试人员脱离枯燥的测试行为,提升测试效率。

[0005] 现有技术中出现了一些自动化测试的工具,但是大都存在一些缺陷。例如,有的测试工具提供可以快速在各种浏览器中切换的方案,但这种方式只是减少了测试人员为了安装这些浏览器而需要做机器、操作系统等资源的准备,对于重复的操作行为没有改善,并且只能在IE浏览器下进行测试,并没有完全达到兼容性测试的需求。还有的自动化测试工具能够提供页面在不同浏览器中的截图,由测试人员对这些截图进行验证,通过页面截图,只能看到各浏览器中的静态页面展示效果,而无法验证对页面的操作行为,等等。

发明内容

[0006] 本申请提供了多浏览器兼容性测试方法及系统,能够有效地提升测试效率,并且能够直观、有效的监控到各浏览器页面的兼容性。

[0007] 本申请提供了如下方案:

[0008] 一种多浏览器兼容性测试方法,包括:

[0009] 根据输入参数启动至少两个浏览器,并在所述至少两个浏览器中打开指定网页;

[0010] 确定其中一个浏览器为实际被操作的浏览器,监控用户在所述实际被操作的浏览器中的操作行为,并获取与所述操作行为相关的操作信息;

[0011] 调用web应用自动化测试框架的应用程序编程接口API函数,将与所述操作行为相关的操作信息同步到其他浏览器中,并在其他浏览器中模拟执行所述操作行为,以便在所述其他浏览器中展现出执行所述操作行为后对应的界面。

[0012] 本申请还提供一种多浏览器兼容性测试系统,包括:

[0013] 启动单元,用于根据输入参数启动至少两个浏览器,并在所述至少两个浏览器中打开指定网页;

[0014] 监控单元,用于确定其中一个浏览器为实际被操作的浏览器,监控用户在所述实际被操作的浏览器中的操作行为,并获取与所述操作行为相关的操作信息;

[0015] 调用单元,用于调用web应用自动化测试框架的API函数,将与所述操作行为相关的操作信息同步到其他浏览器中,并在其他浏览器中模拟执行所述操作行为,以便在所述其他浏览器中展现出执行所述操作行为后对应的界面。

[0016] 根据本申请提供的具体实施例,本申请公开了以下技术效果:

[0017] 通过本申请,能够捕获测试人员在某浏览器中的操作行为,并通过调用浏览器自动化测试框架的API函数,将操作行为相关的操作信息同步发送到其他浏览器中,并在其他浏览器中模拟操作行为的执行。也即,将测试人员在某一浏览器的操作行为,即时同步到其它浏览器中,这样测试人员只需要在一种浏览器中进行操作,就能同时看到其它浏览器中进行相同操作的效果。这样既避免了测试人员在不同浏览器中分别进行相同操作的枯燥,同时有效地提升测试效率,并且能够直观、有效的监控到各浏览器页面的兼容性。

[0018] 其中,可以通过Socket通信机制,将捕获到的操作信息发送到Server中,并由Server通过调用Selenium-WebDriver扩展或Waitr扩展的API函数将操作信息同步到其他浏览器,并在其他浏览器中模拟执行相应的操作行为。在这种方式下,每监控到一个操作行为,都可以通过建立连接的方式发送给Server端,并由Server同步到其他浏览器,可以保证信息同步的可靠性。

[0019] 或者,也可以在实际被操作的浏览器的当前页面中插入隐藏元素,将捕获到的操作信息以该元素的属性的形式进行记录,然后就可以通过读取该隐藏元素的属性值来取出捕获到的操作信息,进而再通过调用Selenium-WebDriver扩展或Waitr扩展的API函数将操作信息同步到其他浏览器,并在其他浏览器中模拟执行相应的操作行为。这种方式在实现过程中不需要借助于Server端,可以缩短信息的传送路径。

[0020] 当然,实施本申请的任一产品并不一定需要同时达到以上所述的所有优点。

附图说明

[0021] 为了更清楚地说明本申请实施例或现有技术中的技术方案,下面将对实施例中所需要使用的附图作简单地介绍,显而易见地,下面描述中的附图仅仅是本申请的一些实施例,对于本领域普通技术人员来讲,在不付出创造性劳动的前提下,还可以根据这些附图获得其他的附图。

[0022] 图1是本申请实施例提供的方法的流程图;

[0023] 图2是本申请实施例提供的一方法中的系统框架示意图;

[0024] 图3是本申请实施例提供的另一方法的流程图;

[0025] 图4是本申请实施例提供的系统的示意图;

[0026] 图5是本申请实施例提供的另一系统的示意图；

[0027] 图6是本申请实施例提供的再一系统的示意图。

具体实施方式

[0028] 下面将结合本申请实施例中的附图，对本申请实施例中的技术方案进行清楚、完整地描述，显然，所描述的实施例仅仅是本申请一部分实施例，而不是全部的实施例。基于本申请中的实施例，本领域普通技术人员所获得的所有其他实施例，都属于本申请保护的范围。

[0029] 本申请实施例中，可以启动多个浏览器，并在各个浏览器中打开某指定网页之后，可以选择其中一个浏览器作为实际操作的浏览器，测试程序会向该浏览器的页面中注入JS (JavaScript) 脚本，该脚本可以有多种功能，包括在测试人员在该实际被操作的浏览器中执行操作（例如，点击某链接、点击某按钮、在某输入框中输入等等）时，对测试人员的操作行为进行监控，捕获相关的操作信息，等等。然后，可以将捕获到的操作信息同步发送到其他浏览器，并在其他浏览器中模拟执行这些操作信息对应的操作行为，以便在其他浏览器中展现出执行操作行为后对应的界面。也即，将测试人员在某一浏览器的操作行为，即时同步到其他浏览器中，这样测试人员只需要在一种浏览器中进行操作，就能同时看到其他浏览器中进行相同操作的效果。这样既避免了测试人员在不同浏览器中分别进行相同操作的枯燥，同时有效地提升测试效率，并且能够直观、有效的监控到各浏览器页面的兼容性。下面就对更为具体的实现过程进行详细地介绍。

[0030] 参见图1，本申请实施例提供的多浏览器兼容性测试方法可以包括以下步骤：

[0031] S101：根据输入参数启动至少两个浏览器，并在所述至少两个浏览器中打开指定网页；

[0032] 在本申请实施例中，可以在测试程序的界面中，为测试人员提供输入参数的入口，测试人员可以按照预定的格式输入参数，该输入参数需要包括其要启动的浏览器、要打开的网页等方面的信息，例如，可以输入：`sync(urlA,ff,ie,chrome)`，也就是说，需要启动Firefox(ff)、IE和Chrome浏览器，并在其中打开网址为urlA的网页。测试程序在接收到该输入参数之后，就可以在用户界面中启动Firefox、IE和Chrome浏览器，并在分别在各个浏览器中打开网址为urlA的网页。

[0033] S102：确定其中一个浏览器为实际被操作的浏览器，监控用户在所述实际被操作的浏览器中的操作行为，并获取与所述操作行为相关的操作信息；

[0034] 实际被操作的浏览器也就是测试人员实际在其中执行操作行为的浏览器。具体实现时，可以有多种方式来确定哪个浏览器是实际被操作的浏览器。例如，可以是预先设定的某固定的浏览器（例如，固定将Firefox作为实际被操作的浏览器），或者，也可以是将输入参数中的第一个浏览器确定为实际被操作的浏览器，等等。

[0035] 在确定了实际被操作的浏览器之后，就可以向其中打开的网页页面中注入JS脚本，该JS脚本可以执行多种操作，其中包括对用户操作行为的监控，对与用户操作行为相关的操作信息的获取等等。

[0036] S103：调用浏览器自动化测试框架的API (Application ProgrammingInterface，应用程序编程接口) 函数，将与所述操作行为相关的操作信息同步到其他浏览器中，并在其

他浏览器中模拟执行所述操作行为,以便在所述其他浏览器中展现出执行所述操作行为后对应的界面。

[0037] 在当前实际被操作的浏览器中获取到操作行为相关的操作信息之后,就可以调用浏览器自动化测试框架(例如,Selenium-WebDriver扩展或者Waitr扩展等)的API函数,将与操作行为相关的操作信息同步到其他浏览器中,并在其他浏览器中模拟执行这种操作行为,这样就可以在其他浏览器中展现出执行这种操作行为后对应的界面。

[0038] 具体实现时,为了能够将在当前被实际操作的浏览器中捕获到的操作信息同步到其他浏览器中,可以有多种实现方式,下面分别进行介绍。

[0039] 实施例一

[0040] 在该实施例一中,可以创建服务端,在当前被实际操作的浏览器与已启动的其他浏览器之间建立起通信通道,然后通过Socket通信机制,将与所述操作行为相关的操作信息发送到所述服务端,再由服务端调用浏览器自动化测试框架的API函数,将与操作行为相关的操作信息同步到其他浏览器中,并在其他浏览器中模拟执行操作行为。

[0041] 具体实现时,整个框架可以分为Server端和Client端。参见图2,具体的实现过程如下:首先,对实际被操作浏览器中打开的网页页面document中注入js脚本,为此次连接分配端口号,创建XMLHttpRequest对象XHR,并注册钩子函数,以绑定鼠标事件和键盘事件、触屏事件等输入事件,从而对测试人员在该浏览器中的操作行为进行监控和捕获;然后,通过TCP Server类创建Server,并通过XHR实现从实际被操作的浏览器(可以看作是Client端)与Server(可以看作是Server端)之间一对一的消息交互,将向Server端发送连接请求。Server在接收到连接请求时,通过创建Socket实例建立起相应端口上的TCP连接,以便通过该TCP连接获取到发送过来的操作信息。最后,再由Server将操作信息发送给其他浏览器;其中,在将操作信息发送给其他浏览器时,可以通过调用Selenium-WebDriver扩展或者Waitr扩展中的API函数的方式,将操作信息发送到已启动的其他浏览器,并调用Selenium-WebDriver扩展或者Waitr扩展的API函数在其他浏览器中执行元素的定位、操作的模拟执行等操作,并显示相应的操作结果,以此来达到不同浏览器上的兼容性测试的目的。当然,在实际应用中,除了在当前实际被操作的浏览器与Server之间创建TCP连接之外,也可以创建UDP连接等其他连接,在本申请实施例中仅基于TCP连接进行介绍,但不限于TCP连接,关于UDP连接等其他连接这里不再详述。

[0042] 可见,在该实施例一中,涉及到多个技术关键点,包括JS脚本的注入、端口号的分配、XMLHttpRequest对象、Server、Socket实例、Selenium-WebDriver扩展,下面分别进行介绍。

[0043] 第一,关于JS脚本的注入

[0044] 为了实现本申请实施例中的兼容性测试方案,首先可在实际被操作的浏览器的当前页面中注入JS脚本(只需要在当前实际被操作的浏览器中注入,其他浏览器中不需要注入),该JS脚本需要进行的操作有很多。包括端口号的分配、对测试人员操作行为的监控及捕获、XMLHttpRequest对象的创建、Server端的创建等。在当前实际被操作的浏览器中打开新的页面之后,还需要在新的页面中重新注入该JS脚本。

[0045] 其中,对用户的操作行为进行监控时,主要包括用户的鼠标操作行为、键盘操作行为以及触屏操作行为等,其中,鼠标操作行为包括点击鼠标的行为(还可以细分为左键单

击、左键双击、右键单击等等)、鼠标移入/移出某控件按钮所在区域、鼠标悬停在某控件按钮上等等;键盘操作主要是指通过按键进行字符的输入、标签的切换等等;触屏操作行为主要是点击、滑动、缩放等等。由于对相关的系统函数注册了钩子函数,因此,当发生这种行为时,测试程序就可以截获到,并从中获取到操作对象、操作类型等信息。

[0046] 对于不同的操作行为,需要获取的操作信息也会略有不同。鼠标操作和触屏操作行为类似,在此仅以鼠标操作为例说明。例如,鼠标点击某页面元素时,就可以获取到当前的操作类型是鼠标左键单击,操作对象是页面元素A,还可以获知该页面元素所在的路径等信息,然后就可以通过XMLHttpRequest将这些信息发送给Server,进而Server就可以在创建Socket实例之后,读取到这些信息,并通过调用Selenium-WebDriver或者Waitr的API函数,将这些信息发送给已启动的其他浏览器,并根据元素的路径定位到页面元素,然后触发对该页面元素执行鼠标左键单击的操作,这样,就可以在对应的浏览器中显示出操作之后的显示效果。当然,除了根据元素的路径进行元素的定位之外,Selenium-WebDriver或Waitr扩展还可以根据元素的id、名称、CSS属性等来进行页面元素的定位,因此,除了获取页面元素的路径信息之外,还可以获取页面元素的id、名称、CSS属性等信息。

[0047] 其中,如果发现操作行为对应的操作对象为输入框,则除了需要获取该输入框的路径等用于进行元素定位的信息之外,还可以获取输入到该输入框中的内容,进而将输入框的路径、输入到该输入框的内容一起通过XMLHttpRequest将这些信息发送给Server,进而Server就可以在创建Socket实例之后,获取到这些信息,并通过Selenium-WebDriver扩展或Waitr扩展将这些信息发送给已启动的其他浏览器。其他浏览器中安装的Selenium-WebDriver或Waitr扩展就可以根据输入框的路径定位到该输入框,对利用捕获到的输入框中的输入内容对该输入框进行赋值。具体在获取输入框中的内容时,可以通过触发a_blur函数来获取,或者,由于测试人员在一个输入框中完成输入之后,可能会通过按下tab键的方式来将输入焦点切换到下一输入框,因此,也可以通过监控tab键的操作,来获取输入框中的输入内容。

[0048] 如果发现操作行为为按下键盘操作的时候,则可以获取被按下按键的keycode,然后通过XMLHttpRequest,传送给Server端,再由Server在创建Socket实例之后,获取到这些信息,并通过Selenium-WebDriver扩展或Waitr扩展将这些信息发送给已启动的其他浏览器,以便其他浏览器中可以根据产生的keycode序列获知通过键盘输入的具体内容。

[0049] 第二,关于端口号的分配

[0050] 由于在该实施例,需要基于TCP连接的通信机制将操作信息从当前被操作的浏览器传送给Server端,因此,在当前实际被操作的浏览器与Server之间建立起一次TCP连接之前,需要为此次连接分配端口号,以便后续基于该端口号创建Socket实例,建立起TCP连接。

[0051] 其中,在分配端口号时,可以是预先设定好一个端口号,每次需要创建时,都是为TCP连接分配该固定的端口号(也即为每次连接分配的端口号都是相同的),这样,在Server端创建Socket实例时,只要基于该固定的端口号进行创建即可。但是,考虑到固定的端口号可能会与其他应用产生冲突,因此,在实际应用中,也可以采用随机分配的形式。也即,在需要分配端口号时,随机生成四位数字作为此次连接的端口号。当然,在这种情况下,就需要以某种形式将该端口号通知到Server(因为Server需要基于为当前连接分配的端口号创建

Socket实例)。为此,具体实现时,可以在随机分配了端口号之后,可以在当前实际被操作的浏览器的页面中创建一个隐藏元素,并将当前分配的端口号作为该隐藏元素的一个属性进行记录;这样Server在创建TCP连接是,就可以根据该隐藏元素的ID(在创建隐藏元素时,可以为该元素设置固定的ID,因此,对于Server而言,该ID可以是预先获知的)查找到该元素,并读取该元素的属性,从中获取到当前的端口号,然后依据该端口号创建Socket实例,建立TCP连接即可。

[0052] 第三,关于Server端

[0053] Server端是在当前实际被操作的浏览器与已启动的其他浏览器之间建立起通信通道的过程中所需使用的通信终端,其功能是等待client端的连接,反复接收连接请求,向客户端发送消息并接收反馈字节。具体实现时,可以是捕获到用户操作行为的操作信息之后,由注入到当前被操作浏览器的页面中的JS脚本,使用TCP Server类创建一个Server端,然后将捕获到的操作信息通过XHR对象发送到该Server端,Server端就可以通过创建Socket实例读取到发送过来的具体操作信息。

[0054] 需要说明的是,只需要在一次连接建立时创建一次Server端,然后在此次连接结束之后,再将该Server关闭,下次建立连接时再创建新的Server。其中,可以通过设置快捷键等方式,作为结束此次连接的标识。例如,测试人员在完成一次测试之后,可以通过按下F9等快捷键的方式通知测试程序此次测试结束,测试程序在接收到该指令之后,就可以结束该Server。

[0055] 第四,关于XMLHttpRequest对象

[0056] XMLHttpRequest允许一个网站连接到远程服务器获取结构化的数据。与建立一系列静态页面不同,XMLHttpRequest能使开发者创建高度动态的应用程序。在创建XMLHttpRequest对象XHR后,可以调用一个初始化方法open(),调用open()方法并不会真正发送请求,而只是启动一个请求准备发送。要真正发送请求要使用send()方法,send()方法接收一个参数,即要作为请求主体发送的数据,如果不需要通过请求主体发送数据,必须传递一个null值。在调用send()之后,请求就会被分派到Server端。

[0057] 具体到本申请实施例,就是Client端(即当前实际被操作的浏览器)使用XmlHttpRequest对象通过端口号的方式连接到指定的服务端,接收和发送消息。被同步的浏览器从服务器接收数据,由Selenium-WebSriver扩展开发分发消息到各个浏览器,并驱动浏览器执行消息。过程如下:

[0058] 1) 通过XmlHttpRequest类的open()和send()方法进行与服务端的通信;

[0059] 2) 使用XmlHttpRequest类的abort()方法关闭连接。

[0060] 3) 每次操作作为一个轮回,迭代建立XmlHttpRequest对象去连接服务器,等待接收服务器发送的下一个命令处理请求。从而实现了服务端和客户端消息的循环发送。

[0061] 第五,关于Socket实例

[0062] Server在接收到用户操作行为相关的操作信息之后,需要针对当前请求创建Socket实例,才能获取到发送过来的操作信息的具体内容。其中,应用层通过传输层进行数据通信时,TCP和UDP会遇到同时为多个应用程序进程提供并发服务的问题。多个TCP连接或多个应用程序进程可能需要通过同一个TCP协议端口传输数据。为了区别不同的应用程序进程和连接,许多计算机操作系统为应用程序与TCP/IP协议交互提供了称为套接字

(Socket)的接口,区分不同应用程序进程间的网络通信和连接。

[0063] 生成套接字,主要有三个参数:通信的目的IP地址、使用的传输层协议(TCP或UDP)和使用的端口号。通过把这三个参数结合起来,与一个Socket绑定,应用层就可以和传输层通过套接字接口,区分来自不同应用程序进程或网络连接的通信,实现数据传输的并发服务。

[0064] Socke可以看成在两个程序进行通讯连接中的一个端点,一个程序将一段信息写入Socket中,该Socket将这段信息发送给另外一个Socket中,使这段信息能传送到其他程序中。例如,Host A上的程序A将一段信息写入Socket中,Socket的内容被Host A的网络管理软件访问,并将这段信息通过Host A的网络接口卡发送到Host B,Host B的网络接口卡接收到这段信息后,传送给Host B的网络管理软件,网络管理软件将这段信息保存在Host B的Socket中,然后程序B才能在Socket中阅读这段信息。

[0065] 要通过互联网进行通信,可通过一对套接字,一个运行于客户机端,称之为ClientSocket,另一个运行于服务器端,称之为serverSocket。根据连接启动的方式以及本地套接字要连接的目标,套接字之间的连接过程可以分为三个步骤:服务器监听、客户端请求、连接确认。

[0066] 服务器监听:是服务器端套接字并不定位具体的客户端套接字,而是处于等待连接的状态,实时监控网络状态。

[0067] 客户端请求:是指由客户端的套接字提出连接请求,要连接的目标是服务器端的套接字。为此,客户端的套接字必须首先描述它要连接的服务器的套接字,指出服务器端套接字的地址和端口号,然后就向服务器端套接字提出连接请求。

[0068] 连接确认:是指当服务器端套接字监听到或者说接收到客户端套接字的连接请求,它就响应客户端套接字的请求,建立一个新的线程,把服务器端套接字的描述发给客户端,一旦客户端确认了此描述,连接就建立好了。而服务器端套接字继续处于监听状态,继续接收其他客户端套接字的连接请求

[0069] 也就是说,设置一个简单的TCP服务器涉及下列步骤:

[0070] 1) 调用socket()函数建立套接字;

[0071] 2) 调用bind()函数把套接字绑定到一个监听端口上;

[0072] 3) 调用listen()函数,使该套接字成为一个处在监听状态的套接字;

[0073] 4) 接下来,服务器可以通过accept()函数接受客户端的连接请求。若没有收到连接请求,accept()函数将不会返回并阻塞程序的执行。接收到连接请求后,accept()函数会为该连接返回一个套接字描述符。accept()函数可以被多次调用来接收不同客户端的连接请求,而且之前的连接仍处于监听状态——直到其被关闭为止。

[0074] 5) 现在,服务器可以通过对send(),recv()或者对write(),read()等函数的调用来同客户端进行通信。

[0075] 6) 对于一个不再需要的套接字,可以使用close()函数关闭它。

[0076] 需要说明的是,在本申请实施例,在建立一次连接时,需要创建一个Server端,但在一次连接的过程中,会截获到多个操作行为,每截获到一个操作行为,就可以发起一次数据发送请求,因此,Server端针对每次请求都需要创建一个Socket实例,此次请求完成之后,则将该Socket实例关闭,下次再接收到新的请求时,再重新创建Socket实例,以此循环

往复。也就是说,在具体实现是,在一次连接中只需要创建一次Server端,但是需要多次创建Socket实例,每捕获到一个新的操作行为信息,都会发起一个新的请求,并通过一个新创建的Socket实例获取到相应的信息,然后再同步给其他的浏览器,这样使得实现的可靠性得到保证。

[0077] 第六,关于Selenium-WebDriver扩展以及Waitr扩展

[0078] 为了便于理解,下面首先分别对Selenium-WebDriver扩展以及Waitr扩展本身进行简单地介绍。

[0079] Selenium是一个浏览器自动化操作框架。Selenium主要由三种工具组成,Selenium WebDriver作为这三种工具的其中一种,提供了各种语言环境的API来支持多种控制权和编写符合标准软件开发实践的应用程序,在项目内部,它也被称为“WebDriver”。

[0080] WebDriver是为web应用的自动化测试提供的一个框架,它针对各个浏览器而开发,使用每个浏览器自身对自动化的支持来直接调用浏览器,取代了嵌入到被测Web应用中的JavaScript。与浏览器的紧密集成支持创建更高级的测试,避免了JavaScript安全模型导致的限制。除了来自浏览器厂商的支持,WebDriver还利用操作系统级的调用模拟用户输入。WebDriver支持Firefox (FirefoxDriver)、IE (InternetExplorerDriver)、Opera (OperaDriver)、Chrome (ChromeDriver) 和Safari (SafariDriver)。它还支持Android (AndroidDriver) 和iPhone (IPhoneDriver) 的移动应用测试。它还包括一个基于HtmlUnit的无界面实现,称为HtmlUnitDriver。WebDriver API可以通过Python、Ruby、Java和C#访问,支持开发人员使用他们偏爱的编程语言来创建测试。

[0081] 为了能够使用WebDriver,则首先需要将WebDriver的JAR包加入到测试项目的CLASSPATH中,如果是使用maven构建的项目,只需要在pom.xml文件中加入特定的依赖项即可,然后,就可以使用它了。WebDriver的API遵从“BestFit (最适合)”原则,在保持良好的用户体验性和灵活性之间找到一个最佳的平衡点。

[0082] 其中,针对不同的浏览器,在安装WebDriver扩展时也需要安装与该浏览器对应的WebDriver,安装的方式也可能略有不同。例如,如果需要使用Firefox浏览器,则只需要设置WebDriver driver=new FirefoxDriver(),前提是Firefox被安装在默认的位置。如果FireFox没有被安装在指定的位置,可以设置“webdriver.firefox.bin”环境变量的值来指定它的位置。

[0083] 如果要使用Chrome浏览器,则需要首先下载一个ChromeDriver。这个程序是由Chrome团队提供的,可以看作它是连接WebDriver和Chrome浏览器的桥梁。然后启动ChromeDriver,会得到一个Url及监听端口。然后使用webDriver=newRemoteWebDriver(url,DesiredCapabilities.chrome())创建一个ChromeWebDriver进行操作。当然也可以在一个子线程中启动ChromeDriver,并设置给WebDriver。

[0084] WebDriver是W3C的一个标准,由Selenium主持,从这个协议中可以看到,WebDriver之所以能够实现与浏览器进行交互,是因为浏览器实现了这些协议。这个协议是使用JSON(JavaScript Object Notation)通过HTTP(Hypertext transfer protocol,超文本传输协议)进行传输。它的实现使用了经典的Client-Server模式。客户端发送一个请求(request),服务器端返回一个应答(response)。Firefox浏览器直接实现了WebDriver的通讯协议,而Chrome和IE则是通过ChromeDriver和InternetExplorerDriver实现的。为了更

好地理解,下面首先明确几个概念:

[0085] Client:调用WebDriverAPI的机器;

[0086] Server:运行浏览器的机器;

[0087] Session:服务器端需要维护浏览器的Session,从客户端发过来的请求头中包含了Session信息,服务器端将会执行对应的浏览器页面。

[0088] WebElement:这是WebDriverAPI中的对象,代表页面上的一个DOM元素。

[0089] 举个实际的例子,下面代码的作用是“命令”firefox转跳到aaa主页:

[0090] `WebDriver driver=new FirefoxDriver();`//实例化一个Driver

[0091] `driver.get("http://www.aaa.com");`

[0092] 在执行`driver.get("http://www.aaa.com")`这句代码时,也就是测试代码向remote server发送了如下的请求:

[0093] `POSTsession/285b12e4-2b8a-4fe6-90e1-c35cba245956/url post_data {"url":http://aaa.com}`

[0094] 通过post的方式请求localhost:port/hub/session/session_id/url地址,请求浏览器完成跳转url的操作。如果上述请求是可接受的,或者说remoteserver是实现了这个接口,那么remoteserver会跳转到该post data包含的url,并返回如下的response:

[0095] `{"name":"get","sessionId":"285b12e4-2b8a-4fe6-90e1-c35cba245956","status":0,"value":""}`

[0096] 该response中包含如下信息:

[0097] name:remote server端的实现的方法的名称,这里是get,表示跳转到指定url;

[0098] sessionId:当前session的id;

[0099] status:请求执行的状态码,非0表示未正确执行,0表示执行正确;

[0100] value:请求的返回值,这里返回值为空,如果client调用title接口,则该值应该是当前页面的title;

[0101] 如果client发送的请求是定位某个特定的页面元素,则response的返回值可能是这样的:

[0102] `{"name":"findElement","sessionId":"285b12e4-2b8a-4fe6-90e1-c35cba245956","status":0,"value":{"ELEMENT":{"2192893e-f260-44c4-bdf6-7aad3c919739"}}`

[0103] Name、sessionId、status跟上面的例子类似,区别是该请求的返回值是ELEMENT:{2192893e-f260-44c4-bdf6-7aad3c919739},表示定位到元素的id,通过该id,client可以发送如click之类的请求与server端进行交互。

[0104] 需要说明的是,在本申请实施例,可以预先对Selenium-WebDriver扩展提供的多种API函数进行封装,生成多种调用接口,测试程序只需要调用这些接口就可以实现信息的发送、元素的定位等等操作。并且,由于测试程序封装了Selenium-WebDriver扩展的API接口,因此,只要安装该测试程序,就可以自动实现对各种浏览器的支持,不需要测试人员再分别执行为各种浏览器安装插件等操作。

[0105] 另外需要说明的是,在将操作行为相关的操作信息发送到其他安装有Selenium-WebDriver扩展的浏览器之后,Selenium-WebDriver扩展就可以驱动对应的浏览器执行对

应的操作,进而各浏览器中展现出操作后的效果。例如,假设测试人员输入参数sync(url,ff,ie,chrome)时,界面上会开启装有Selenium-WebDriver扩展开发的Firefox、IE和Chrome浏览器,其中,实际被测试人员操作的是Firefox浏览器,因此,需要将测试人员在Firefox浏览器中的操作同步到IE和Chrome浏览器中。其中,当被操作对象为input输入框之外的其他页面元素,操作类型为鼠标左键单击(click)时,可以在IE和Chrome浏览器中就可以根据从服务器端获取到的元素路径path,通过Selenium-WebDriver扩展开发定位到该元素,并对其进行click操作;当被操作的对象为“input”输入框时,通过Selenium-WebDriver扩展在IE和Chrome浏览器查找到输入框后,可以对该输入框其进行set(value)操作;当被点击元素为链接(link)类型时,点击操作引起页面跳转,在将该操作同步到IE和Chrome浏览器的同时,还需要在Firefox浏览器(也即当前实际被操作的浏览器)中进行页面刷新,重新给Firefox浏览器中的新页面注入Javascript脚本,并继续在新开启的页面中进行操作行为的同步操作。

[0106] 以上对Selenium-WebDriver扩展进行了介绍,而Watir实际上与Selenium-WebDriver扩展类似,也是一种Web应用的自动化测试框架。Waitr是一个使用Ruby实现的开源Web自动化测试框架,相对于那些庞大的商业工具来说,它很小巧,也很灵活,提供的功能也足够用。同样,Waitr扩展也提供了多种API,支持开发人员使用他们偏爱的编程语言来创建测试。

[0107] 因此,关于本申请实施例提供的前述兼容性测试方案,在实际应用中,既可以使用Selenium-WebDriver扩展来实现同步,也可以使用Waitr扩展来实现同步。只不过,Selenium-WebDriver扩展的功能更强大,对各种浏览器的支持方面也比较完善,而Waitr扩展通常仅对IE浏览器有比较好的支持,因此,可以根据实际需要进行选择。另外,也可以将两种扩展相结合,例如,针对IE浏览器就使用Waitr扩展来实现同步,而针对其他的浏览器则使用Selenium-WebDriver扩展来实现同步,等等。关于Waitr扩展,具体的实现方案与Selenium-WebDriver扩展类似,这里不再详述。

[0108] 为了更好地理解本申请实施例一提供的技术方案,下面以一个流程图为例进行详细介绍。

[0109] 参见图3,可以包括以下步骤:

[0110] S301:根据参数开启相应浏览器;

[0111] S302:实际被操作浏览器的当前页面中注入JS;

[0112] S303:实际被操作的浏览器中每进行一次操作,就会触发鼠标按下(mousedown)以及键盘按下(keydown)事件,此时,可以获取被操作元素的路径(path)、被按下按键的keycode、输入框元素的输入内容等;

[0113] S304:通过XHR对象向TCP Server发起连接请求;

[0114] S305:TCP Server根据预先为此次连接分配的端口号创建Socket实例,以建立起TCP连接,接收到实际被操作的浏览器发送过来的操作信息;

[0115] S306:TCP Server通过Selenium-webdriver扩展(或者Waitr扩展)将接收到的操作信息传给其他浏览器,根据操作信息中包括的被操作元素路径等,在其他浏览器查找到该元素做以下相应操作:;

[0116] S307:判断被操作元素是否为输入框(input)类型,如果是,进入步骤S308,否则进

入步骤S310;

[0117] S308:从所述操作信息中获取input对象的输入内容;

[0118] S309:对被操作元素进行设置(set)操作,以将输入内容填入到输入框中;

[0119] S310:判断操作类型是否为鼠标点击操作,如果是,则进入步骤S313,否则进入步骤S311;

[0120] S311:如果鼠标点击的对象为链接(link)元素,则会在打开新的页面,并在实际被操作的浏览器新打开的页面中重新注入JS脚本;

[0121] S312:继续在新页面中监控操作行为;

[0122] S313:判断页面是否发生跳转,如果是,则进入步骤S311,否则,进入步骤S314;

[0123] S314:继续在当前页面中监控操作行为;

[0124] S315:判断此次测试是否结束,如果是,则结束,否则返回步骤S303。

[0125] 以上对本申请实施例一进行了详细的介绍,在该实施例一中,在捕获到测试人员在一个浏览器中的操作行为之后,可以通过Socket通信机制,将捕获到的操作信息发送到Server中,并由Server通过已安装的Selenium-WebDriver扩展或Waitr扩展将操作信息同步到其他浏览器,并在其他浏览器中模拟执行相应的操作行为。在这种方式下,每监控到一个操作行为,都可以通过建立连接的方式发送给Server端,并由Server同步到其他浏览器,可以保证信息同步的可靠性。而在另一种实现方式中,也可以不必通过Socket通信机制进行操作信息的发送,下面对此进行介绍。

[0126] 实施例二

[0127] 在该实施例二,同样可以向当前实际被操作的浏览器的当前页面中注入JS,来监控并捕获测试人员的操作行为相关信息,然后,可以在当前被操作浏览器中创建一个隐藏元素,将获取到的消息作为其属性记录下来,并通过Flag标志位等方式来通知测试程序该隐藏元素的属性已经更新,测试程序在发现该标志位的值为“true”之后,就可以根据该隐藏元素的ID定位到该元素,并读取该元素的属性值,然后通过Selenium-WebDriver扩展或Waitr扩展同步到其他浏览器中。也就是说,在这种方式下,不需要再创建Server端,就可以将捕获到的操作行为相关信息发送到其他浏览器。但需要说明的是,由于针对一个操作行为而言,捕获到的操作信息一般需要体现多个方面的信息,例如,当前操作行为的操作类型(包括鼠标操作、键盘操作等)、操作对象元素的path等,如果操作对象是输入框元素,则还可能包括输入内容等等。因此,在将这些信息作为某页面元素的属性进行保存时,可以采用多个字段进行描述,各个字段之间采用预置的分隔符(例如@、%等)进行分隔。每捕获到一个操作行为,就可以产生一条操作信息,将其写入到隐藏元素的属性中,将原有的属性值覆盖,并将Flag标志位的值置为“true”;测试程序根据该标志位可以判断出产生了新的操作信息,然后读取隐藏元素的属性值获取该操作信息,之后便将Flag标志位的值置为“false”,以此循环。该实施例二提供的这种方式中,实现过程中不需要借助于Server端,因此有利于缩短信息的传送路径。

[0128] 与本申请实施例提供的多浏览器兼容性测试方法相对应,本申请实施例还提供了一种多浏览器兼容性测试系统,参见图4,该系统可以包括:

[0129] 启动单元401,用于根据输入参数启动至少两个浏览器,并在所述至少两个浏览器中打开指定网页;

[0130] 监控单元402,用于确定其中一个浏览器为实际被操作的浏览器,监控用户在所述实际被操作的浏览器中的操作行为,并获取与所述操作行为相关的操作信息;

[0131] 调用单元403,用于调用web应用自动化测试框架的API函数,将与所述操作行为相关的操作信息同步到其他浏览器中,并在其他浏览器中模拟执行所述操作行为,以便在所述其他浏览器中展现出执行所述操作行为后对应的界面。

[0132] 具体实现时,在获取到操作行为信息之后,为了将其同步给其他浏览器,一种实现方式下,可以通过Socket通信机制,将获取到的操作行为信息传送给Server,然后再由Server通过web应用自动化测试框架将操作行为信息发送到其他浏览器,并在其他浏览器中模拟相应的操作行为。参见图5,在这种实现方式下,所述调用单元403可以包括:

[0133] 服务端创建子单元4031,用于创建服务端;

[0134] 通信子单元4032,用于通过Socket通信机制,将与所述操作行为相关的操作信息发送到所述服务端;

[0135] 间接调用子单元4033,用于由所述服务端调用web应用自动化测试框架的API函数。

[0136] 其中,所述通信子单元4032包括:

[0137] 请求子单元,用于在每次获取到与所述操作行为相关的操作信息之后,为待建立连接分配端口号,并创建XMLHttpRequest对象,通过所述XMLHttpRequest对象向所述服务端发送连接请求;

[0138] Socket实例创建子单元,用于所述服务端基于所述端口号为每次收到的连接请求创建Socket实例,建立起相应端口上的连接,通过所述连接接收与所述操作行为相关的操作信息。

[0139] 具体实现时,该系统还可以包括:

[0140] Socket实例关闭单元,用于在每次接收完所述操作信息之后,将针对此次连接请求创建的Socket实例关闭。

[0141] 另外,该系统还可以包括:

[0142] 服务端关闭单元,用于在接收到结束此次测试的指示后,将所述创建的服务端关闭。

[0143] 在另一种实现方式下,也可以不必借助于Server端,而是直接调用web应用自动化测试框架的API函数,参见图6,在这种实现方式下,所述调用单元403可以包括:

[0144] 隐藏元素创建子单元4034,用于在所述实际被操作的浏览器的当前页面中创建一个隐藏元素;

[0145] 属性信息记录子单元4035,用于将与所述操作行为相关的操作信息作为所述隐藏元素的属性信息进行记录;

[0146] 直接调用子单元4036,用于通过读取所述隐藏元素的属性值获取与所述操作行为相关的操作信息,并调用web应用自动化测试框架的API函数。

[0147] 在实际应用中,所述监控单元402可以包括:

[0148] 监听子单元,用于通过注册钩子函数,监听用户在所述实际被操作的浏览器中的鼠标及键盘操作;

[0149] 信息获取子单元,用于当监控到鼠标操作时,获取被操作页面元素的类型及路径,

并在所述页面元素为输入框类型时,获取输入到输入框中的内容;当监控到键盘操作时,获取被操作按键对应的keycode;

[0150] 所述调用单元403具体用于:

[0151] 调用web应用自动化测试框架的API函数,将监控到的操作类型、被操作页面元素的类型及路径、输入到输入框中的内容、被操作按键对应的keycode同步到其他浏览器中,并在其他浏览器中模拟执行所述操作行为。

[0152] 能够捕获测试人员在某浏览器中的操作行为,并通过调用浏览器自动化测试框架的API函数,将操作行为相关的操作信息同步发送到其他浏览器中,并在其他浏览器中模拟操作行为的执行。也即,将测试人员在某一浏览器的操作行为,即时同步到其它浏览器中,这样测试人员只需要在一种浏览器中进行操作,就能同时看到其它浏览器中进行相同操作的效果。这样既避免了测试人员在不同浏览器中分别进行相同操作的枯燥,同时有效地提升测试效率,并且能够直观、有效的监控到各浏览器页面的兼容性。

[0153] 通过以上的实施方式的描述可知,本领域的技术人员可以清楚地了解到本申请可借助软件加必需的通用硬件平台的方式来实现。基于这样的理解,本申请的技术方案本质上或者说对现有技术做出贡献的部分可以以软件产品的形式体现出来,该计算机软件产品可以存储在存储介质中,如ROM/RAM、磁碟、光盘等,包括若干指令用以使得一台计算机设备(可以是个人计算机,服务器,或者网络设备等)执行本申请各个实施例或者实施例的某些部分所述的方法。

[0154] 本说明书中的各个实施例均采用递进的方式描述,各个实施例之间相同相似的部分互相参见即可,每个实施例重点说明的都是与其他实施例的不同之处。尤其,对于系统或系统实施例而言,由于其基本相似于方法实施例,所以描述得比较简单,相关之处参见方法实施例的部分说明即可。以上所描述的系统及系统实施例仅仅是示意性的,其中所述作为分离部件说明的单元可以是或者也可以不是物理上分开的,作为单元显示的部件可以是或者也可以不是物理单元,即可以位于一个地方,或者也可以分布到多个网络单元上。可以根据实际的需要选择其中的部分或者全部模块来实现本实施例方案的目的。本领域普通技术人员在不付出创造性劳动的情况下,即可以理解并实施。

[0155] 以上对本申请所提供的多浏览器兼容性测试方法及系统,进行了详细介绍,本文中应用了具体个例对本申请的原理及实施方式进行了阐述,以上实施例的说明只是用于帮助理解本申请的方法及其核心思想;同时,对于本领域的一般技术人员,依据本申请的思想,在具体实施方式及应用范围上均会有改变之处。综上所述,本说明书内容不应理解为对本申请的限制。

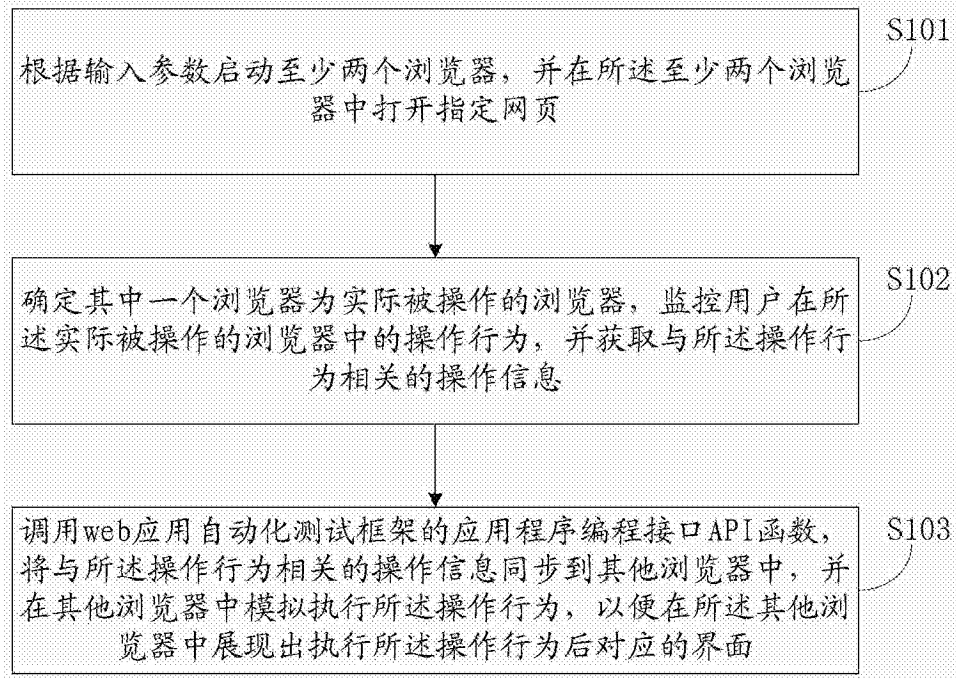


图1

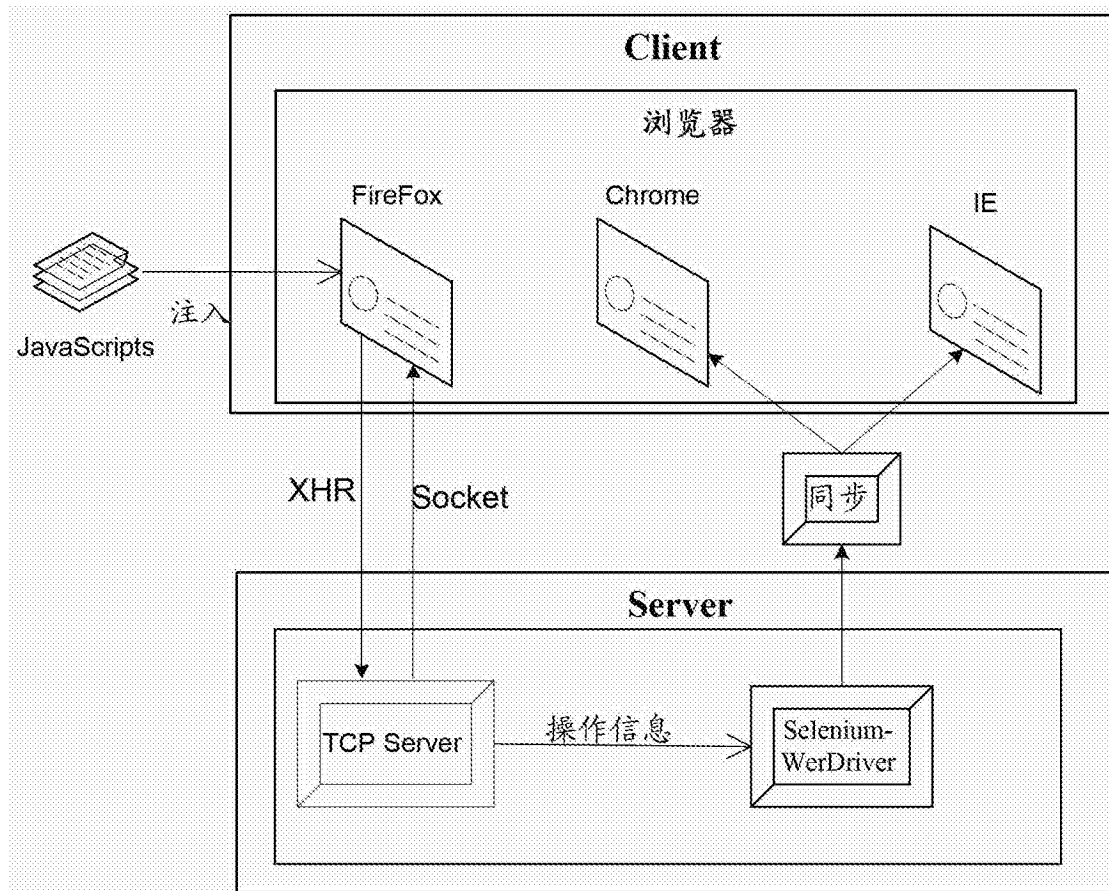


图2

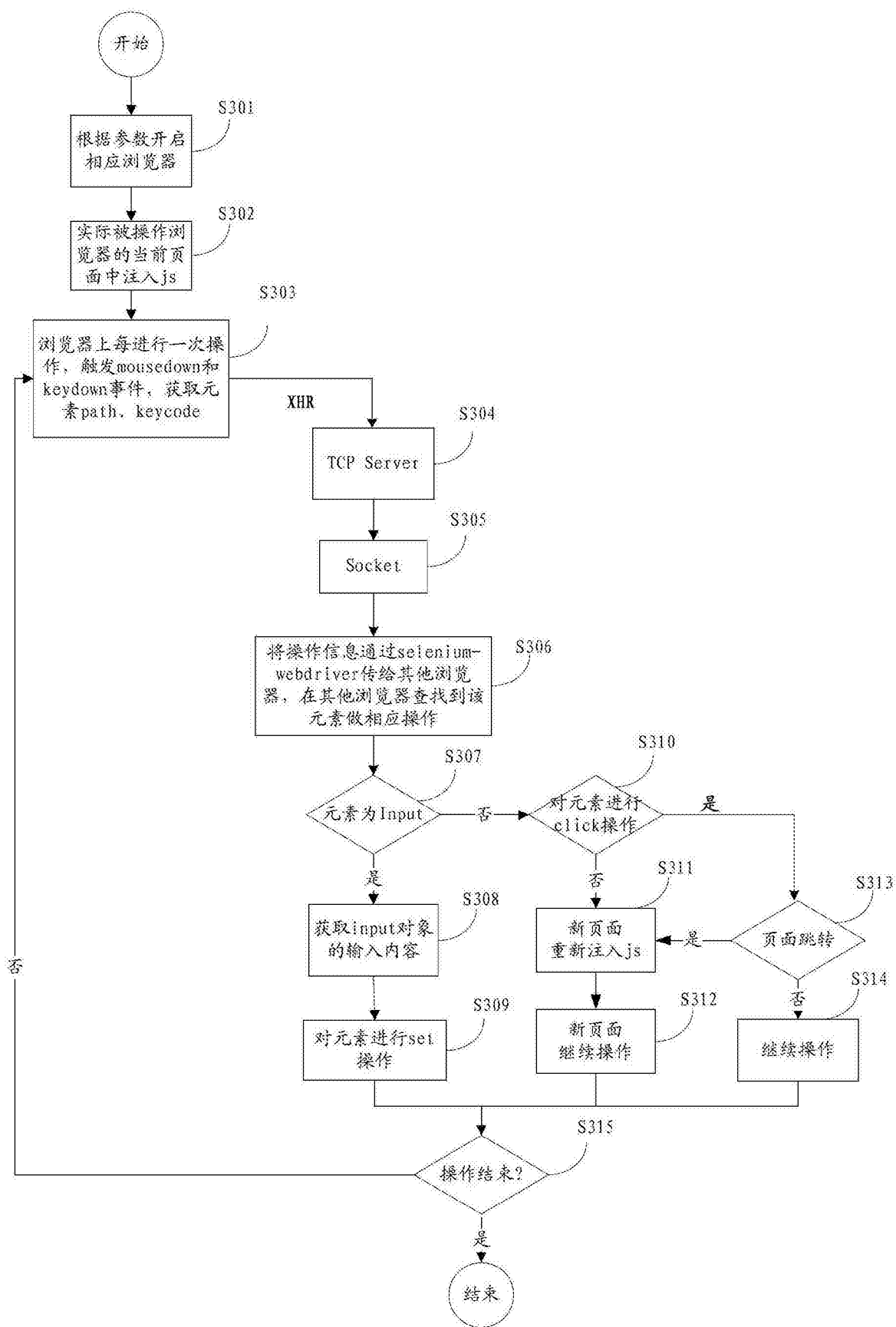


图3

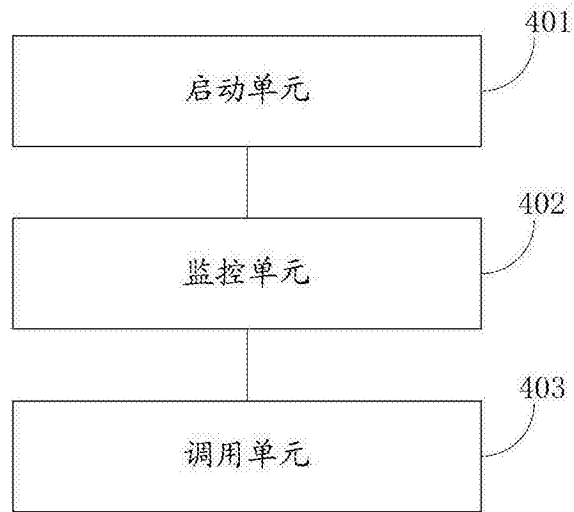


图4

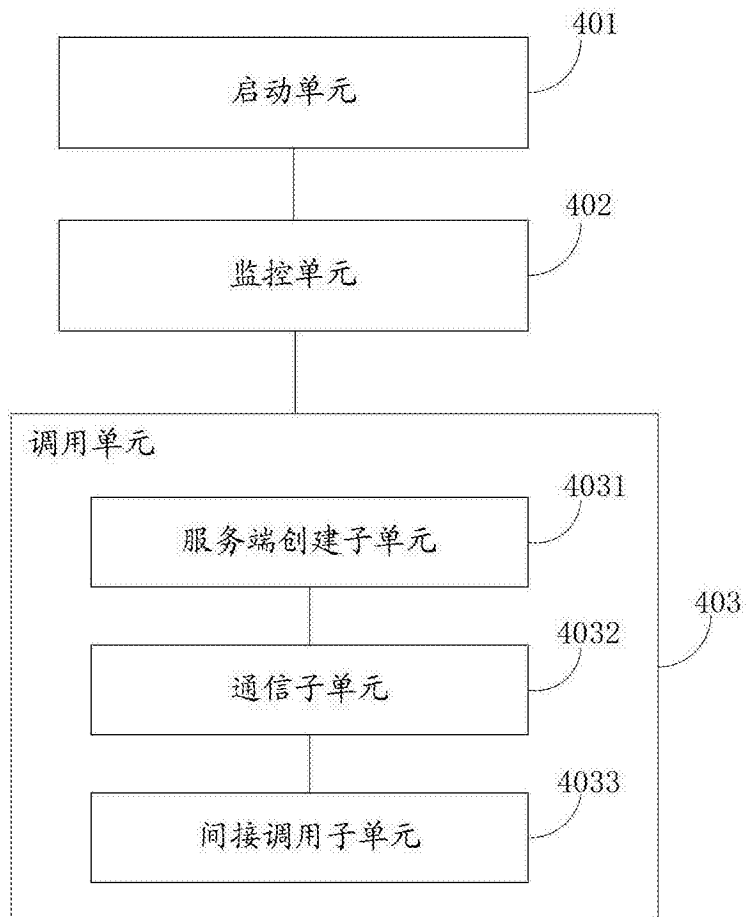


图5

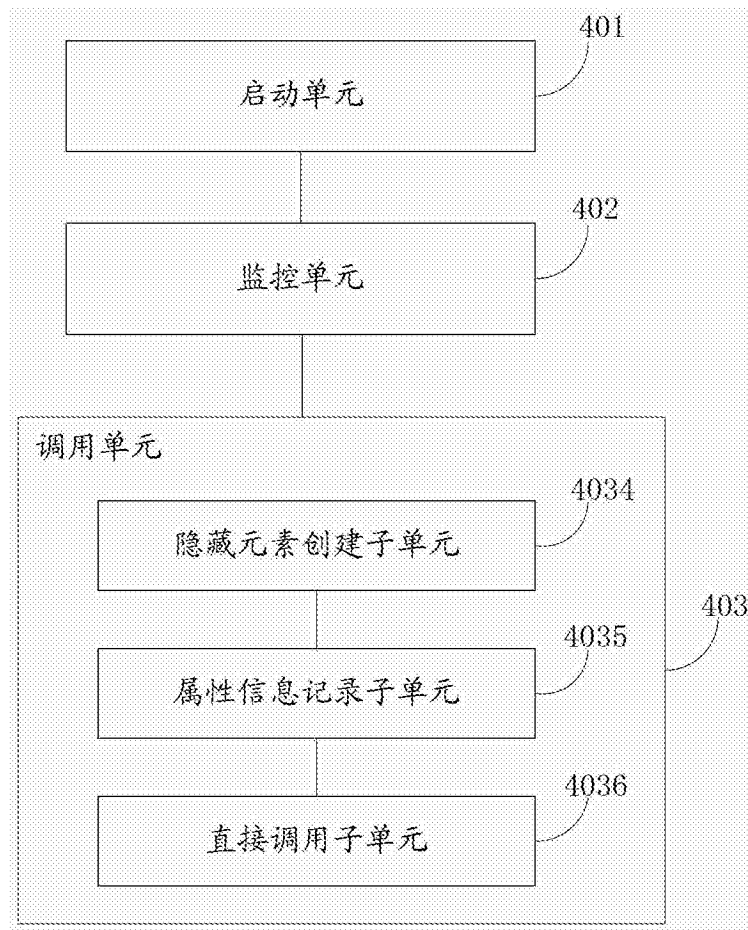


图6