各為口委大家好，我是研究生林稟宸。我今天要報告的論文題目是“支援多國語言的Robot Framework網頁自動化驗收測試工具的功能改善與擴充”

大綱，本論文將分為五個項目進行介紹，分別是研究動機與目標、相關背景知識、研究方法與實作、測試案例分析，以及結論與未來展望。

# 研究動機及目標

## 研究動機

根據朱峻平的論文:「支援多國語言的 Robot Framework 網頁自動化驗收測試」，透過使用者提供的 JSON 格式翻譯檔，建立出翻譯路徑檔，加上程式定義的翻譯邏輯，即可在不改動Robot Framework 測試腳本的情況下，完成多國語言網頁自動化驗收測試

然而，第一版的i18n工具仍然存在著許多待改善之處，像是:

1. 只支援 7 種 Robot Framework的原生關鍵字，如果未來測試腳本使用到其他未支援的 Robot Framework 原生關鍵字，便會發生錯誤。
2. XPath的翻譯對象僅限於網頁上的text()、normalize-space()、@title三種HTML屬性。如果使用者的測試腳本是用沒有列舉出來的屬性撰寫，測試便會出錯
3. 測試腳本執行期間，若遭遇一詞多譯的情況，目前i18n工具僅於報表上列出待翻譯詞的可能翻譯有哪些，並從中選取會使測試通過的翻譯詞當作當前翻譯。但在一些情況下，這種作法會有問題。

例如: XPath內部使用contains(Support->支援)撰寫，使得畫面上要驗證的資訊(支援服務)只要包含contains中的內容，測試就會通過

例如: 又或者畫面上剛好有多個元件(支援)符合翻譯後的測試腳本(Support-> 支援、\*支援服務)，

1. 沒有包裝成可以讓使用者直接安裝後使用的擴充工具，停留在使用者必須將 github 上的專案 clone 下來，並且在該專案上開發測試腳本的階段。造成使用者的許多不便

## 研究目標

1. 擴充完剩下的 Robot Framework 原生關鍵字代理，使得所有常用原生關鍵字的參數部分都能正確的被翻譯。
2. 撰寫出一套新的 XPath 翻譯邏輯，支援除了 @title、normalize-space()、text() 以外，其他 HTML 屬性的翻譯檢查。
3. 在執行完測試腳本後，開啟一詞多譯的圖形化使用者介面，讓使用者可以從中去選擇，
4. 將i18n工具包裝成為可以 pip install的python模組。

其中，一詞多譯UI的目標要能達成已下功能:

1)記錄當下關鍵字的名稱、參數部分、待翻譯詞、所有翻譯選項，

2)在使用者選擇了希望的翻譯後，能夠產生一個設定檔之後再次執行測試腳本時，i18n 工具便會根據設定檔的內容去選擇適當的翻譯詞

3)因為擔心使用者可能選錯，所以提供undo功能清除設定檔中的使用者選擇

# 相關背景知識

**Robot Framework**是一個可以用來執行自動化驗收測試的開源框架語言，核心框架是由 Python編寫而成，測試者可以使用 Python 或 Java 擴充其函式庫。特色是擁有簡單的語法，以及容易理解的原生關鍵字 (Keyword)，測試

者可以視需求包裝成為更接近自然語言的關鍵字

**Robot Framework 測試腳本**由四個區塊構成，分別是Settings、Test cases、keywords、variables。

Settings區塊包含了使用到的library與resource file，也可以將 Test Setup(測試腳本執行前要做的動作)、Test Teardown(測試腳本執行後要做的動作)定義於此。

Test Cases區塊，測試者可以根據測試需求，撰寫核心的測試腳本

Keywords區塊可以讓測試者撰寫新的關鍵字，將原生關鍵字或已定義的原生關鍵字包裝成更高階、易讀的關鍵字

Variables區塊，測試者可以此處定義測試案例需要用到的 Global 變數。

**Robot Framework測試報表**如圖，在測試腳本運行結束後，Robot Framework 會產生出一份測試報表，記錄了整體的執行狀況，包含測試通過或失敗、腳本執行時間，並透過可展開的階層式圖表，方便測試者去追蹤到當前出現問題的關鍵字，進而修復錯誤。

**Robot Framework Listener**介面，用以接收測試不同時期的資訊，測試者可以實作此介面，在測試腳本執行的不同時期做出額外的操作。

本論文使用到的Listener版本是Version2，必須在實作 Listener 處加上ROBOT\_LISTENER\_API\_VERSION=2(如圖)

而本論文實作的listener函式有start\_suite()和end\_suite()，

例如改寫:start\_suite()函式，在測試開始時讀取設定檔的資訊，以利之後翻譯邏輯的判斷。和改寫end\_suite() 函式，在測試結束後開啟一詞多譯 UI，供使用者選擇。

**XPath**全名為 XML Path Language，可以用來定位 XML 檔案中某節點所處在的位置。借助XPath ，Robot Framework 測試腳本可以定位畫面上元件的位置，以對其狀態進行測試。如圖的xpath可以定位到Microsoft網頁上的橫向標題container所包含的support字樣。

**第一版 i18n 的系統架構**，如這個class diagram，包含了 3 個 Robot Framework 原生類別: Robot Framework、Listener、和 Report；第一版 i18n 所設計的 5 個系統類別:I18nListener、MappingRoutesGenerator、I18nMap、I18nTrigger、Proxy，以及 5 個實作自 Proxy類別的代理關鍵字類別: ShouldContainProxy、ElementTextShouldBeProxy、FindElement-

Proxy、ListsShouldBeEqualProxy、ShouldBeEqualProxy。

I18nListener 類別實作 Listener 類別，負責程式開始與結束時執行 start\_suite() 和end\_suite() 的內容，並對 MappingRoutesGenerator、I18nMap、I18nTrigger 三個類別進行初始化的動作。

I18nMap 類別負責處理各個代理關鍵字類別透過 I18nListener 類別的呼叫，並將待翻譯詞做翻譯後回傳。

I18nTrigger 類別負責建立各個代理關鍵字類別的實例；將代理關鍵字類別的實作包裝於 RobotFramework 原生關鍵字之外，使每次呼叫關鍵字時，必先執行代理關鍵字的實作。

MappingRoutesGenerator 類別負責以英文 JSON 翻譯檔為基準，產生出翻譯路徑檔。

Proxy 類別提供了一個介面，讓實作 Proxy 類別的代理關鍵字類別們可以根據各自的需要，去擴充內部的實作。

**第一版 i18n 系統流程** 如此sequence diagram，

首先使用者必須於 Red 編輯器中的 Additional Robot Framework arguments 處設定系統參數。之後執行測試，系 統 會 先 去 呼 叫 I18nListener 類 別 並 執 行 其 實 作。 之後初始化MappingRoutesGenerator 類別，藉由讀取由使用者自行提供的 JSON 格式翻譯檔，建立出一份由「待翻譯詞」和「Key 階層」構成的翻譯路徑檔。接著系統會依序初始化 I18nMap、I18nTrigger，以及5種代理關鍵字類別。

之後，測試腳本才會開始執行；每當腳本運行到一個有定義代理的關鍵字，系統就會去呼叫對應的代理關鍵字物件，並透過I18nMap執行翻譯邏輯。最後，當測試腳本執行結束，系統則會將一詞多譯的 warning 資訊顯示在報表上。

# 研究方法與實作

**新版 i18n 系統架構**，如此class diagram，沿用第一版 i18n 工具的架構

經過部分實作的改善，並且新增了一個用於顯示一詞多譯選項的圖形化使用者介面類別 (UI)，以及 20 種代理關鍵字的類別。如圖中紅色框起的部分。

**新版i18n新增的類別** 共有21種，其中

UI類別會在程式執行期間，記錄下遭遇到一詞多譯的詞語以及其翻譯。在程式執行後，產生一詞多譯的 UI；使用者選擇並提交希望的翻譯後，便會產生一個系統設定檔，記錄翻譯選擇。

而其他 20 種新增的代理關鍵字類別 (如表中所示)，皆實作父類別 Proxy，負責代理各自對應之Robot Framework原生關鍵字的參數部分翻譯，並將翻譯好的參數部分回傳給對應的原生關鍵字。

**新版i18n系統流程差異**

新版i18n系統執行流程，沿用自第一版i18n的基本流程，差別在測試腳本結束後；除了將一詞多譯的 warning 資訊顯示在報表上外，若是遭遇過一詞多譯，便會跳出一詞多譯的UI介面，顯示翻譯資訊，讓使用者可以從中去選擇並產生一個設定檔。之後再次執行測試腳本時，系統便會根據設定檔的內容去選擇適當的翻譯詞，同時消除報表上的 warning 提示資訊。

**擴充代理關鍵字**

第一版 i18n 工具支援 7 種 Robot Framework 原生關鍵字。尚有 31 種參數部分需要翻譯的原生關鍵字，未提供支援。如果測試腳本使用到這些未支援的原生關鍵字去執行多國語言網頁測試，則會導致出錯。因此，本論文的解法是擴充完目前 Robot Framework 版本剩下的代理關鍵字，使得有翻譯需求的原生關鍵字，其參數部分都能正確的被翻譯。而又因為Robot Framework 版本已更新到 3.2.2，SeleniumLibrary 版本則更新到 4.5.0，導致原先需要被支援的 31 種原生關鍵字，有兩種已被淘汰:分別是input\_text\_into\_prompt、xpath\_should\_match\_x\_times(如表中\*標示處)。目前只剩下29種代理關鍵字需要擴充。

以下將以 **FindElementsProxy 的 Flow Chart** 為例，敘述現今配合新增的一詞多譯 UI下，本論文是如何完成代理關鍵字的擴充和修改。(\*標示處代表此步驟沿用第一版i18n)

首先，創出記錄參數部分的變數full\_args(如第1點)。此變數之後會成為判斷當下待翻譯詞是否「已被使用者選擇翻譯」的依據。並於測試腳本結束後，顯示於一詞多譯 UI 上，最後隨著使用者的選擇一併存入設定檔內。

接著，執行翻譯。(如第2點)

隨後判斷是否遭遇一詞多譯的情況，有的話則產生 warning 資訊於報表上。(如第3點)

之後判斷此一詞多譯情況，最後會使測試通過或是失敗(如第4點)，如果會失敗的話直接將翻譯結果回傳。否則繼續判斷該待翻譯詞是否屬於重複，如果不屬於重複的待翻譯詞，則對預計開啟的 UI 做一些資料的準備。(如第5點)

若此次代理關鍵字沒有遭遇一詞多譯(如第6點)，則將第一筆翻譯同時也是唯一一筆翻譯指派給之後要回傳的 xpath。

最後，將翻譯好的參數部分回傳給 Robot Framework 原生關鍵字。(如第7點)

此頁整理了以上圖提及的7個擴充代理關鍵字的步驟，僅供參考，其中1、4、5點(分別是…)，為新版i18n新增的擴充代理關鍵字步驟。2、3、6、7點則為沿用第一版i18n的擴充代理關鍵字步驟。

**改善 XPath 翻譯邏輯**

第一版i18n 工具使用了列舉法，僅可以翻譯@title、text()、normalize-space()三種HTML屬性，若測試腳本的 XPath 是用沒有列舉出來的屬性撰寫，但卻有翻譯的需求，例如:@placeholder、@arial-label 等等，則會導致測試出錯。為了解決以上問題，本論文最後決定採用「負面表列法」，將確定不會執行翻譯的HTML 屬性在程式中用 list 的方式儲存。若 XPath 中的 HTML 屬性不在該 list 中，則都去執行翻譯檢查。

新版i18n xpath翻譯邏輯的優點是: 支援所有HTML屬性翻譯檢查，並且可以透過程式定義的不翻譯屬性，略過部分屬性，減少耗時。且將翻譯的決定權交給使用者。 缺點是:和第一版i18n的翻譯邏輯比較起來，時間成本卻增加了。

**新版i18n的Xpath翻譯邏輯** 如下圖以FindElementsProxy為例的sequence diagramn所示，

首先，被呼叫後，FindElementsProxy 透過 I18nListener 類別，呼叫 I18nMap 類別的locator() 函式執行 XPath 的翻譯。

之後，I18nMap 先定義一個內含 HTML 屬性特徵的rule\_for\_filter 變數。隨後呼叫 find\_all\_match\_word() 函式，來過濾出 XPath 中包含的所有屬性。

接著，先判斷這些屬性是否包含在「不需被翻譯屬性」的 list 中。若不包含其中，則根據屬性含有｀@＇或｀()＇，分別分配給 rule\_for\_at 和 rule\_for\_bracket變數，並呼叫 combine\_locate\_rule() 函式，得到一套新的翻譯邏輯，locate\_rule。

之後，再次呼叫 find\_all\_match\_word() 函式，得到所有符合新翻譯邏輯的 HTML 屬性，all\_match\_words。並利用迴圈遍歷 all\_match\_words 中的屬性，執行 translate() 函式，進行翻譯檢查。

接著，判斷經過翻譯後，原先的xpath是否發生變化；例如xpath從原本的

//\*[text()=｀Software’]，變成了翻譯後的//\*[text()=｀軟體’]。若 xpath 改變，則執行log\_translation\_info() 函式，將翻譯資訊顯示於測試報表上。

最後，將翻譯後的 translated\_xpath，回傳給 FindElementsProxy 類別，完成一次XPath 的翻譯。

**第一版i18n一詞多譯處理**

第一版 i18n 工具提供了 warning 資訊於測試結束後的報表上，此作法確實提醒了使用者該腳本存在著一詞多譯問題，且也顯示出目前系統採用的翻譯詞；並且是站在最大化讓測試腳本通過的立場上；當遭遇一詞多譯時，系統會將所有的翻譯都執行看看，直到一包可能的翻譯中，出現了一個會使測試通過的翻譯，則算測試通過，並將該翻譯詞呈現於報表上。

此種**一詞多譯處理方法的潛在問題**，其中一個是假如存在「多個翻譯都會使測試通過」，翻譯過後的腳本測試對象，就有機會偏離使用者原先的預期。如畫面上的xpath，是用contains撰寫，此xpath本來想要驗證的詞是”支援服務”，但Support一開始卻被系統翻譯為支援，並且通過了測試。則會使翻譯後的測試對象偏離了使用者的預期。 又或是畫面上同時存在含有不同翻譯詞的元件，支援 和 支援服務。今天xpath是//\*[@text=’Support’]，要驗證的是”支援服務”元件，但因為其中一個翻譯，”支援”率先通過了測試，而不會再去檢查後續的翻譯。

還有一種潛在問題是，測試腳本原先會發生錯誤，翻譯後卻通過的情況，例如原XPath: //\*[@text=‘Service’]會使測試錯誤，但翻譯後的xpath，//\*[@text= ‘支援服務’]卻因為翻譯檔提供之在特殊情況下的翻譯而碰巧使測試通過，如此不但偏離使用者預期，也直接改變了測試結果。

因此，本論文想到的解法是，提供一詞多譯UI讓使用者去選擇希望的翻譯詞。

**新版i18n一詞多譯UI設計** 如圖，

本論文將其設計成兩個區塊，上半部以條列的方式呈現完整的翻譯資訊，包含關鍵字名稱、參數部分、待翻譯詞、以及所有可能的翻譯選項。其中，參數部分記錄了當前關鍵字接受的使用者傳入參數，以作為和其他相同待翻譯詞的區別；因為相同的待翻譯詞，在不同情況下，可能擁有不同的翻譯。下半部則包含一行功能資訊的提示、和一個用於開啟翻譯紀錄頁面的 TransRecord 按鈕，以及提交使用者選擇並寫入設定檔的 Submit 按鈕。

翻譯紀錄的介面設計，本論文同樣將其設計成兩個區塊，上半部以條列式

呈現設定檔中現存的使用者翻譯選擇。下半部包含一個 Undo 按鈕，考量到使用者可能不小心選錯了翻譯，用於將使用者的選擇從設定檔中刪除。

**新版i18n一詞多譯UI實作** 如此sequence diagram，

當測試腳本結束時，會呼叫 I18nListener 類別的 end\_suite() 函式，若本次測試腳本遭遇一詞多譯，則呼叫 UI 類別，使其初始化並執行 run() 函式，產生一詞多譯 UI 的介面。之後，run() 函式內部會呼叫 draw\_trans\_options() 函式，其內會再呼叫 get\_transdic\_keys\_and\_values() 函式，取出待翻譯詞與其對應的翻譯，最後隨著翻譯當下的關鍵字名稱和參數部分一併顯示於一詞多譯 UI 的介面上。

如果點擊 TransRecord 按鈕則會呼叫 open\_record() 函式，打開翻譯紀

錄介面，其會將先前使用者選擇過的翻譯詞，以條列式的方式呈現，且介面上還包含一顆 Undo 按鈕。點擊 Undo 按鈕會呼叫 undo\_trans() 函式，將使用者選擇的翻譯紀錄從設定檔中清除掉，同時關閉翻譯紀錄頁面。

如果點擊 Submit 按鈕則會呼叫 output\_setting\_file() 函式，將使用者

選擇的待翻譯詞對應其翻譯以及完整參數寫入設定檔中，同時將剛剛已寫入的翻譯資訊從一詞多譯 UI 上隱藏。

此頁列出了一詞多譯UI實作所包含的8個函式，其中，add\_trans\_info()和add\_keyword\_name()雖然沒有出現在上圖的sequence diagram中，代表一詞多譯UI開啟期間沒有作用。但他們卻在測試腳本執行時，負責處理來自各代理關鍵字的翻譯資訊和關鍵字名稱，並將這些資訊記錄下來，供後續一詞多譯UI使用。

再來要介紹如何將**i18n工具設計成為可以安裝的python模組**，

首先，必須準備以下五個檔案setup.cfg、pyproject.toml、README.md、LICENSE、MANIFEST.in，並透過build建立模組，最後透過twine將建置好的模組上傳到PYPI網站上即可。

**新版i18n工具使用方法** 為透過pip install RF-i18n-tool指令，即可安裝最新版的i18n工具。

使用上則必須將JSON檔依照畫面上的路徑放置在專案目錄下，並設定系統參數(如畫面上所示)，使用者若想使用i18n系統預設的JSON翻譯檔，則可以在系統參數最後加上:i18njson即可。

# 測試案例分析

以下將以 Microsoft 中文官方網頁 [1]，以及筆者使用 Node.js [19] 自行架設的 i18n測試網頁 (i18n Testing Website) 為受測網站，提供測試案例介紹

範例一:

**Alert should be present**用於驗證網頁跳出alert視窗中的資訊。

此測試腳本會打開 i18n Testing Website(如第5行)，並雙擊 Show Alert Message 按鈕(如第12行)，使畫面跳出 alert 視窗。最後使用 Alert Should Be Present 來驗證 alert 資訊是否為“Welcome to Bing＇s website＂(如第13行)。

此圖為網頁上alert視窗的文字訊息，顯示“歡迎光臨 Bing 的網頁。

測試通過，代表參數“Welcome to Bing＇s website＂有成功被翻譯為“歡迎光臨 Bing 的網頁。

範例二:

**Count values in list**用於計算list中包含多少個特定詞

此測試腳本一開始會利用 Create List 創出一個包含「支援」和｀Software＇的 list(如第10行)，並使用 Count Values In List 來計算參數｀Support＇出現在此 list 中幾次(如第11行)，並用Log印出前一行取得的次數。

第一次執行測試通過，透過 Log 印出的數字為 1，代表參數｀Support＇被翻譯為「支援」後，出現在 list 中一次，所以測試通過；且系統將支援當成｀Support＇翻譯的其中一種，並跳出一詞多譯 UI。之後，使用者選擇「支援」作為唯一翻譯。

第二次測試通過，透過 Log 印出的數字為 1，且因為有使用者選擇的唯一翻譯，所以不跳出一詞多譯 UI 與 warning 資訊。

範例三:

**Dictionaries should be equal** 用於比較兩個dictionary是否相同。

此測試腳本，會利用 Create Dictionary 創出兩個 dictionaries，分別是 {’ 軟體’: ’ 支援’} 和 {’Software’: ’Support’}(如10、11行)，並使用 Dictionaries Should Be Equal 檢查兩者是否相同(如第12行)。

第一次執行測試通過，代表 dictionary {’Software’: ’Support’} 有成功被翻譯，且系統將「支援」當成“Support＂翻譯的其中一種，並跳出一詞多譯 UI。之後，使用者選擇「支援」作為唯一翻譯。

第二次測試通過，且不跳出一詞多譯 UI 與 warning 資訊，代表｀Support＇根據使用者選擇成功被翻譯為「支援」。

範例四:

**Select from list by label**用於根據label的值，選取網頁上的selection list選項。

此測試腳本會打開 i18n Testing Website(如第5行)，並使用 Select From List By Label 選取畫面上 selection list 的 Support 選項(如第14行)。之後用 List Selection Should Be驗證結果是否正確(如第15行)。

此圖為網頁上selection list的選項，Support在中文網頁上可能被翻譯為支援或支援服務。

第一次測試通過，而畫面上選擇了「支援」，代表參數｀Support＇有成功被翻譯，且系統將「支援」當作｀Support＇的其中一種翻譯，隨後跳出了一詞多譯 UI。之後，使用者選擇了「支援服務」當作｀Support＇的唯一翻譯。

第二次測試腳本通過，畫面上被選擇的 selection list 顯示「支援服務」，代表參數｀Support＇成功被翻譯為使用者前一次的選擇。

範例五:

**\*Page should contain element**用來驗證畫面上應該包含某特定的元件。此代理關鍵字屬於修改第一版i18n的代理關鍵字。

此測試腳本，會打開 Microsoft 中文官方網頁(如第5、6行)，前往支援頁面(如第11行)，之後使用Page Should Contain Element 驗證畫面上的｀Support＇文字(如第13行)。

此圖紅框為page should contain element要驗證的網頁元件，且Support對應到網頁上正確的翻譯為支援服務。

第一次測試通過，代表參數｀Support＇有成功被翻譯，且系統將「支援」當作｀Support＇的其中一種翻譯，隨後跳出了一詞多譯 UI。之後，使用者選擇了「支援服務」當作｀Support＇的唯一翻譯。

第二次測試腳本通過，且沒有跳出一詞多譯 UI，代表參數｀Support＇成功被翻譯為使用者前一次的選擇-支援服務。

範例六:

**Table should contain**用來驗證網頁上的table是否包含特定的元件。

此測試腳本，會打開 I18n Testing Website(如第5行)，之後使用 Table Should Contain驗證畫面上的｀Support＇文字(如第13行)。

此圖為待測網頁上Table的欄位，可見包含三個欄位，且Support對應到中文網頁上的正確翻譯應為支援。

第一次測試通過，代表參數｀Support＇有成功被翻譯，且系統將「支援」當作｀Support＇的其中一種翻譯，隨後跳出了一詞多譯 UI。之後，使用者選擇了「支援」當作｀Support＇的唯一翻譯。

第二次執行測試腳本通過，且沒有跳出一詞多譯 UI，代表參數｀Support＇成功被翻譯為使用者前一次的選擇。

範例七:

**改善XPath翻譯邏輯後的驗收測試**

以下測試腳本將會去測試xpath中的@placeholder屬性能否正常的被翻譯。

首先此測試腳本會打開 Microsoft 中文官方網頁(如第5、6行)，Page Should Contain Element驗證畫面上的｀How can we help you＇文字(如第13行)。

此圖為網頁上含有@placeholder屬性的元件，且@placeholder屬性的值- How can we help you，對應到網頁上正確的翻譯應為”我們該如何協助您?”

根據測試報表結果可得知，第一版的 i18n 工具，會因為無法對 XPath 內的

@placeholder 屬性做翻譯，而使得測試發生錯誤。

而新版的 i18n 工具，因為改善了翻譯邏輯，所以支援各種 HTML 屬性的翻譯。並成功將“how can we help you?＂翻譯為「我們該如何協助您」，進而使測試通過。

範例八:

**一詞多譯情況下所產生的一詞多譯UI介面**

以下將以Test should be equal測試腳本為例，說明遭遇一詞多譯的情況下，測試結束後i18n工具會如何產生一詞多譯UI介面，以及有什麼功能。

此測試腳本首先會創出word1和word2，其值分別為More和Support(如第12、13行)，之後用Should Be Equal驗證word1、word2的值是否符合腳本預期(如第14、15行)。

在此情況下，Should be equal會發生一詞多譯的情形，進而在測試結束時產生一詞多譯UI介面。上面分別記錄了遭遇一詞多譯的關鍵字其完整翻譯資訊:有關鍵字名稱、參數部分、待翻譯詞、以及翻譯可能的選項，供使用者做選擇。當使用者選擇了希望的翻譯並按下 Submit 按鈕後，系統便會將翻譯選擇寫入一份設定檔，以利下次執行到同一份腳本的同一關鍵字時，直接套用選擇的翻譯，同時清空一詞多譯介面上的翻譯選項。

假如使用者選擇錯誤時，可以透過點擊 TransRecord 按鈕來打開翻譯紀錄的介面，上面記錄著使用者曾經的翻譯選擇，並有一顆 Undo 按鈕。使用者選擇了要清除的翻譯紀錄後，按下 Undo 按鈕，系統便會將設定檔中的該筆資料刪除，同時關閉翻譯紀錄介面，當再次點擊TransRecord打開翻譯紀錄介面時，可以發現被選擇清除的資料已消失。

例外一: **若原先測試腳本不通過**

若原先的測試腳本並不會通過的話，則不管是否遭遇一詞多譯，都不會跳出一詞多譯UI，僅會於測試報表上顯示一詞多譯的 warning 資訊。因為讓使用者選擇錯誤的翻譯是沒有意義的。

例外二: **若測試腳本通過，且沒有遭遇一詞多譯**

若測試腳本通過，也沒有遭遇過一詞多譯，則也不會跳出一詞多譯UI。

但有一種情況是，關鍵字的參數部分原本會遭遇一詞多譯，但因為前一次執行時使用者已針對此待翻譯詞選擇了翻譯，所以此次執行被系統視為只有一種翻譯，因此不跳出一詞多譯 UI

範例九:

**使用涵蓋多項代理關鍵字的測試腳本**

此測試腳本首先會打開 Microsoft 中文官方網頁(如第5行)，然後到支援頁面(如第11行)，之後點擊「支援」的 Logo，到新的支援頁面(如第14行)。最後驗證畫面上的橫向標語有包含 ‘Support’字樣(如第17行)，以及畫面上搜尋框有包含 “How can we help you”字樣(如第18行)。因為同時遭遇到 3 種｀Support＇需要被翻譯，系統便會逐一將它們紀錄下來，待測試腳本結束後，分別顯示於一詞多譯 UI 上。

第一次執行測試腳本結果如圖，測試通過，且最後使用者分別為三種不同情況下遭遇一詞多譯的｀Support＇，選擇了「支援」、「支援」、「支援服務」為其正確翻譯，並點擊 Submit按鈕，將選擇寫入設定檔中。

第二次執行測試腳本時，會根據設定檔內的翻譯紀錄，分別對三種

｀Support＇進行翻譯。因為Support被i18n系統根據不同情況判定為含有唯一翻譯，所以測試結束後，並不會顯示一詞多譯 UI。

範例十:

**以指令使用i18n工具來執行測試腳本**

在用command執行i18n工具前，使用者必須在專案目錄下準備自定義的JSON翻譯檔。之後，我們使用指令執行測試腳本，跳出error代表我們目前還沒有安裝i18n工具，因此無法讀取到，系統目前仍使用原生關鍵字去執行。

接下來我們使用pip install 下載i18n工具，並確認下載到的版本。

之後再次以指令使用i18n工具執行測試腳本，會發現有跳出一詞多譯UI。Submit後，會寫入設定檔，之後我們在將其用Undo清空。

若我們要使用i18n預設的JSON翻譯檔執行測試腳本，就將i18njson參數去掉，執行，會發現翻譯部分，和我們自定義的JSON翻譯檔有差異。Submit將其寫入設定檔，

再次執行時，便不會跳出一詞多譯UI。

範例十一:

**以RED IDE使用i18n工具來執行測試腳本**

接下來我們以RED IDE 來使用i18n工具執行測試腳本，首先，打開windows->preference設定系統參數。在此我們先使用預設JSON檔。

執行腳本，會跳出一詞多譯UI。Submit後，會寫入設定檔，之後我們在將其用Undo清空。因為選擇被清空了，所以再次執行還是會有跳出一詞多譯UI。

接下來使用自定義JSON翻譯檔執行測試腳本，先將前一次測試的資料清除，並將系統參數的i18njson拿掉。執行腳本，跳出一詞多譯UI，會發現翻譯部分有所不同。Submit後，會寫入設定檔，再次執行腳本，便不會再跳出一詞多譯UI。

# 結論與未來展望

結論

本論文提出了四項措施去改進第一版 i18n 工具所面臨的問題，分別是擴充代理關鍵字、支援所有 HTML 屬性的 XPath 翻譯邏輯、支援自由選擇翻譯的一詞多譯圖形化使用者介面、可以提供安裝的 Python 模組。

透過以上努力，即可讓使用此 i18n 工具的多國語言網頁自動化驗收測試腳本，不再侷限於只能使用特定的關鍵字去撰寫。並且 XPath 的撰寫也不再需要考慮只有 @title、text()、normalize-space() 會被翻譯的問題，使用者的測試腳本可以用更多元的 XPath 寫法來搭配此 i18n 工具使用。遭遇一詞多譯時，使用者也能夠在第一次測試腳本結束後，選擇內心所預期的翻譯詞來執行接下來的測試。最後，藉由將 i18n 工具包裝成可以 pip install 的 Python 模組，使用者可以隨安裝即使用本工具所提供的功能，而無須將自己的測試腳本移動到 i18n 工具的資料夾下。

論文限制

本論文的限制:

其一是某些 Robot Framework 的原生關鍵字，其參數部分接受的是特殊的字元，目前 i18n 工具是不支援的。如圖中的Get Match Count 測試腳本，Get Match Count接受的第二個參數是｀S\*＇，用來抓出list中符合的字串(如第10行)。若是強行將｀S\*＇翻譯，便會直接導致測試結果出錯。

其二是某些 Robot Framework 的關鍵字，其內部實作會呼叫另一個原生的關鍵字，且被呼叫的關鍵字也有一詞多譯的需求。當使用者為第一次執行測試的一詞多譯選擇了翻譯後，並執行第二次測試腳本時，因為此時傳入代理關鍵字的參數部分已改變，系統會將其判定為不同的待翻譯詞，而需要使用者事後再選擇一次。

如圖中的dictionary\_should\_contain\_item原生關鍵字的實作，會呼叫dictionary\_should\_contain\_key。導致第一次執行時傳入代理關鍵字的｀More＇，和第二次執行時傳入的｀More＇，由於參數紀錄的局部從 [｀支援＇,｀支援服務＇] 變成 [｀支援＇]，因此被系統判定成不同的待翻譯詞，而需要重新選擇。

未來展望

未來展望的部分，可以根據 i18n 工具尚存在的使用限制，未來朝向優化特殊字元的翻譯方式，以及改善重複翻譯詞的判定邏輯去努力。而且目前 JSON 翻譯檔還需要由使用者提供，未來可以研究如何透過設計爬蟲，抓取多國語言網頁上的對應翻譯詞，自動將 JSON 翻譯檔產生出來，將人工的成本降低。進而讓「多國語言網頁自動化驗收測試」工具變得更貼近測試者的需要，錯誤率低且容易使用。