**云南大学软件学院**

**形式化方法实验报告**

**题 目： Synthesizing Web Element Locators**

**姓 名 ：** 李翔宇

**学 院 ：** 软件学院

**专业班级 ：** 2016级领域软件工程

**学 号 ：** 12016000761

**指导教师** ： 王炜

**日 期 ：**  2016年 12月 19日

# 目录

[目录 2](#_Toc20324)

[一、 实验目的 3](#_Toc10012)

[二、 实验内容 3](#_Toc3772)

[三、 研究内容与原理 3](#_Toc25145)

[四、 研究方法 4](#_Toc1174)

[五、 实现与测试 5](#_Toc21297)

# 实验目的

本次实验的根据为：加拿大不列颠哥伦比亚大学的Kartik Bajaj、Karthik Pattabiraman和Ali Mesbah发表的论文Synthesizing Web Element Locators。根据论文内容，对论文中所述研究进行实现与验证。

# 实验内容

1. 理解论文中所描述的问题、研究动机、方法及其原理。
2. 根据给出的方法对其内容进行实现。
3. 验证论文的研究结论。

# 研究内容与原理

该论文的主要内容是实现一个合成网页元素定位器。作者认为目前每年都有大量的网络应用被开发出来，其中包含着非常多的网页，许多动态网页的构成十分复杂，实际开发中往往要在很庞大的DOM（Document Object Model）树中调用许多DOM元素，这样的工作使用人工定位的方式是很耗费时间的。因此本论文着手开发一种技术解决复杂的多元素所带来的问题，并且实现了一个叫做LED（Live Editor for DOM）的工具用于帮助网页开发者在JS/css代码中编写DOM元素定位器时与DOM元素进行交互，工具提供了在浏览器中定位网页元素的功能。

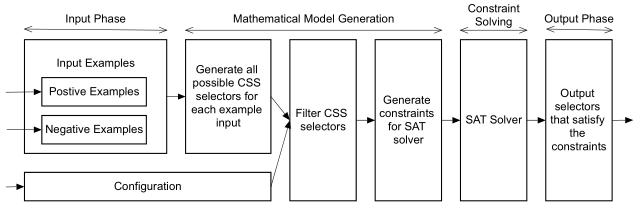
作者认为一个好的DOM定位器必须满足如下条件：

* DOM元素在DOM层上的依赖要最小化，这能保证在对代码进行小改动时定位器不易失效。
* DOM元素定位器的描述必须详尽到足以表示需要的DOM元素以及避免不必要的DOM元素的程度

为了满足以上条件，需要对DOM层中的所有选择器进行排序然后结合最佳的选择器来生成需要的DOM元素定位器。但由于这一过程非常曲折，已有研究中有人使用了DOM树中的可用信息作为标准进行排序，但没有任何人使用需要查找的元素本身的特性进行排序的。后续的研究就是基于元素本身的特性进行的。

# 研究方法

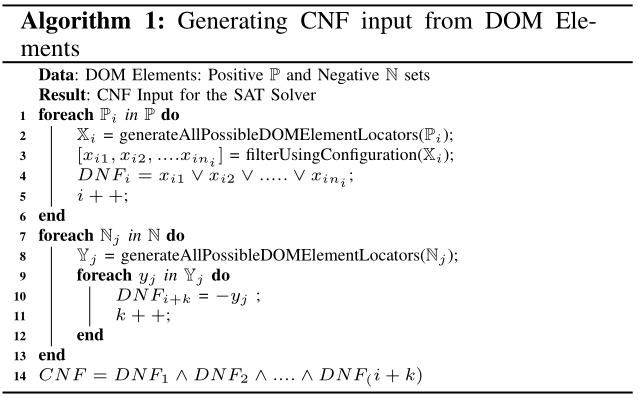
对DOM元素定位器代码的合成过程概括如下图：



第一阶段是进行输入收集。这一阶段分为两个部分：第一部分是对实例进行采集，用户需要设置希望获得的元素和希望排除的元素，它们一起构成了实例集合。如在网页中你要求找到的元素必须包含li元素，你就需要将li元素设置为PE。如果你不希望结果中出现包含如html元素的结果，就将html设置为NE。第二部分是对合成定位的过程进行参数配置，以此对合成过程进行控制和限制，保证执行效果。主要有三个参数：

* DOM元素定位器的深度。增加深度相当于增加元素的依赖。优点是能够帮助排除不想要的元素，然而会使得DOM元素操作效果变弱。因此开发者可以手动选择搜索深度。
* 最大执行时间。由于搜索过程会消耗大量时间，尤其是在大项目中。因此开发者可以限制最大执行时间从而防止大量计算导致计算机卡死。
* 搜索范围。搜索范围用于限制被选中元素的数量，有最小范围、局部范围、全局范围三种可选择。如开发者只希望搜索选中元素本身，则选择最小范围搜索，如希望获取多个元素，并且元素共处于一个自树内，则可以选择局部范围。

第二阶段是生成数学模型。这一阶段首先会根据输入的实例集合生成所有可能的CSS选择器。然后结合用户设置对CSS选择器进行过滤。然后根据结果生成约束。下图是对这一过程的伪代码解释：



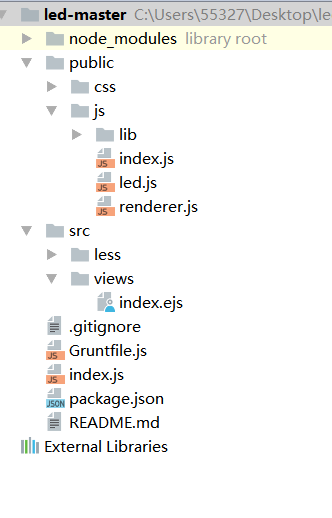
其中CNF表示的是结合的标准形式，DNF表示的是分离的标准形式。算法的输入是获取的实例集合。首先计算PE部分，每一个元素都计算出它有可能使用的定位器，然后带入用户参数进一步过滤结果，每一个元素得到的一个结果称之为一个DNF。类似的计算需要排除的NE元素，将其结果取负值（即排除掉）得到一个DNF。最后将所有DNF结合（取交集）得到CNF。

第三阶段是根据约束求解。这一步将从实际网页中选取DOM元素并且验证其是否符合第二阶段得出的约束条件，如符合条件则记为一种解决方案。因为最终要得到的选择器是最佳的，所以需要找到所有可能的解决方案后进行排序以得到最优解。这个过程中使用了由Zhao等人提供的递归方法进行实现。

第四阶段是输出阶段。根据上一阶段所得结果，程序对结果进行显示。用户可以对结果进行评价（正确与否），随后算法会重新计算并更新输出。

# 实现与测试

LED工具使用nodejs 开发。其项目结构如下：

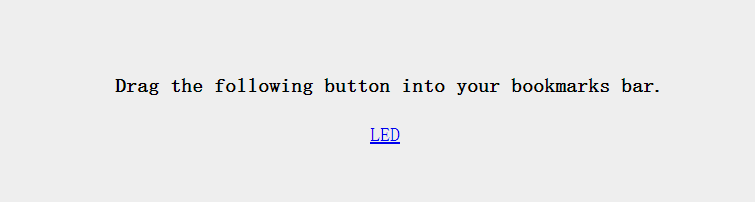


其中index.ejs为项目入口，lib文件夹中包含了所编写的库文件，index.js文件内为项目源码的实现。

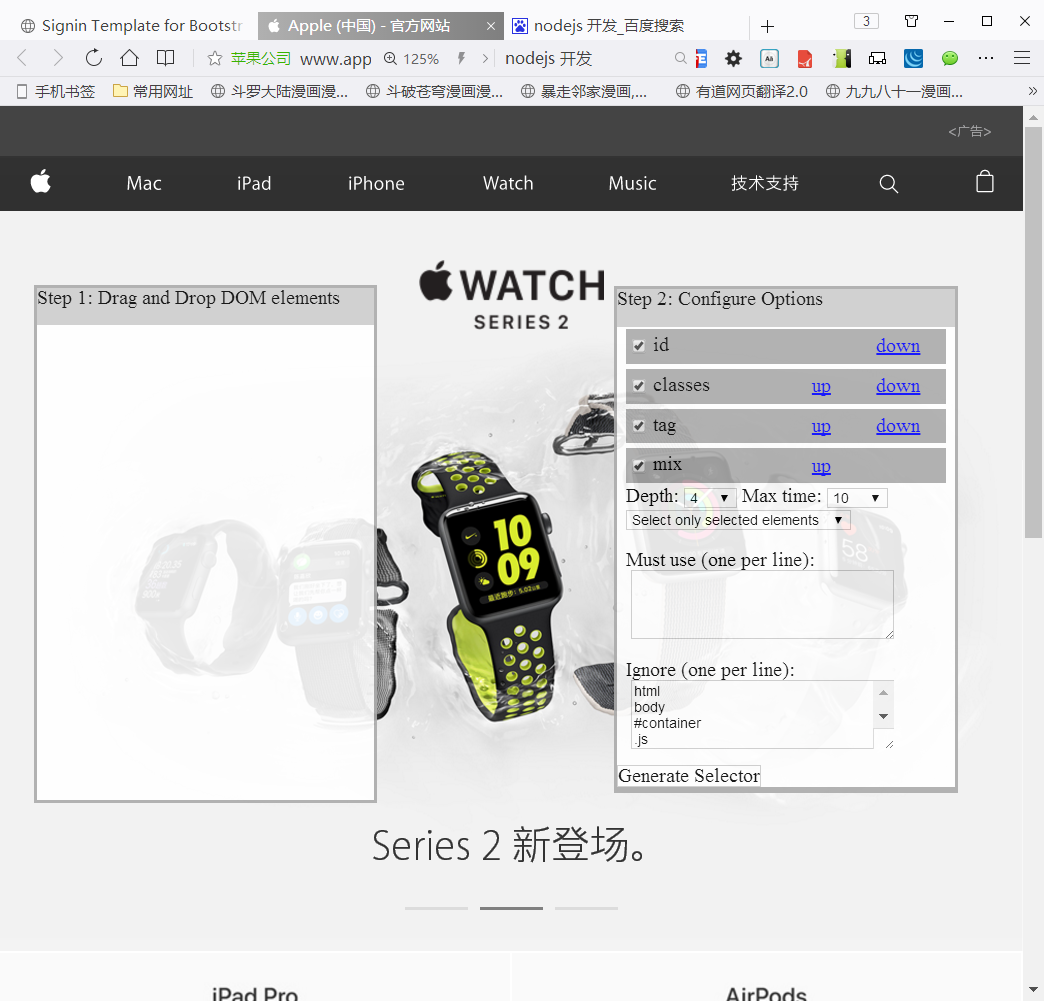
Node\_modules文件夹内为nodejs项目的依赖库。

测试过程如下：

1. 使用CLI运行工具。在项目根目录下打开命令行，运行命令：node index.js。打开浏览器，访问http://localhost:3000/网址打开工具。下图为工具入口界面

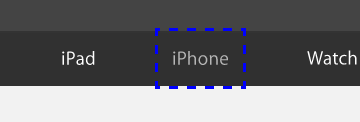


1. 打开苹果公司官网作为测试网页。
2. 按住工具页面的LED按钮拖动至苹果官网页面，成功在测试页上开启工具。

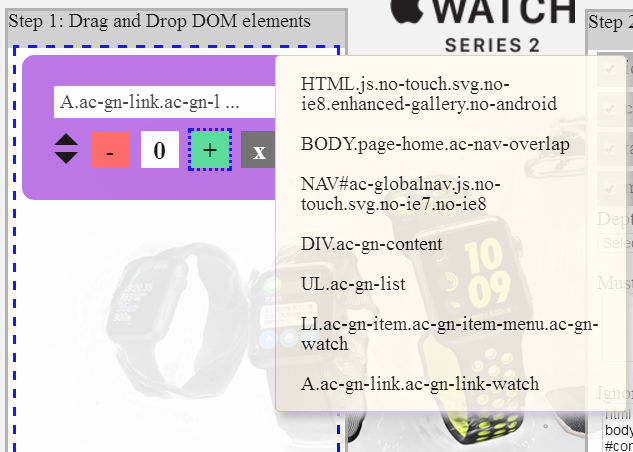


1. 功能测试。

对鼠标选取的页面元素进行加亮显示，测试正常。



将元素拖入工作区域，能够显示出该元素的所有父节点。测试正常。



设置以id/classes/tag/mix为序，搜索深度为4，最大搜索时间10秒，搜索范围为仅选择被选元素。生成的结果显示该元素受21个CSS选择器影响。测试成功

