半自動詞彙擷取：簡化的詞夾子方法以及其 JavaScript 元件的開發與應用

杜協昌*

摘 要

人文學者對感興趣的主題，通常都擁有數位化的文字檔（稱爲**文本** texts）。在進行研究時，有時會需要從文本中盡可能找出某些類型的詞彙。**詞彙擷取方法**是指能夠從大量數位化文本中，擷取出有意義詞彙的演算法。**詞夾子方法**是一種半自動的詞彙擷取演算法，它在詞彙擷取的過程中，需要電腦進行大量的字串比對，也需要人力的介入以提升擷取成效。詞夾子方法在多年前，就已經被證實在中文古籍的詞彙擷取上有相當的成效。然而，或許因爲缺乏和善的使用者介面，這些年來這個方法並沒有廣泛被人文研究者所使用。

在這篇論文裏，我們將提出一個簡化的詞夾子方法，並強調使用者介入選取有效詞夾和正確詞彙的必要性。我們應用這個方法撰寫一個 JavaScript 元件，接著利用這個元件開發一套可讓使用者從自有文本擷取詞彙的系統。將演算法以元件的方式開發並發佈，有助於後續的系統開發，因爲系統開發者可以直接取用元件的功能，而不需重新實作這個演算法。這個詞夾子的 JavaScript 元件已經被一套專注於中文古籍標記的 MARKUS 系統所採用。

我們也將討論詞夾子元件在設計與實作上的一些考量。我們會展示如何利用這個元件開發簡單的系統，並且運用這個元件開發一套完整的應用系統。我們將利用這套系統擷取《紅樓夢》中的人名和衣飾名稱，並在這個過程中討論詞夾子方法的優缺點。

關鍵詞：中文古籍，詞彙擷取，詞夾子方法，軟體元件，MARKUS，紅樓夢。

* 國立臺灣大學資訊工程系博士後研究員。

Semi-Automatic Term Extraction with Simplified Term-Clips Method: Development and Applications of a JavaScript Component

Hsieh Chang Tu*

Abstract

Humanists often have their own digitalized texts. Sometimes they want to extract as many terms of specific type as possible from the texts. *Term extraction methods* are computational algorithms to extract meaningful terms from a large corpus of digitized texts. *Term-clips* method is a semi-automatic term extraction algorithm that highly demands human-computer interaction to get comfortable extraction results. The algorithm of term clips was developed years ago, and it was shown to be powerful in extracting categorical terms from Chinese ancient novels. However, perhaps due to the complicated original algorithm, or maybe due to the lack of user-friendly interface, this method does not used widely by humanists.

In this paper, we shall propose a *simplified term-clips method*. We emphasize the importance of human interaction to select effective clips and desirable terms. We adopt this algorithm to implement a JavaScript component, and use this component to develop a real system that allows people to extract meaningful terms from their own texts. Encapsulating algorithms into components is helpful to the development of more complicated systems, since system developers do not need to implement the algorithms themselves. Developers can simply include the components and reuse the functions provided by the components. This term-clip component has been used by MARKUS, an open source system developed for taking markups on Chinese ancient texts.

In addition, we shall discuss considerations in the design and implementation of the JavaScript component. We show how to use this component to make a simple application, and develop a complete system that helps people extract meaningful terms from their own texts. We illustrate how to use the system by the extractions of personal names and clothing terms from the Chinese ancient novel *Dream of the Red Chamber.*

Keywords: Chinese Ancient Texts, Term Extraction, Term-Clips Method, Software Component, MARKUS, Dream of the Red Chamber.

* Postdoctoral Researcher, Department of CSIE, National Taiwan University.

第一節　導論

數位人文 (Digital Humanity) 的一個重要目標，是利用資訊科技，幫助研究者從事人文研究。人文研究的許多基本素材來自於**文本**（texts，在此指數位化的文字檔），而許多研究問題的解決，需要在研究者有了初步的問題意識之後，從文本中找出例證或反例，進而利用這些內容對問題進行回答與解釋。尋找例證的過程中，有時研究者會想從文本中找出某個特定類別的詞彙，例如擷取文本中出現的人名、地名，武器、或者服飾的名稱等等。當文本的文字數量逐漸增加，利用人力以逐字閱讀的方式來尋找詞彙，就不但不符經濟效益，而且容易發生錯漏。因此，利用資訊技術，協助研究者從大量的文本中進行**詞彙擷取** (term extraction)，藉此減少所需的人力並提高準確率，就成爲數位人文發展的一項研究課題。

詞夾子方法 (term clips method) 是一種半自動 (semi-automatic) 的詞彙擷取方法。[1]它的概念相當直觀，試看以下幾個《宋會要輯稿》片段：

- 太祖開寶四年八月二十六日，宰臣趙普等上表請加尊號曰應天廣運……
- 十八日，宰臣章惇等請上神宗皇帝徽號曰紹天法古運德……
- 宰臣沈該等奏曰：「聖意高遠，非臣等所及。」

在這幾個例子中，很明顯地「宰臣」和「等」之間的趙普、章惇、沈該都是曾經擔任宰臣的人名。我們不難猜想，出現在「宰臣…等」之間「…」部分的文字，應該多半都是人名。事實上，如果用「宰臣…等」去比對《宋會要輯稿》文本，找出「…」所代表的文字，我們可以擷取出王旦、蔡京、湯思退、丁謂、留正等 59 個詞彙。[2] 詞夾子方法的基本概念，就是利用文字比對與計算，從文本中找出類似「宰臣…等」的文字結構（稱之爲詞夾子，請參考第三節），然後再利用這樣的文字結構，從文本中擷取出想要找的詞彙。

實務上，（張尚斌 2006）曾利用詞夾子方法從 THDL（Taiwan History Digital Library 臺灣歷史數位圖書館）的《明清臺灣行政檔案》和《古契書》中擷取人名詞彙。此外，（謝育平等 2009）也曾利用這個方法從中國古典文學典籍中擷取出人物、地名、武器、衣物等各種類別的詞彙。然而，即便詞夾子方法相當直觀，也確實能夠有效擷取文本的詞彙，但或許因爲原始的詞夾子方法仍嫌複雜，或許因爲先前開發出來的工具缺乏友善的使用介面，這些年來似乎很少人文學者利用詞夾子來輔助研究。

---

[1] 全自動 (fully automatic) 的方法在詞彙擷取過程中並不需人力的介入。半自動的方法表示，在應用這個方法時，有時需要電腦進行計算，有時需要人力的介入以提升系統的擷取成效。

[2] 我們使用的《宋會要輯稿》文本，來自於中央研究院和哈佛大學的合作計畫。在此我們要求「宰臣」和「等」之間的詞彙字數介於 2 和 3 之間，在這樣的條件下，「宰臣...等」可從文本中捕捉到 59 個相異的詞彙。其中有兩個詞彙「察京」和「召臣」並非人名：「察京」應為蔡京，起因於文本打字錯誤（八月四日，宰臣**察京**等奏請上僖祖皇帝徽號曰立道肇基積德起功懿文獻武睿和至孝皇帝）；另一方面，「召臣」則顯示詞夾子方法在本質上就可能會擷取到錯誤的詞彙（今月十八日，宰臣***召臣***等問所降德音不鎖院之故）。

人文研究者通常都有自己感興趣的文本。我們認爲，如果能夠讓研究者自己利用詞夾子來擷取這些文本的詞彙，他們將可在操作的過程中累積詞夾子方法的使用經驗與信賴程度。隨著資訊科技的進步，現今的研究者多半已熟悉網際網路與瀏覽器 (web browser) 的操作。因此，若能將詞夾子演算法封裝成現今瀏覽器廣爲支援的 JavaScript 元件 (component)，就可以方便後人利用這些元件來提供更友善的使用介面，或者開發功能更爲強大的數位人文工具。

這篇論文嘗試兼顧理論與實務。我們會向讀者介紹一個簡化的詞夾子方法，討論實作這個演算法的 JavaScript 詞夾子元件，並說明如何應用這個元件開發瀏覽器端的使用介面。最後，我們也會介紹一個利用此元件開發的完整系統，並利用它來展示簡化詞夾子方法的擷取成效。

論文的組織如下。首先，我們在第二節介紹一些從中文古籍擷取詞彙的方法，並簡單回顧詞夾子方法的歷史。接著，我們在第三節以數學模型正式介紹簡化的詞夾子方法。第四節說明詞夾子元件在設計與實作上的一些考量，它的主要目的是與開發數位人文工具的研究者探討理論和實務上的許多差異。最後，我們在第五節將花費相當長的篇幅來介紹利用這個元件所開發出來的一套詞夾子系統。在介紹系統使用方式的過程中，我們會以《紅樓夢》作爲文本，嘗試從中擷取人物與服飾的名稱，並討論詞夾子方法的優缺點。我們在論文最末提供一份附錄，記錄實驗室成員利用詞夾子系統從《西遊記》擷取兵器詞彙的完整過程，提供有興趣者作爲參考。

第二節　相關文獻

一般說來，**詞彙擷取方法** (term extraction methods) 是指從大量數位化文字檔（本論文稱之爲文本 texts）中，擷取出有意義詞彙的電腦演算法。詞彙擷取的方法很多，在 Google 或維基百科上搜尋 “term extraction”，就可以找到許多相關的學術論文，在此不多加討論。一般的詞彙擷取方法，由於目的在於處理極大量的文本，通常都是全自動的 (fully automated)，擷取過程中並不需人力的介入。這些自動的方法，在方法的評估上會強調擷詞成果的**精準率**（precision，找出的詞彙中，有多少比例是正確的）和**召回率**（recall，所有出現在文本中的欲找詞彙，實際上有多少比例被演算法所擷取出來）。然而，這些全自動的詞彙擷取方法，通常需要有量大且質精的**訓練文獻集** (training corpus，指大量經人力標註正確詞彙、用來調整演算法參數的文字檔)，才能獲得較爲滿意的精準率與召回率。對中文古籍而言，由於缺乏合適的訓練文本，因此多半都只能利用個別文本的特性，針對某些類別的詞彙來開發演算法。例如（王昱鈞等 2012）討論如何進行外來語的音譯詞擷取；（彭維謙等 2014）利用**正規表示式** (regular expression) 從地方志擷取職官資料；（彭維謙等 2012）採用**點相互關聯度**（pointwise mutual information, PMI）從《資治通鑑》擷取人名；而（嚴漢偉等 2014）則運用類似的**相互關聯度**概念，再加上其他的規則來從《明實錄》擷取地名資料。

另一方面，**詞夾子方法** (term-clips method) 則主要被用來從研究者自己的文本中，擷取出他們認爲有意義的詞彙。值得注意的是，個別研究者所關注的研究

主題一般並不相同，他們對何謂「有意義詞彙」的判定也經常有異。這使得高通用性的自動化詞彙擷取方法，對人文研究者需求，在實際應用上有著相當程度的侷限性。本文所討論的簡化詞夾子方法，將捨棄納入機器學習 (machine learning) 的機制，高度要求使用者在擷取的過程中提供選擇（從候選詞彙中選擇正確詞彙、從候選詞夾中選擇高效能的詞夾）。因此，即便使用相同的文本，由於使用者判定相關詞彙的條件不同，將可能導致不同的擷取結果。這個特性使得簡化詞夾方法被歸屬於半自動的演算法（需要電腦與使用者的互動）。它在應用與效能評估上，都與全自動擷詞演算法有著相當大的差異。

詞夾子方法最早出現在張尚斌的碩士論文（張尚斌 2006）。該方法的原始發想者謝育平在其後數年也接續進行了許多研究。在張尚斌的論文中，詞夾子方法用到「前文、詞首、詞尾、後文」四項參數，而謝育平在（謝育平 2010）將其補足爲「前文、前綴、中綴、後綴、後文」等五項參數。雖然謝育平等研究者在（謝育平等 2009）的論文中敘述他們曾進行過許多實驗，成功從《水滸傳》、《搜神記》、《三國演義》、《紅樓夢》等古籍擷取出「人物」、「動物」、「衣物」、「飲食」的相關詞彙，但由於該論文僅列出「成功找出了多少詞彙」，缺乏對擷取過程與各類詞彙的細節描述，相當程度上增加了後續研究者利用這些研究成果的困難度。

詞夾子方法的擷取成效，取決於文本在文字敘述上的結構特性，也就是所謂「同位語」的敘述結構（謝育平等 2009）。在第一節的例子中，「宰臣…等」之所以能夠有效擷取人名，主要是因爲在《宋會要輯稿》這種類型的文本中，出現於官職（宰臣）後的文字，有相當高的機率就是人物的名稱。此外，由於人名通常不會出現「等」這個字，因此出現在官職和「等」之間，長度爲 2 或 3 的字串，就非常可能是個人名。對於其他的文本（例如《紅樓夢》），「宰臣…等」就不再是個能夠有效擷取人名的詞夾。

MARKUS 是一個針對中文古籍文本所開發的半自動標記平台。它的目的是幫助研究者對文本進行標記，從而利用這些標記下來的特定詞彙，將文本轉成電腦可以進行分析的數據（何浩洋 2014）。MARKUS 是開放原始碼 (open source) 的系統，採用 JavaScript 來開發。該系統目前仍處於 beta 版，有興趣者可以留意該計畫網站的訊息來獲取最新的資訊。[3]本文所敘述的詞夾子元件，已經被 MARKUS 系統所採用，作爲進行文本採礦 (text mining) 時所用到的工具之一。

第三節　簡化的詞夾子方法

雖然詞夾子方法的概念相當直觀，但完整的的詞夾子演算法卻因爲需有「前文、前綴、中綴、後綴、後文」等多項參數而顯得複雜（謝育平 2010）。我們在這篇論文中，將只使用完整詞夾子演算法中的「前文」與「後文」參數，然後利用這個簡化後的詞夾子方法來開發 JavaScript 元件。

以下我們正式介紹簡化的詞夾子方法。令 $\mathbb{A}$ 爲所有文本字元所形成的集合

[3] http://did-acte.org

（稱之爲**字彙** alphabet）。令 L, R 爲任意兩個非空的字串，[4]我們定義**詞夾子**（簡稱**詞夾**或**夾子**）爲一個字串對 (L,R)，記爲 L..R。我們稱 L 爲這個詞夾子的**左夾** (left clip) 或**前夾**，稱 R 爲這個詞夾子的**右夾** (right clip) 或**後夾**。例如，第一節所提到的「宰臣…等」就是一個詞夾子，其中「宰臣」是這個詞夾子的左夾，而「等」則是右夾。[5]

給定兩個字串 s 與 t，若 s 爲 t 的子字串，我們將其記爲 s∈t。我們用 |s 表示字串 s 的長度，用 s•t 表示兩字串 s 與 t 串接 (concatenation) 後所形成的新字串。一份**文本** τ 是一個由字彙 𝔸 的字元所形成的字串。我們定義**單夾擷詞函數** $f_0(\tau,L,R,m,n)$，其中參數 τ 爲文本，L, R 爲某個詞夾子的左右夾，而 m, n 爲希望擷取的最短詞彙長度與最長詞彙長度（m, n 爲兩個正整數，m 不大於 n），如下：

$$f_0(\tau,L,R,m,n) = \{\, t: L \bullet t \bullet R \in \tau,\ m \le |t| \le n \,\}.$$

例如，假設我們有一份（由《宋會要輯稿》幾個片段所構成）文本 τ 如下：

- 太祖開寶四年八月二十六日，宰臣趙普等上表請加尊號曰應天廣運；十八日，宰臣章惇等請上神宗皇帝徽號曰紹天法古運德；宰臣沈該等奏曰：「聖意高遠，非臣等所及。」五年二月八日，太師文彥博言：「前通判同州知趙亢所管沙苑監馬數蕃息，乞堂除有監處知州軍、通判。」詔亢權知隴州。

那麼 f_0(τ,宰臣,等,2,3) = {趙普, 章惇, 沈該}，因爲「宰臣趙普等」、「宰臣章惇等」、「宰臣沈該等」都是 τ 的子字串。此外，由於 m=2, n=3（限制函數回傳的詞彙長度介於 2 與 3 之間），因此雖然「宰臣趙普等上表請加尊號曰應天廣運；十八日，宰臣章惇等」也是 τ 的子字串，但因最前方的「宰臣」和最後方的「等」之間的字串長度遠超過詞彙長度的上限 3，這個「宰臣…等」中間的字串「趙普等上表……宰臣章惇」並不會屬於 f_0 回傳的結果集。

單夾擷詞函數只用一個詞夾子作爲輸入。若有 n 個詞夾 $C = \{L_1..R_1, L_2..R_2, \ldots, L_n..R_n\}$，我們定義**(多夾)擷詞函數** f 爲個別應用**單夾擷詞函數**到這 n 個詞夾的聯集：

$$f(\tau,C,m,n) = \cup_{L..R \in C} f_0(\tau,L,R,m,n).$$

因此，以剛才的文本 τ 爲例，若 C={宰臣... 等, 太師…言}，那麼 f(τ,C,2,3) = f_0(τ, 宰臣,等,2,3) ∪ f_0(τ,太師,言,2,3) = {趙普, 章惇, 沈該} ∪ {文彥博} = {趙普, 章惇, 沈該, 文彥博}。換句話說，擷詞函數 f 可以利用 C 的兩個詞夾，從文本中擷取到「趙普、 章惇、沈該、文彥博」四個人名詞彙。

---

[4] 正式地說，s 為非空的字串等同於 $s \in \mathbb{A}^+$，其中 $\mathbb{A}^+ = \cup_{n \in \{1,2,\ldots\}} \mathbb{A}^n$。

[5] 注意到，當利用數學符號 L..R 表示詞夾子時，我們在 L 和 R 之間加了兩個句點。另一方面，當我們利用「宰臣…等」這類文本實例說明時，我們在「宰臣」和「等」之間加了三個句點。

擷詞函數可以幫我們利用詞夾從文本中擷取詞彙（也就是說，f 的輸入是詞夾，而輸出則是詞彙）。問題是，任意兩個非空的字串都可以構成一個詞夾子，我們該如何從大量的可能詞夾中，找出實際上對我們有幫助的夾子？例如，在上面的文本中，「太祖…四」、「臣趙…表」和「請加…曰」等等也都是詞夾，但它們夾中的（開寶、普等上、尊號）卻不太可能是我們想要找的詞彙。我們該怎麼利用電腦來找出較爲有效、類似「宰臣... 等」和「太師…言」的夾子？

爲了解決這個問題，我們定義**詞夾模具函數** g(τ,T,a,b)，其中 τ 爲文本，T 爲詞彙的聯集，而 a, b 分別表示左右夾的長度。詞夾模具函數的目的，是要幫我們利用給定的一些詞彙 T，從文本中推算出「可夾中詞彙 T 的夾子」。[6]確切地說，

$$g(\tau,T,a,b) = \{\ L..R:\ t \in T,\ L \bullet t \bullet R \in \tau,\ |L|=a,\ |R|=b\ \}.$$

以剛才文本爲例，假設我們知道 T={趙普}，那麼 g(τ,T,2,1)={宰臣…等}。換句話說，因爲文本中趙普的前面兩個字是宰臣，後面一個字是等，利用詞彙「趙普」可從文本中比對出詞夾「宰臣…等」。

詞夾子方法的核心概念，就是交替使用詞夾模具函數 g 和多夾擷詞函數 f，從文本中找出新的詞彙。我們只需在一開始知道幾個種子詞彙，就可以（利用函數 g）藉由這些種子比對出許多詞夾，然後（利用函數 f）對文本套用這些詞夾，又可以衍生出許多新的詞彙。有了這些新的詞彙，我們又可利用它們作爲新種子，從而（利用函數 g）算出新的詞夾，再運用這些新詞夾（套用函數 f）擷取詞彙……。如此交替使用函數 g 與 f，我們就有機會藉由幾個種子詞彙，從文本中擷取出多個原先並不知曉的新詞彙。

以剛才文本爲例 (a=2, b=1, m=2, n=3)，假設我們一開始知道兩個種子詞彙 T={章惇, 文彥博}。我們可以利用 g(τ,T,2,1) 比對出詞夾 C={宰臣…等, 太師…言}，然後再利用 f(τ,C,2,3) = {趙普, 章惇, 沈該, 文彥博} 擷取到「趙普」、「沈該」這兩個不屬於 T 的人名。[7]另外也請注意，在文本 τ 中還有一個人名「趙亢」，它並沒有被詞夾 C 所擷取出來。[8]

我們將簡化的詞夾子方法描述如下：

1. 給定文本 τ，使用者提供起始的詞彙集 T_0，設定左右夾長度 a, b 和詞彙長度限制 m, n。令反覆迭代次數 i := 1。
2. 利用 $g(\tau, T_{i-1}, a, b)$ 計算出**候選詞夾**集合 B_i。
3. 從 B_i 中取得下個步驟要使用的**種子詞夾**集合 C_i。
4. 利用 $f(\tau, C_i, m, n)$ 計算**候選詞彙**集合 S_i。

---

[6] 感覺起來就像是一個可產生詞夾的模具，它能夠「輸入詞彙，比對文本後輸出對應的詞夾」。

[7] 以數學式子來說，新擷取到的詞彙就是就是 f(τ,g(τ,T,2,1),2,3)-T = {趙普, 沈該}。

[8] 在註腳 2 我們曾說明詞夾可能會擷取出錯誤的詞彙（也就是說，精準率 precision 並非 100%）。這裏的目的是強調詞夾可能無法擷取出所有想要的詞彙（也就是說，召回率 recall 並非 100%）。

5. 從 S_i 取得接下來要使用的**種子詞彙**集合 T_i。
6. 如果達到終止條件（例如滿足於產生的詞彙集 T_i，或者當迭代次數 i 超過某個數目）就停止演算法；否則令 i := i + 1，回到步驟 2。

在上述的演算法中，我們並沒有清楚說明步驟 3、步驟 5 和步驟 6 的細節。步驟 3 和步驟 5 的目的，是從候選詞夾和候選詞彙中產生種子詞夾和種子詞彙。我們可以設計**評估函數** (evaluation function) 讓電腦自動篩選，也可以讓使用者介入進行挑選（謝育平 2010）。最後，步驟 6 的終止條件也相當有彈性，我們可以設計評估函數讓電腦自動結束執行，也可以讓使用者決定是否要繼續下一次迭代。

理想上，設計一個好的評估函數讓電腦自動執行，應可節省大量的人力。然而，由於目前的電腦演算法都僅能處理語法 (syntax) 而非語意 (semantics)，評估函數所篩選出來的詞彙或詞夾，在實務上經常無法令人滿意。因此在這篇論文裏，我們將強調使用者介入的必要性，並著重在軟體元件與使用者介面的開發與應用。

第四節　JavaScript 詞夾子元件的設計與實作

詞夾子元件的開發，屬於萊頓大學（荷蘭 Universiteit Leiden）、哈佛大學和臺灣大學在數位人文合作計畫 (DID-ACTE[9]) 中的一部份。該計畫的目標是開發一個開放原始碼的標註工具 MARKUS，讓人文研究者能夠從中文古籍的數位文本裡，轉換出可計算的結構化資料。[10] 目前仍處於開發階段的 MARKUS 是個針對中文古籍文本設計的半自動標記平台，它可以提供自動或半自動的標記模式，幫助沒有資訊工程師支援的人文學者對文本進行標記。在半自動模式下，使用者將可要求系統利用詞夾子或其他詞彙擷取工具從文本中找出一些詞彙，接著再從中篩選出想要的詞彙來加上標記。由於 MARKUS 最終決定以 JavaScript 進行開發，[11]臺灣大學負責的詞夾子元件自然也採用相同的程式語言（也就是 JavaScript）來撰寫。

然而，即便現今的 JavaScript 功能強大，又有免費的 jQuery 套件可以輔助製作美觀流暢的網頁介面，但若希望利用它來執行需要大量計算 (computationally intensive) 的應用，在執行的效率上還是會顯得不足。不巧的是，詞夾子演算法在計算時，必須多次掃瞄文本並進行大量的樣式比對。因此，在詞夾子元件的設計與實作上，我們需要特別關注元件的執行效能，並且在需要較長運算時間時，顯示執行的進度，以避免讓使用者誤以為系統停滯或當機。在 4.2 小節和第五節 5.4 小節，我們還會繼續討論到這個問題。

---

[9] DID-ACTE 是 Digging into Data: Automating Chinese Text Extraction 的縮寫。這個計畫是由 The Arts & Humanities Research Council (AHRC)、The National Endowment for the Humanities (NEH) 和 The Joint Information Systems Committee (JISC) 等機構所贊助。

[10] 詳情請參考 http://did3.jiscinvolve.org/wp/projects/chinese-texts。

[11] 根據（何浩洋 2014）的論文，採用 JavaScript 的原因是為了讓 MARKUS 在最少資源下，可永久提供服務為考量。

4.1 關於文本的章節與標點符號

篇幅稍大的中文古籍（例如《紅樓夢》、《西遊記》、《三國演義》等），通常都會將內容分放於多個章節。由於詞彙本身並不會跨章節出現，[12]上一章所介紹的簡化詞夾方法實際上並不需要用到章節的資訊。然而，人文學者在統計文本詞彙的研究過程中，有時並不僅關心「文本中出現哪些詞彙，又各出現幾次」，還會想瞭解「這些詞彙出現在哪些章節，它們爲什麼會出現在那些地方」。爲了讓使用者知道詞彙出現在哪個章節，我們將文本 τ 切分爲多個章節 $\tau_1, \tau_2, \ldots, \tau_n$（其中 τ_i 表示文本的第 i 個章節），並且用符號 $t \in \tau_i$ 表示詞彙 t 出現在 τ_i（t 出現在第 i 個章節）。

文本字彙 𝔸 通常包含有標點符號。雖然在中文的古籍裏並沒有通用的標點符號，[13]但在古籍的數位化過程中，通常會配置有標註的人力，將新式的標點符號加入文本的內容。由於標點符號提供了基本的斷句功能，讓詞彙本身不會跨兩個句子出現，因此詞彙擷取的演算法通常都會利用標點符號來增加整體的精準率 (precision) 與召回率 (recall)。詞夾子方法也不例外，在實務上經常可利用標點來提升擷取成效。然而，文本的標點有時會因語氣變化或語句是否需停頓而有異。[14]詞夾子元件提供一個選項，讓開發者決定是否要對文本的中文標點進行一致化的處理：將除了頓號（、）之外的各種中文標點，通通置換成不屬於字彙 𝔸 的特殊符號「⊥」。一致化的過程排除頓號，是因爲頓號經常被用來表示並列的詞語，而這些詞語有可能就是使用者想要擷取的詞彙。在標點一致化時保留頓號，可以讓系統利用這些頓號來取得連續（用頓號分隔）的一串詞彙。[15]

此外，對於某些特殊的文本，研究者想擷取的詞彙可能經常出現於每一行（或每個段落）的最前面。[16]詞夾子元件也提供一個選項，讓開發者決定是否要在每一

---

[12] 詞彙不會跨章節的意思，是說一個長度大於 1 的詞彙 s.t，其中 s, t 均不為空字串，並不會有「s 出現在前一個章節，t 出現在後一個章節」的情形。

[13] 參考維基百科 (https://zh.wikipedia.org) 對中文標點符號的說明。

[14] 例如，我們可能希望以「穿著」為左夾，它後面出現的標點為右夾，來擷取出現在兩者之間的詞彙。但試看《紅樓夢》以下幾個片段：

……身上穿著縷金百蝶穿花大紅洋緞窄褂襖，外罩五彩刻絲石青銀鼠褂；下著…… （第三回）

……蔥綠盤金彩繡綿裙，外面穿著青緞灰鼠褂。鳳姐笑道……（第五十一回）

……身上穿著金絲織的鎖子甲、洋錦襖袖；帶著倭刀…… （第五十二回）

可以發現，緊接在「穿著」後面的衣飾描述後，出現的標點可能有逗點、句號或頓號。

[15] 例如研究者可能想從《丁中丞政書》的以下內容

前經臣函飭臺灣道夏獻綸將臺地所產琉磺、磺油、樟腦、茶葉等項應如何擴充開辦之處……

擷取出「琉磺」、「磺油」、「樟腦」、「茶葉」等物產名稱。我們可利用詞夾「所產…等」先擷取出字串「琉磺、磺油、樟腦、茶葉」，再利用頓號將這四項物產從該字串中切分出來。

[16] 例如研究者可能想從《神農本草經》的以下內容

赤芝，味苦平。主胸中結，益心氣，補中，增慧智，不忘。久食，輕身不老……

黑芝，味咸平。主癰，利水道，益腎氣，通九竅，聰察。久食，輕身不老……

青芝，味酸平。主明目，補肝氣，安精魂，仁恕，久食，輕身不老延年神仙。一名龍芝。

中擷取出「赤芝」、「黑芝」和「青芝」等藥草名稱。這些名稱都位於於每個段落（在此就是每一次換行之後）的最前面。使用標點一致化並加上行段起始符號後的字彙 𝔹，我們就可利用詞夾「⊦…⊥」擷取出每一行起始到第一個標點之間的詞彙。

行的第一個文字之前加上另一個特殊符號「⊦」。最後，爲了簡化實作的複雜度，讓演算法能夠順利取得文本開頭和末尾的詞彙，我們也在文本的最前方和最後方加上足夠數量的貼補字元 (padding character)「⊙」。因此，經過標點一致化、加上行段起始符號 ⊦ 與貼補字元 ⊙ 之後，文本的字彙將會變成 $\mathbb{B} = (\mathbb{A} - \mathbb{P}) \cup \{\bot \vdash \odot\}$，其中 $\mathbb{P}$ 是除了頓號之外的所有中文標點字元。

若使用者想找的詞彙並不包含標點符號，我們建議將標點進行一致化的處理，如此可確保候選詞彙並不會包含有標點符號。[17]在標點一致化的模式下，詞夾子元件也會利用一些詞句的特性，盡量縮減可能的候選詞夾。[18]例如，詞夾 L..R 加上詞彙 t 所形成的字串 $L \bullet t \bullet R$，除了前後可以有標點，中間都不應有標點出現。因此，在標點一致化的模式下，若左夾 L 含有標點，詞夾子元件會刪除標點左方的所有字元（也就是說，標點會出現在左夾 L 的最左方）；類似地，若右夾 R 含有標點，元件也會刪除標點右方的所有字元。

4.2 元件在設計與實作上的一些考量

我們先討論文本大小的問題。設計詞夾子元件的時候，我們希望這個元件有能力處理數百萬字以上（例如《宋會要輯稿》、《明實錄》或《清實錄》等）的文本。然而，現實的問題是，若要處理這類文本，即使當前個人電腦的執行速度已經相當快，用 JavaScript 實作的程式仍常需頗長的時間才能完成擷詞函數（計算候選詞彙）或詞夾模具函數的計算（計算候選詞夾）。[19]因此，詞夾子元件在結構上會將較爲龐大的文本（在元件中稱之爲一項工作 task）切分爲較小文本（在元件中稱之爲子工作 subtasks），並允許將每個較小文本切分爲多個章節。這樣設計的目的，是讓元件就有機會以較小文本爲單位來進行批次計算。在這篇論文第五節所介紹的詞夾子系統，由於設定的目標是能夠流暢處理稍大（約數十到一兩百萬字，例如《西遊記》或《紅樓夢》）的文本，因此僅需將文本視爲單一的子工作來處理。

其次，第三節的簡化詞夾方法只用到兩種類型的詞夾（候選詞夾、種子詞夾）和兩種類型的詞彙（候選詞彙、種子詞彙）。實作上，我們需要考量到元件的效能和使用者操作的方式，因此還需加入其他類別的詞夾和詞彙。

尤其是，我們希望減少不必要的重覆計算。例如，若詞彙集合 $T = T_1 \cup T_2$，我們知道 $g(\tau, T, a, b) = g(\tau, T_1, a, b) \cup g(\tau, T_2, a, b)$。因此，若 T_1 是先前步驟就已

---

[17] 注意到，第三節的簡化詞夾子方法並沒有區隔一般文字與標點。在標點未一致化處理的模式下，詞夾子元件將允許擷取的詞彙包含有標點符號，也就有機會從《宋會要輯稿》文本捕捉到《歐陽修文集・皇第九女封福安公主制》和《[續]正元錄》這類關於出處來源的字串。

[18] 若使用者想擷取的詞彙並不包含標點，卻並未將標點進行一致化處理，擷取出來的候選詞夾和詞彙可能含有過多雜訊，也可能捕捉不到一些「利用標點一致化」可擷取出的詞彙。以第三節的文本 τ 為例，假定左右夾長度 a=b=4。在標點未一致化的模式下，詞彙「文彥博」將比對出詞夾『日，太師...言：「前』。這個詞夾會因為左夾的開頭有個「日」，右夾的末端有個「前」而很難再（從比這個範例文本大得多的正式文本）擷取出其他人名。另一方面，在標點一致化的模式下，計算出的詞夾會是『⊥太師...言⊥』，而有機會擷取出曾擔任太師職位的「蔡京」。

[19] 若採用 C/C++ 或 Java 這類程式語言來開發，效能有機會提升一到兩個數量級，但將失去 JavaScript 可在各類瀏覽器上運作、可充分應用 HTML/CSS 來提供互動式網頁的優點。

經使用過的種子詞彙，T_2 是使用者經由步驟 5 的結果所挑選出的新詞彙，那麼計算 $g(\tau, T, a, b)$ 時，因爲先前已經計算過 $g(\tau, T_1, a, b)$，實際上我們僅需計算 $g(\tau, T_2, a, b)$。爲了達到這樣的目的，實作時我們必須區隔出「先前步驟就已經使用過的種子詞彙」和「給後續計算使用的種子詞彙」。

類似地，若詞夾集合 $C = C_1 \cup C_2$， $f(\tau, C, m, n) = f(\tau, C_1, m, n) \cup f(\tau, C_2, m, n)$ 這個式子也永遠成立。因此若 C_1 是先前步驟就已經使用過的種子詞夾，C_2 是使用者從步驟 3 的結果所挑選出的新詞夾，我們也不需重覆計算 $f(\tau, C_1, m, n)$。

另外還有一個詞夾子方法本身的問題。當使用者操作詞夾子方法，進行幾次迭代後，可能會發現候選的詞夾數量暴增。當候選詞夾的數量超過數百、甚至高達數千個，要求使用者逐一看過每個詞夾，並從中找出較有效（較有可能夾中正確詞彙）的夾子，就顯得不符經濟效益。此外，許多時候並不容易從詞夾 L..R 的字面意義（例如「宰臣... 等」和「太師... 言」）推測這個詞夾是否有效。因此，在使用者從候選詞彙挑選種子時，系統最好能夠提供一些提示來幫助使用者進行挑選（參考第五節 5.3 小節）。

我們使用 JavaScript 內建的正規表示式 (regular expression) 功能來進行字串比對。對於稍長的文本（字數約數十到一兩百萬），當詞彙（或詞夾）數量增加到數百個以上，逐一對這些詞彙（或詞夾）進行文本比對會顯得很沒效率。因此，我們在實作詞夾子元件時，會盡量將多個詞彙（或詞夾）串接成單一的正規表示式，如此就可降低文本掃瞄 (scan) 的次數，增加系統的執行速度。

最後，使用者在實際操作詞夾子方法的過程中，可能會想更改詞夾長度和詞彙的長度限制。有時，他們還可能（例如從詞彙或詞夾的前後文）會發現一些新的種子詞彙（請參考第五節 5.5 最後兩段的例子）。詞夾子元件當然應該允許使用者利用這些發現來改善最後的擷取成效。

4.3 元件的基本使用方式

將詞夾子演算法封裝成一個元件 (component)，並提供使用這個元件的方法，可以讓其他開發者很容易地利用這個元件撰寫其他的應用程式。由於 MARKUS 是開發原始碼 (open source) 的系統，詞夾子元件自然也是開放原始碼，任何人都可從網路上取得這個元件（以及程式碼）來進行加值應用。在接下來的兩個小節，我們將假設讀者具有系統開發的背景或能力，懂得如何撰寫 HTML (HyperText Markup Language) 和 JavaScript 程式碼，也知道如何運用 jQuery 套件來處理網頁的事件 (events)。

詞夾子元件的程式碼寫在 clipper.js 檔案中。[20]我們用全域變數(global variable) `ChTermClipper` 將詞夾子演算法所需用到的資料與函式封裝起來。[21]系統開發者

[20] 元件程式碼的網址 http://dev.digital.ntu.edu.tw/DADH-2015/js/clipper.js。

[21] JavaScript 的撰寫風格建議（除了用作建構函式之外的）變數的第一個字母採用英文小寫字母。在此我們稍加延伸，建議會被廣泛使用的使用者自訂全域變數，第一個字母改採大寫字母。

引用 clipper.js 後，可以先用 `var Clipper = ChTermClipper` 定義另一個較簡短好記的廣域變數 `Clipper`，之後就可以直接利用 `Clipper` 來存取詞夾子元件的內容。對應於上一節所介紹的演算法，詞夾子元件至少需讓後續的應用程式能夠：

1. 輸入文本 τ。系統開發者可將文本儲存於一個陣列 `texts`，其中每個陣列的元素 `texts[i-1]` 對應到一個文本的章節 τ_i。[22]接著，開發者可呼叫詞夾子元件所提供的 `Clipper.setTexts(texts)` 方法，將文本內容放入元件。
2. 指定種子詞彙集合 $T=\{ t_1, t_2, \ldots, t_m\}$ 與種子詞夾集合 $C=\{c_1, c_2, \ldots, c_n\}$。開發者可將詞彙字串 $t_1, t_2, \ldots, t_m$ 放入陣列 `terms`（其中 `terms[i-1]` 對應到 t_i），然後利用元件的 `Clipper.setPositiveTerms(terms)` 方法將 T 放入元件。類似地，開發者可將詞夾字串 $c_1, c_2, \ldots, c_n$ 放入陣列 `clips`（其中 `clips[i-1]` 對應到 c_i），再呼叫 `Clipper.setPositiveClips(clips)` 將 C 置入元件。
3. 讓使用者設定起始的左右夾長度 a, b 和詞彙長度限制 m, n。這是透過 `Clipper.setParameters(…)` 函式來實現。
4. 提供詞夾模具函數 g(τ, T, a, b)，讓系統開發者可以利用種子詞彙 T 和左右夾長度 a 與 b，從文本 τ 中比對出候選詞夾。經過前三個步驟的設定，開發者可直接呼叫 `Clipper.computeCandidateClips()`，這個函式會利用 τ, T, a, b 的內容，回傳一個候選詞夾的陣列。
5. 提供多夾擷詞函數 f(τ, C, m, n)，讓開發者可利用種子詞夾 C 和詞彙的長度限制 m, n，從文本 τ 中擷取出候選詞彙。類似於詞夾模具函數，元件提供 `Clipper.computeCandidateTerms()` 方法，可讓開發者利用 τ, T, m, n 的內容，回傳一個候選詞彙的陣列。
6. 開發者可以透過變數 `Clipper.candidateTerms`, `Clipper.whiteTerms`, `Clipper.positiveTerms`, `Clipper.negativeTerms` 來取得或設定元件的候選詞彙、已運算過的種子詞彙、尚未用來計算詞夾的種子詞彙、使用者欲丟棄的錯誤詞彙（這些變數的資料型態都是詞彙字串的陣列）。類似地，使用者可透過變數 `Clipper.candidateClips`, `Clipper.whiteClips`, `Clipper.positiveClips`, `Clipper.negativeClips` 來取得或設定元件的候選詞夾、已運算過的種子詞夾、尚未用來計算詞彙的種子詞夾、使用者認爲無用而欲丟棄的錯誤詞夾（這些變數的資料型態都是詞夾字串的陣列）。

4.4 一個簡單但完整的元件使用範例

我們在網站上提供了一些例子[23]，示範如何利用詞夾子元件開發簡單的應用程式。例如，以下就是一份簡單但完整的範例程式碼：[24]

---

[22] τ_i 的 i 是從 1 開始計數，而陣列 `texts[i]` 的索引 i 則是從 0 開始計數。

[23] 請參考網頁 http://dev.digital.ntu.edu.tw/did-acte/demos.html

[24] 此範例程式碼的網址 http:// dev.digital.ntu.edu.tw/Harvard-Did/clipper-simple-demo.html

```
<!doctype html>
<html>
<head>
   <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
   <title>詞夾子展示</title>
   <script src="js/jquery.min.js"></script>
<!doctype html>
<html>
<head>
   <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
   <title>詞夾子展示</title>
   <script src="js/jquery.min.js"></script>
   <script src="js/clipper.js"></script>
   <style type="text/css">
      .divText { background-color:#DFDFEF; padding:5px; margin:3px;
                 border:2px solid #5F5F8F; border-radius:4px; }
   </style>
</head>
<body style="background-color:#EFEFEF">
   <h3>詞夾子的基本應用</h3>
   <table>
      <tr>
        <td valign="top">文本：</td>
        <td><textarea id="inTexts" rows="5" cols="80">請貼上文本的內容</textarea></td>
      </tr>
      <tr>
         <td valign="top">種子詞彙：</td>
         <td><textarea id="inPositiveTerms" rows="3" cols="80">請輸入種子詞彙（用逗號區隔）
         </textarea></td>
      </tr>
     <tr>
        <td colspan="2">
          <button type="button" id="butSubmit">計算</button>
        </td>
        </tr>
   </table>
   <p/>
   <table width="100%">
      <tr><td width="80" align="right"><nobr><b>比對出的詞夾:</b></nobr></td>
          <td width="60%"><div id="divClipsResult" class="divText">詞夾輸出</div></td>
          <td width="30%"></td></tr>
      <tr><td align="right"><nobr><b>擷取出的詞彙:</b></nobr></td>
           <td><div id="divTermsResult" class="divText">詞彙輸出</div></td>
           <td width="30%"></td></tr>
   </table>
</body>
<script language="javascript">
   var Clipper = ChTermClipper;
   document.title = Clipper.package + ' v' + Clipper.version + ': ' + document.title;

   $("#butSubmit").click(function() { computeTerms(); });

   function computeTerms() {
      Clipper.resetClipState().setParameters({maxRightClipChars:1});

      var texts = $("#inTexts").val().replace(/(\r*\n)+/g,"\n").split("\n");
      Clipper.setTexts(texts);
      Clipper.setPositiveTerms($("#inPositiveTerms").val().trim().split(","));

      Clipper.computeCandidateClips();
```

```
      $("#divClipsResult").html(Clipper.candidateClips.join(","));
      Clipper.moveAllCandidateClipsToPositive();

      Clipper.computeCandidateTerms();
     $("#divTermsResult").html(Clipper.candidateTerms.join(","));
   }
</script>
</html>
```

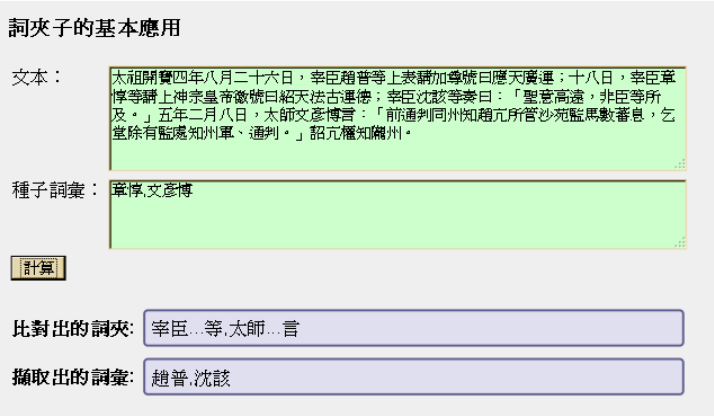
這個範例程式的網頁呈現如下：

這份程式碼並不算太長。程式碼的前半部分，要求瀏覽器載入 jquery.min.js 和詞夾子元件 clipper.js，接著就利用 HTML 在網頁上繪出讓使用者輸入文本和種子詞彙的表單。後半段 <script>標籤內的程式碼，則主要在處理使用者（在文本區塊貼上內容，並輸入種子詞彙後）按下「計算」按鈕後的動作。我們將程式中用到詞夾子元件的部分解釋如下：

- `var Clipper = ChTermClipper` 將詞夾子物件指定到 `Clipper` 全域變數。
- `Clipper.resetClipState()` 重設詞夾狀態：它會將左右夾長度 a, b 都設爲 2，而將詞彙的長度限制 m, n 設在 m=2, n=3。它也會將元件內的種子詞夾、候選詞夾、種子詞彙、候選詞彙等等都清空。
- `Clipper.setParameters({maxRightClipChars:1})` 設定詞夾子元件的參數，將右夾長度改設爲 1。[25]
- `Clipper.setTexts(…)` 設定文本內容。
- `Clipper.setPositiveTerms(…)` 設定種子詞彙。
- `Clipper.computeCandidateClips()` 計算出候選詞彙。
- `Clipper.moveAllCandidateClipsToPositive()` 將所有的候選詞彙搬移到種子詞彙。實務上，由於詞彙需經過使用者挑選，這個函式的用途並不大。
- `Clipper.computeCandidateClips()` 利用種子詞彙計算出候選詞夾。

[25] 大部分的詞夾子元件函式呼叫都採用 JavaScript 的 method chaining pattern。這可以讓此開發者在程式碼上用串接的方式 `Clipper.resetClipState().setParameters(…)` 取代連續的函式呼叫 `Clipper.resetClipState()` 和 `Clipper.setParameters(…)`。

以上的範例程式，其實就是實作了 f(τ,g(τ, T, 2, 1), 2, 3) 這個函數，其中 τ, T 爲使用者輸入的文本與種子詞彙。當使用者按下「計算」鍵，程式會計算出 g(τ, T, 2, 1)，並把這份結果輸出到「比對出的詞夾」區域，接著計算 f(τ,g(τ, T, 2, 1), 2, 3)-T 並將這些「新發現的詞彙」放到「擷取出的詞彙」區塊中顯示。例如，輸入我們在第三節介紹演算法所用的測試文本，並以「章惇, 文彥博」作爲種子詞彙，按下「計算」鈕後，可以得到以下的畫面：



也就是說，這個簡單的範例程式，可做到我們在第三節所介紹的例子：使用者只需提供「章惇, 文彥博」，就能從文本中擷取出另外兩個人名「趙普」和「沈該」。

這份簡單的範例程式，目的是讓系統開發者了解如何引用詞夾子元件，並以論文第三節的例子來進行實際測試。當然，使用者也可以利用這個簡單的網頁，輸入不同的文本和種子詞彙，觀察並理解詞夾子方法的運作過程與優缺點。

第五節　JavaScript 詞夾子系統與操作實例

在第四節 4.4 小節所介紹的範例程式太過簡單，在實務上應用的效果有限。我們利用詞夾子元件開發一個較爲完整的系統，[26]讓使用者可以貼上自己的文本，利用詞夾子方法擷取想要的詞彙，並將找出的詞彙存到瀏覽器所處的硬碟。我們希望這個系統能夠流暢地處理稍大（數十萬到一兩百萬字）的文本。接下來幾個小節，我們將以《紅樓夢》作爲文本來實際操作這個系統。[27]在介紹這個工具的過程中，我們也會討論到使用介面上的設計考量。

雖然簡化的詞夾子方法符合直觀，但要設計出友善的使用介面卻並不容易。

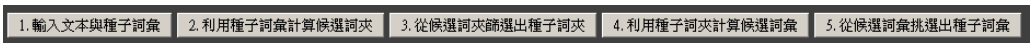
---

[26] 我們選擇瀏覽器 Firefox 作為展示所用的瀏覽器，主要原因是因為它是我們在開發與測試時所用的瀏覽器。這個詞夾子工具（以及詞夾子元件）在 Chrome 也能運作，但在 IE 11 的瀏覽器上則有部分（例如「載入進度」）功能無法運作。

[27] 詞夾子方法相當合適於從《清實錄》或《宋會要輯稿》擷取人名詞彙。但因這類文本較為龐大，系統處理時會需要較長的等待時間（參考第四節 4.2 小節，若文本字數超過兩百萬，我們就會建議將整份文本切分為多項子工作來分批計算）。我們在此採用章回小說來作例子，部分原因是它的字數不到百萬，另一個原因是想強調：應由使用者因需求來認定詞彙是否相關（參考第五節 5.4 小節）。

對於系統設計者而言，可加入的功能太多，可讓使用者調整的參數也不少。要如何簡化操作介面，卻又不失系統的可用性，是一件相當有挑戰性的工作。

最單純的想法，是直接將第三節的演算法對應到系統的使用介面。我們可以將該演算法中，牽涉到使用者篩選或決定的每個步驟都對應到一個按鈕，這樣使用者就可依照演算法，逐次點選按鈕來操作這些步驟：



也就是說，使用者先點選第一個按鍵來輸入文本與種子詞彙，然後點選第二個按鍵算出候選詞夾，接著點選第三個按鍵從候選詞夾中篩選出種子詞夾，然後點選第四個按鍵來算出候選詞彙，接著點選第五個案件從候選詞彙中找出新的種子詞彙，然後再點選第二個按鍵利用新的種子詞彙計算詞夾……。然而，我們發現第二個按鍵和第四個按鍵的動作，其實只是要求系統依照前一個步驟的種子（詞夾或詞彙）來計算接下來的候選項目（詞彙或詞夾），因此可將這兩個步驟整併到它們的前一個步驟中。如此頁面上就可減少兩個按鍵，降低使用介面的複雜度。

5.1 啟用詞夾子系統

從瀏覽器直接輸入網址，就可啓動詞夾子系統。[28]系統載入後頁面的截圖如下：

詞彙擷取的過程中，使用者主要是利用右上方的三個按鈕與系統互動。這三個按鈕由左而右分別爲「輸入文本與種子」、「詞夾→詞彙」與「詞彙→詞夾」。大致上，「輸入文本與種子」對應到第三節詞夾演算法的步驟 1 與步驟 2（讓使用者輸入文本與種子詞彙，並算出候選詞夾），「詞夾→詞彙」對應到步驟 3, 4（讓使用者輸入或挑選詞夾來擷取新的詞彙），「詞彙→詞夾」對應到步驟 5 和步驟 2（讓使用者從候選詞彙中，挑選出新的種子詞彙，並據以計算出新的詞夾）。

[28] 詞夾子系統的網址 http://dev.digital.ntu.edu.tw/DADH-2015。

在這三個按鈕的右邊，還有一個齒輪長相的圖示（代表「設定」的意思），點選這個圖示可以讓使用者更改詞夾模具的長度、指定是否要將標點一致化，或者進行擷詞進度的存取。我們將在 第五節 5.5 和 第五節 5.6 小節介紹這些功能。

在右上方的按鈕下，系統會顯示當前的詞夾參數和狀態。第一項是詞夾模具的四個參數 a[m-n]b，其中 a 代表左夾長度， m-n 表示擷取的詞彙長度必須介於 m, n 之間，而 b 代表右夾長度。上圖頁面顯示的 2[2-3]2 是系統預設的詞夾模具參數：左右夾的長度都是 2，欲擷取的詞彙長度則是在 2 到 3 之間。[29]模具參數的右邊項目，則是說明系統是否需對文本進行標點一致化的處理（預設爲「是」）。[30] 這兩項參數的下一行則列出目前系統的候選（詞）夾數量、候選詞彙數量、以及已經擷取出的正確詞彙數量。列出這些參數和狀態值，可以讓使用者對於「目前擷詞程序已進行到什麼程度」有些掌握。

最後，系統參數和狀態的下方，則是一大片顯示文本內容的區域。我們認爲，人文研究者在使用詞夾子擷取詞彙時，應也會想參看文本的內容。由於需要用到擷詞功能的文本，其內容應不會太短，因此我們在此刻意留下大面積的顯示空間。

5.2 文本與種子詞彙的輸入

操作詞夾子方法的第一個步驟，就是輸入文本和種子詞彙。按下「輸入文本與種子」鍵，系統會顯示一個對話方塊，讓使用者貼上文本內容，並且輸入起始的種子詞彙。在此遇到的設計問題是，《紅樓夢》共 120 章回，要如何讓使用者一次性地貼上完整的內容，卻又仍然保留章回的結構？用第四節 4.1 的話來說，就是該如何一次性地輸入文本 τ，然後自動將其切分爲多個章節 $\tau_1, \tau_2, \ldots, \tau_n$？

我們的做法是，請使用者在章節間，自行加上連續四個（或四個以上）等號來分隔前後的章節。人文研究者通常有自己的文本。若這些內容本身就有分章節，他們通常會將每個章節獨自儲存在一個檔案裡。我們認爲現今的使用者有能力運用他們所熟悉的的編輯器 (editor) 來串接所需的章節，並且在連續的兩個章節之間加上四個以上的等號。我們在展示網站上提供了一些文本的樣本[31]，使用者可以直接拷貝、貼上這些文本內容，來練習使用詞夾子系統。展示的網站上另提供一個線上的小工具，讓使用者將多份文字檔串接成一個大檔案（這個工具會在文字檔之間加上四個等號，方便詞夾子系統將其識別爲不同章節）。

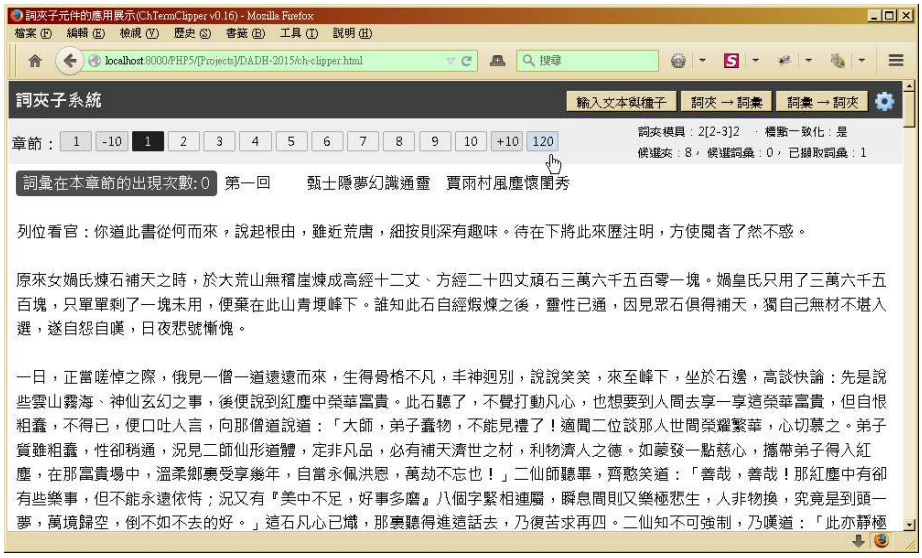
---

[29] 對中文的文本而言，通常左右夾的長度設在 2 最合適。左右夾長度過短，容易擷取到錯誤的詞彙，過長則容易產生過多候選詞夾。此外，因為中文詞彙的長度多半介於 2 和 3 之間，因此系統將 m=2 與 n=3。若 m 過小或 n 過大，計算出的候選詞夾和詞彙都可能包含過多雜訊。

[30] 我們在這一節所舉的例子，因想找的詞彙都不會包含標點，因此都是採用標點一致化的模式來進行擷詞程序（參考腳註 18）。

[31] 網址: http://dev.digital.ntu.edu.tw/DADH-2015。文本的樣本目前包含有《紅樓夢》、《西遊記》、《水滸傳》和《三國演義》。

上圖是貼上《紅樓夢》文本內容、填上「賈寶玉」作爲種子詞彙、而詞夾模具的參數設爲 2[2-3]2 的截圖。注意到，文本在第二回的前面有連續四個等號，這樣可以讓系統知道：等號前和等號之後分屬不同的章節。按下「確定（計算詞夾）」鍵後，系統會利用輸入的種子詞彙計算詞夾：



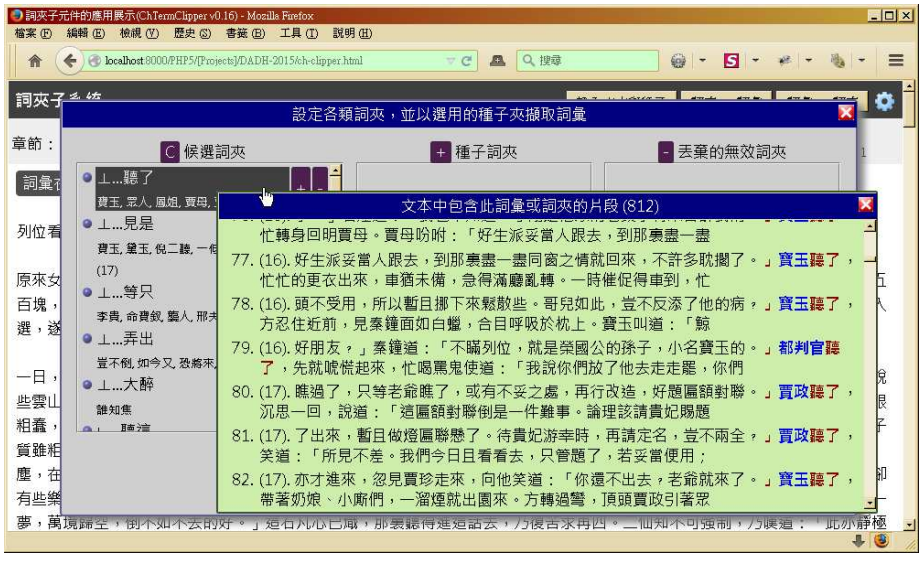
從頁面上可以看到，左上方的「章節」後顯示出一串連結，讓使用者可以直接點選連結來檢視章節內容。注意到，最後面的連結顯示 120，表示這份文本共有 120 個章節。另外，右上方的狀態區顯示出，利用「賈寶玉」作爲種子詞彙，目前共得到 8 個候選（詞）夾，0 個候選詞彙，以及 1 個已擷取詞彙（因使用者輸入的種子「賈寶玉」本身就是使用者想要的詞彙之一）。

5.3 利用種子詞夾計算候選詞彙

上個步驟中，我們使用「賈寶玉」作爲種子詞彙，比對出 8 個候選的詞夾。接下來我們想從候選的詞夾中，篩選出種子詞夾，並利用種子詞夾擷取文本詞彙。

點選頁面右上方的「詞夾→詞彙」按鈕，系統會彈出一個對話方塊：

對話方塊左方顯示目前的 8 個候選詞夾。這份截圖的游標位置停在第一個詞夾「⊥... 聽了」上。該詞夾的左夾是一個標點符號（一致化後的符號⊥），右夾是「聽了」這兩個字。在詞夾的下方，系統會提示這個詞夾可夾中的詞彙中，尚未被使用者分類（被挑選爲種子或丟棄）者。例如，上圖就告訴使用者，詞夾「⊥... 聽了」至少可以擷取出「寶玉」、「眾人」、「鳳姐」、「賈母」和「賈政」這些詞彙。若使用者想更進一步，查看這個詞夾出現的前後文，可點選這個詞夾來顯示如下的畫面：



從上圖中可看到，詞夾「⊥... 聽了」在文本中出現了 812 次（提醒一下，夾中的詞彙長度需介於 2 和 3 之間）。其中，第 79 個比對到的詞彙（都判官）出現在第 16 回，第 80 和第 81 個詞彙（賈政）出現在第 17 回，而第 82 個比對到的詞彙（寶玉）也出現在第 17 回。

這裏我們岔開系統的使用說明，討論一下詞夾子元件對於上圖列表輸出的支

援。我們認爲，在挑選候選詞夾的過程中，提示該詞夾可擷取到的詞彙，對於使用者判斷某詞夾是否有效，是相當有幫助的。例如詞夾「⊥...大醉」表面上看起來應該可以夾中許多人名（在日常的用語中，我們似乎也頗常看見「某某某大醉」，其中某某某是個人名），但實際上這個詞夾在《紅樓夢》中僅出現兩次，且除了使用者提供的「賈寶玉」之外，另一個擷取到的「誰知焦」明顯不是個人名（也就是說，對《紅樓夢》而言，若已知「賈寶玉」是個人名，「⊥...大醉」對於擷取人名詞彙就是個無效的詞夾）。[32]此外，有時使用者也可能需要類似上圖顯示的詞夾前後文，才能判斷某個詞夾是否有效。爲了減輕開發者的負擔，詞夾子元件提供一個函式 `getItemContextHtml(item,classNames,contextLen,section)` 讓開發者呼叫。開發者只需提供 `item`（詞夾或詞彙的字串）、`classNames`（使用者可自行定義詞夾和詞彙該如何顯示，例如上圖的詞夾是以暗紅色顯示，而詞彙則以藍色顯示）、`contextLen`（詞夾左右的前後文字串長度）、`section`（是否要顯示章節數字的真假值旗標），就可以透過這個函式取得類似上圖列表的輸出。[33]

接下來回到系統的使用說明。看過上圖，我們知道「⊥... 聽了」可以擷取出許多人名詞彙，因此我們想將它設爲種子詞夾。點選「⊥... 聽了」右方的「+」符號，就可將選到的項目從候選詞夾移到種子詞夾（對應於第三節演算法的步驟3）。我們將移動後的畫面截圖如下：[34]

最後，按下「確定（計算詞彙）」鈕，系統會利用這個種子詞彙比對文本，然

---

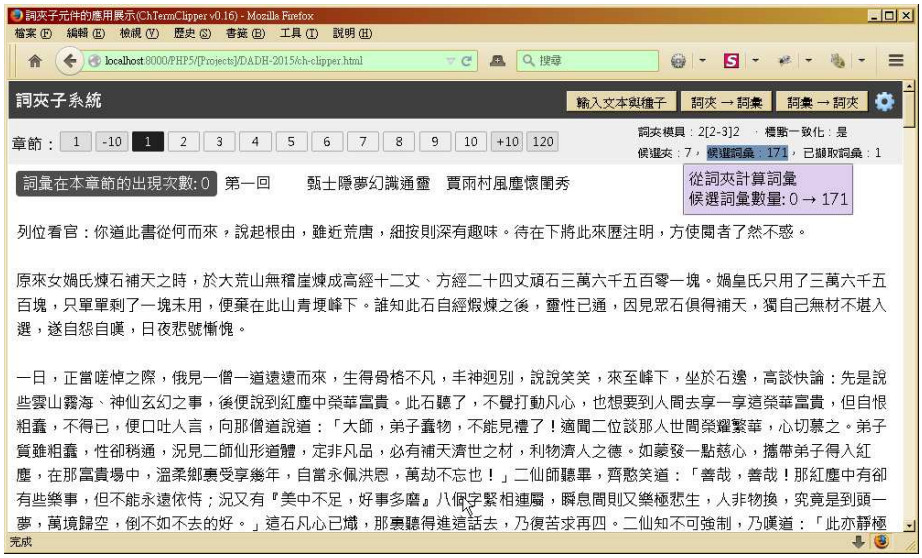
[32] 詞夾「⊥...大醉」所比對到的兩段文字如下：

- 媳婦們回說：「外頭派了焦大，誰知焦大醉了，又罵呢。」（第七回內文）
- 第八回　薛寶釵小恙梨香院　賈寶玉大醉絳芸軒（第八回標題）

[33] 因為元件的目的在於實作詞夾子演算法（而非與網頁呈現相關的內容），依照軟體工程方法論的 MVC 框架 (Model-View-Controller framework)，並不適合在元件中提供與網頁呈現相關的內容。然而，為了簡化回傳的結果並減少應用程式的計算，我們還是讓元件回傳 HTML 程式碼。使用者可傳入參數 `className`，利用 CSS (Cascading Style Sheets) 來改變網頁呈現的樣貌。

[34] 使用者在操作的過程中，可（利用「新增詞夾」的輸入欄位）隨時加入新的種子詞夾，也可（利用「新增詞彙」的輸入欄位）加入新的種子詞彙。

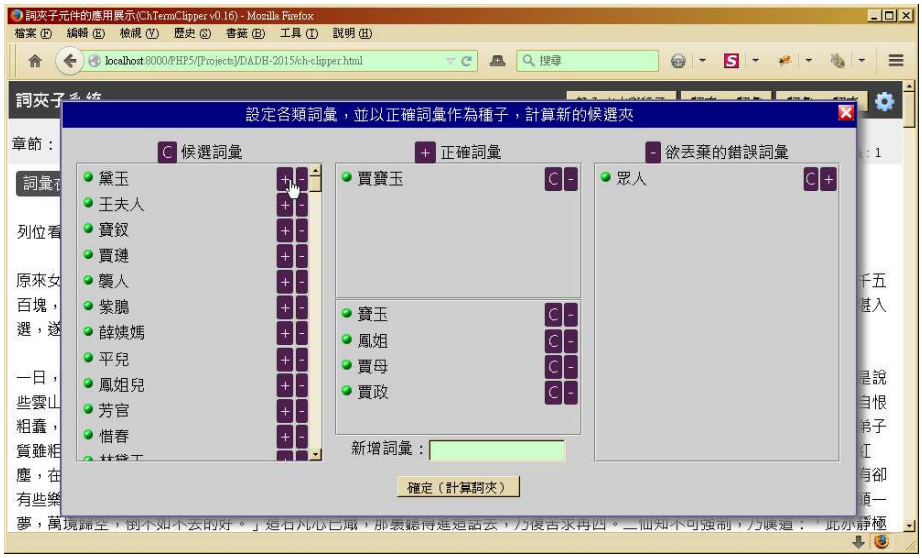
後擷取出 171 個候選詞彙（對應於第三節演算法的步驟 4）。系統介面在計算完成後，會強調顯示候選詞彙在數量上的變化（尚未計算前的數量是 0，計算後的數量是 171）：



5.4 利用種子詞彙計算候選詞夾

接下來，我們要從上一節所擷取到的 171 個候選詞彙中，挑選出新的種子詞彙（對應於第三節演算法的步驟 5）。按下右上方的「詞彙→詞夾」鈕並參考下圖。我們可點選項目旁的「＋」符號，將明顯代表某人物的詞彙（例如寶玉、鳳姐、賈母、賈政等）移到「正確詞彙」；點選項目旁的「－」符號，可將明顯非指特定人物的詞彙（例如眾人）移到「欲丟棄的錯誤詞彙」：[35]

[35] 這裏就用到第四節 4.2 的概念：在「正確詞彙」下列有兩個方框，上面的方框（目前只有「賈寶玉」一個項目）表示「先前步驟就已經使用到的種子詞彙」，而下面的方框則代表「給後續計算使用的種子詞彙」。此外，我們還加入「欲丟棄的錯誤詞彙」，這樣可以避免某些經常出現的錯誤詞彙（例如「眾人」）再次被演算法放入候選詞彙。



從上圖中，我們可以發現詞夾「⊥...聽了」所擷取到的 171 個候選詞彙中，已包含寶玉、鳳姐、賈母、賈政、黛玉、王夫人、寶釵、賈璉、襲人等許多《紅樓夢》的人名詞彙。這顯示出詞夾子方法的威力：如果種子詞夾挑選得當，可以很快就擷取到相當數量的正確詞彙。

在挑選正確詞彙的過程中，往往也會遇到一些不是很容易判斷的狀況。例如，「鳳姐」和「鳳姐兒」，以及「寶玉」和「賈寶玉」都指稱到相同的人物，但兩者是否都算是研究者想要找的詞彙，就顯得見仁見智。又如，詞彙「石頭」應不算人名，但在《紅樓夢》故事的開端，這「石頭」卻也是一個能聽能說的擬人物件：

> .....然後好攜你到那昌明隆盛之邦、詩禮簪纓之族、花柳繁華地、溫柔富貴鄉去安身樂業。」石頭聽了，喜不能禁，乃問：「不知.....

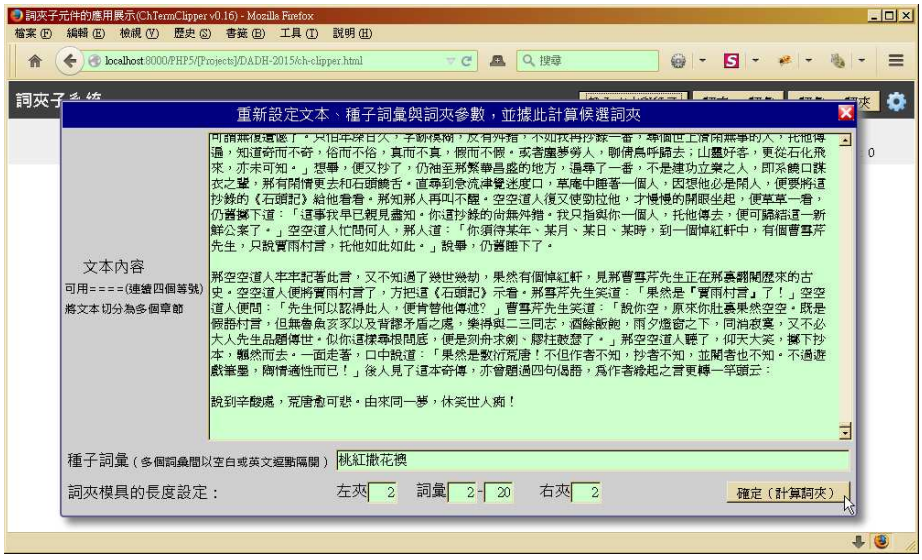
另外，像是「婆子」、「小丫頭」、「老嬤嬤」、「老耗」、「小耗」這類並非指稱特定人物的詞彙，算不算是正確的詞彙，也都需由使用者自行決定。

按下「確定（計算詞夾）」後，系統就會利用使用者挑選的種子詞彙計算詞夾。當種子詞彙或詞夾的數量增加，有時系統會需要較長的時間來計算候選詞夾或詞彙。在這種狀況下，Firefox 瀏覽器會顯示如下的等待頁面：[36]

[36] 我們選擇瀏覽器 Firefox 作為展示所用的瀏覽器，主要原因在於它是我們在開發與測試時所用的瀏覽器。這個詞夾子工具（以及詞夾子元件）在 Chrome 瀏覽器上也能運作，但在 Internet Explorer 11 瀏覽器上則有部分（例如「載入進度」）功能失常。此外，由於 Chrome 與 Internet Explorer 11 在忙碌執行 JavaScript 程式碼時，似乎並不會抽空更新頁面的內容，因此這兩個瀏覽器在元件進行計算時，頁面並不會顯示執行的進度，而是呈現停滯的狀態，

5.5 詞夾參數設定與詞彙在文本中的顯示

接下來的例子，我們的目標將跳離人物名稱的擷取，而改嘗試從《紅樓夢》找尋衣飾的名稱。因該小說中會出現「縷金百蝶穿花大紅洋緞窄褂襖」、「荔色哆羅呢的天馬箭袖」之類的長詞彙，我們將左右夾和最短詞彙的長度保持在 2，而最長詞彙的長度則設爲 20。按下頁面上方「輸入文本與種子」的按鈕，再次貼上《紅樓夢》文本內容，但這回將種子詞彙改設爲「桃紅撒花襖」，並將詞夾模具的參數設爲 2[2-20]2：



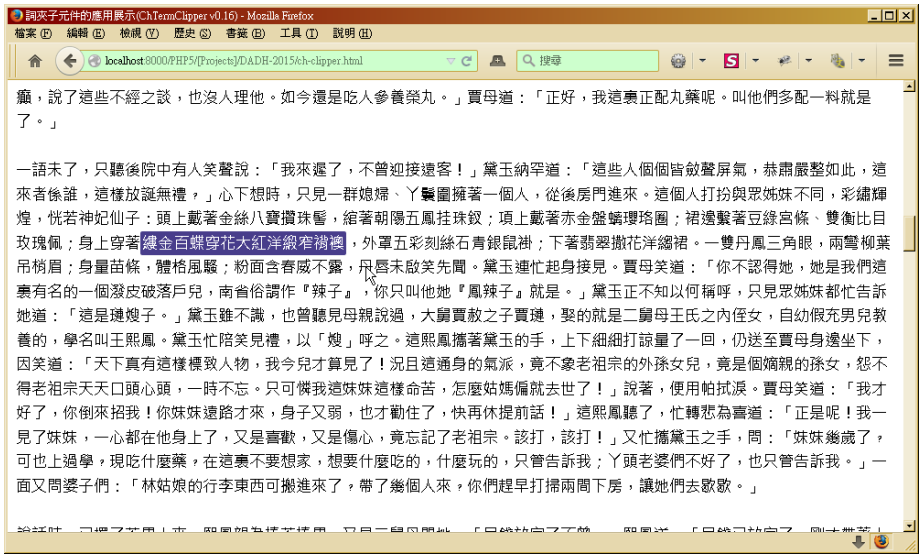
按下「確定（計算詞彙）」後，系統將清除先前的詞夾和詞彙，重設文本與種子詞彙，並利用它們計算候選詞夾。結果是，利用「桃紅撒花襖」作爲種子，系統只比對出一個候選詞夾「穿著...⊥」。利用這個詞夾作爲新的種子，系統可以擷取到 55 個候選詞彙：

從上圖可以看見，我們僅操作幾個步驟，就可以利用詞夾子方法找出「縷金百蝶穿花大紅洋緞窄褙襖」、「銀紅撒花半舊大襖」、「秋香色立蟒白狐腋箭袖」、「江牙海水五爪坐龍白蟒袍」等服飾名稱。由於這些詞彙在文本中僅出現一次，運用統計特性的多用途 (general purposed) 擷詞方法，通常無法成功計算出這類詞彙。

在上圖中我們也可以看到詞彙「家常衣服」和「幾件半新不舊的衣裳」，它們並不算是特定衣飾的名稱。然而，若使用者想發掘的是《紅樓夢》中對各種穿在人們身上衣飾的描述，它們仍有可能屬於使用者想要的詞彙。

爲了方便接下來的說明，我們暫且僅將「縷金百蝶穿花大紅洋緞窄褙襖」和「銀紅撒花半舊大襖」當作正確詞彙（移到種子詞彙區），然後按下「確定」。在系統的顯示頁面上，我們點選章節 3：

從上面的截圖可以看到，第三回「詞彙在本章節的出現次數」爲 2。這是因爲目前的正確詞彙只有「桃紅撒花襖」、「縷金百蝶穿花大紅洋緞窄褙襖」和「銀紅撒花半舊大襖」，而它們在第三回共出現 2 次。捲動頁面可看到如下的截圖：



這份截圖顯示出，系統會以反白來強調已擷取到的詞彙。檢視詞彙「縷金百蝶穿花大紅洋緞窄褂襖」的前後文，我們可以發現「金絲八寶攢珠髻」、「朝陽五鳳挂珠釵」、「赤金盤螭瓔珞圈」、「豆綠宮條」、「雙衡比目玫瑰佩」、以及「五彩刻絲石青銀鼠褂」和「翡翠撒花洋縐裙」也都是衣飾的名稱。這表示使用者可藉由內容檢視，從文本中再提取許多想要的詞彙。[37]使用者只需在文本上選取一小段文字，系統會跳出一個詢問視窗，確認是否要將該文字加入種子詞彙。[38]

在文本檢視過程中，使用者應也不難注意到，曹雪芹在王熙鳳出場時的衣著描繪順序。這個順序是由上而下，由內而外，從頭上、項上、裙邊、身上、外罩、一直形容到下著。這些觀察雖然不屬於詞彙擷取的範疇，但由於人文研究者經常需要細細體會文本的意涵，這些「額外」的觀察或發現對他們可能也很重要。這也是我們在設計這個詞夾子系統時，再三強調內文檢視的重要性，並盡可能保留較大內文顯示區域的原因。

5.6 進度存取與詞彙輸出

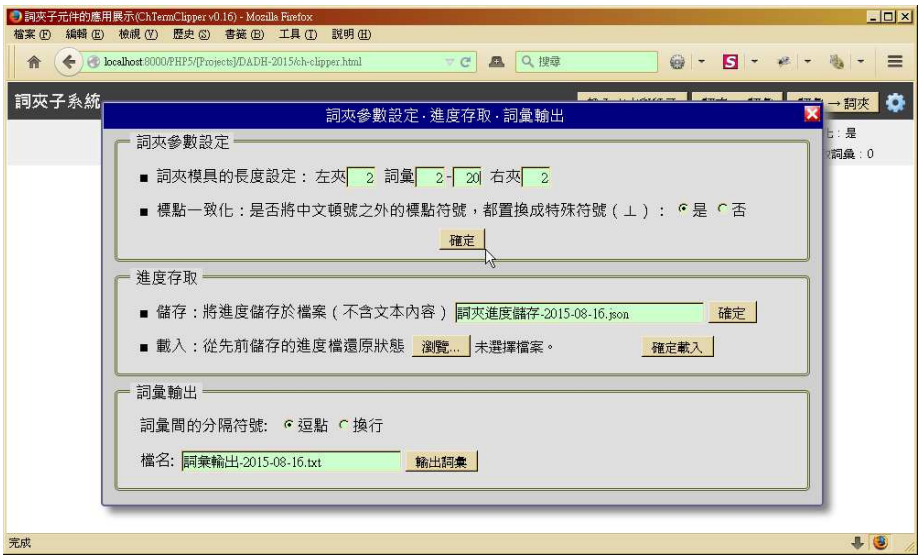
若想從文本中盡量擷取出所有想要找的詞彙，使用者可能需要操作多次迭代，並從包含上百或成千個項目的候選清單中，篩選出合適的詞夾，並挑選出想要保留的詞彙。爲了讓這個詞夾子系統更具實用性，我們加上第三節演算法所沒有提及的「進度存取」和「詞彙輸出」功能。

---

[37] 詞夾子方法的目的，是幫助使用者在較短的時間內找出想要的詞彙。這個方法並沒有保證能夠擷取出所有使用者想要的詞彙。我們在這個詞夾子系統的設計上，保留了相當大的顯示區域讓使用者可以對文本進行檢視。由於已擷取出的詞彙會被強化顯示，因此使用者可以很快地找到這些詞彙出現的地方，並從文本檢視的過程中，發現一些詞夾子方法可能遺漏的詞彙。

[38] 在較早期的版本，使用者必須透過頁面右上方的「詞彙→詞夾」按鈕，才能以「新增詞彙」的方式輸入新的種子詞彙。系統從 v0.18（元件版本 v0.17）之後，加入從文本顯示區直接框選一段文字來新增詞彙的功能。

按下頁面右上方的齒輪圖示，系統會呈現如下的頁面：



這個對話方塊中有三個小區塊，分別是「詞夾參數設定」、「進度存取」與「詞彙輸出」。最上方的「詞夾參數設定」可讓使用者在操作詞夾子方法的過程中，更改詞夾模具的長度設定。「進度存取」則可將目前的詞夾參數設定（包含左右夾長度、欲夾中詞彙的長度限制、是否標點一致化）、系統詞夾狀態（包含候選詞夾、已使用過的種子詞夾、尚未使用的種子詞夾、丟棄的詞夾）、與系統詞彙狀態（包含候選詞彙、已使用過的種子詞彙、尚未使用的種子詞彙、丟棄的詞彙）以 JSON (JavaScript Object Notation) 格式儲存在瀏覽器所處的硬碟中。使用者可以更改系統預設的儲存檔名，以儲存多組擷取進度。使用者可從「進度存取」的「載入」行，按下「瀏覽」鈕從硬碟中選取先前儲存的進度檔，再按下「確定載入」來還原先前的進度。必須注意的是，進度儲存與載入並不包含文本的內容。使用者若想更換文本的內容，必須先經過 5.2 小節「文本與種子詞彙輸入」的步驟，然後再載入進度。

詞夾子方法的目的，是幫助使用者從文本中擷取出想要找的特定類別詞彙。對話方塊最下方的「詞彙輸出」功能，可將系統目前所擷取到的正確詞彙（也就是已使用或未使用的種子詞彙）儲存在硬碟上。使用者可以選擇要以逗號（，）或換行來區隔輸出的詞彙。詞彙輸出後，日後也可以拿它們來當作起始的種子詞彙，透過 5.2 小節「文本與種子詞彙輸入」的步驟來繼續使用這個詞夾子系統。

第六節　結論與展望

在這篇論文中，我們介紹了簡化的詞夾子方法，並利用這個演算法實作了一個 JavaScript 元件。我們討論這個元件在設計與實作上的一些問題，並利用它開發了一套完整的詞夾子應用系統。我們花了相當長的篇幅，介紹如何使用這個詞夾子系統，並從系統操作中展示詞夾子方法的擷詞能力。

詞夾子方法的優點，是一旦找到有效的詞夾，就可以很快擷取出相當數量的

正確詞彙。這個方法的潛在缺點，是經過數次迭代後，候選的詞夾可能高達數千個，很難用人力逐一進行篩選。爲了減輕使用者挑選詞夾的負擔，詞夾子元件從種子詞彙計算候選詞夾時，會進一步計算候選詞夾可夾中的詞彙，如此就可顯示「這個詞夾可擷取到，但尚未被分類的詞彙」，方便使用者快速篩選詞夾。

在這篇論文中，我們展示如何利用詞夾子系統，從古典小說《紅樓夢》初步擷取人物和衣飾名稱。由於詞夾子方法的擷取成效，高度依賴於文本的文字敘述結構，在實務上如何利用詞夾子系統來有效擷取詞彙，不同的文本類型應該會偏好不同的詞夾參數、種子詞彙和詞夾篩選策略。也由於我們相信實務上不應僅僅利用詞夾子方法來擷取詞彙，因此在詞夾子系統的設計上強調文本檢視（並從檢視中發現相關詞彙）的重要性。我們希望能有數位人文的研究者，實際應用這個系統來擷取研究所需的詞彙，並與大家分享擷詞過程的感想或心得。這些使用的經驗，將可對系統接下來的改進方向產生相當大的助益。

另一方面，由於人文研究者多半缺乏堅實的資訊工程背景，我們也希望這篇論文能夠對數位人文的工具開發有所貢獻。除了用來開發論文所提的詞夾子系統，我們的 JavaScript 元件也已經被 MARKUS 標記工具所使用。這證實軟體工程中，基於元件的開發方法 (component-based software engineering) 對於數位人文系統也一樣有效。將演算法包裝成元件，可以減輕開發者的負擔（MARKUS 開發者不必自己實作詞夾子方法），進而降低開發複雜系統的困難度。

我們希望，會有更多研究者投入數位人文工具的開發，並且將這些研究成果包裝成可以重覆使用的元件。我們也期待，有善於人機介面 (human-computer interaction) 的系統開發者，能夠妥善應用這些元件，開發出適合人文研究者使用的工具系統。

參考文獻

張尚斌，2006 年。〈詞夾子演算法在專有名詞辨識上的應用 – 以歷史文件爲例〉。臺北：國立臺灣大學資訊工程學研究所碩士論文。

謝育平、楊龍廉、趙建宏、黃銘立、古馮文、林郁智，2009 年。〈使用詞夾子建立中文典籍分析加值服務〉，銘傳大學 2009 資訊科技與實務研討會。

謝育平，2010 年。〈同位詞夾子:主題式分類詞庫萃取演算法〉，《數位典藏與數位人文國際研討會論文集》，臺北，頁 551-579。

王昱鈞、呂翊瑄、蔡宗翰、劉青峰、金觀濤、劉昭麟，2012 年，〈漢文文獻之外來語音譯詞擷取方法〉，《第四屆數位典藏與數位人文國際研討會論文集》，臺北，頁 247-263。

彭維謙、劉士綱、杜協昌、翁稷安、項潔，2012 年，〈自動擷取中文典籍中人名之嘗試：以 PMI 斷詞於《資治通鑑》的應用爲例〉，《第四屆數位典藏與數位人文國際研討會論文集》，臺北，頁 53-76。

彭維謙、陳詩沛、程卉，2014 年，〈從全文到表格：地方志職官資料的擷取〉，《第

五屆數位典藏與數位人文國際研討會論文集》，臺北，頁 93-116。

何浩洋，2014 年。〈MARKUS：古籍文本半自動標記平台〉，《第五屆數位典藏與數位人文國際研討會論文集》，臺北，頁 117-137。

嚴漢偉、白璧玲、廖泫銘、林誠謙、范毅軍，2014，〈漢籍全文資料之地名標誌與時空座標建置技術探討〉，《第五屆數位典藏與數位人文國際研討會論文集》，臺北，頁 211-240。

附錄

這篇附錄整理臺灣大學資訊工程學系 303 實驗室的謝博宇同學，實際使用詞夾子系統，從《西遊記》擷取兵器（武器）名稱的過程記錄。以下爲謝同學的參數設定與擷詞策略：

- 詞夾參數 2[2-6]2，起始種子詞彙有四個：金箍棒、如意金箍棒、九齒釘耙、月牙鏟。從詞夾參數可看出，使用者並不考慮長度爲一（例如「刀」、「劍」）的單字兵器名稱。
- 篩選詞夾時，並不會去思考詞夾和夾中詞彙所形成的語意。若候選詞夾能夠夾中足夠數量的正確詞彙（以下稱爲目標詞彙），就判定爲有效詞夾。
- 如果候選詞夾擷取出的詞彙數量和目標詞彙數量不成比例（例如該詞夾可夾中 70 個詞彙以上，當中卻只有一兩個屬於目標詞彙），就捨棄該詞夾，但手動將這個步驟所發現的目標詞彙另外儲存起來。
- 在參閱詞夾出現的前後文時，若發現該節錄的前後文另含有目標詞彙（屬於兵器的詞彙），就利用文本檢視的功能檢查該段落附近是否還有目標詞彙。

在這樣的條件下，第一次迭代可得到 50 個候選詞夾，而謝同學從中篩選出 21 個有效詞夾。在參閱詞夾前後文的過程中，他手動記錄了 43 個目標詞彙：

篩選出的 21 個有效詞夾	
慣使...⊥	長的...⊥
⊥掣...劈頭	又使...⊥
聖使...架住	即取...⊥
即掣...⊥	掣出...⊥
件是...⊥	執著...⊥
⊥把...幌一	⊥輪...⊥
⊥使...劈面	取出...⊥
持著...⊥	一條...⊥
輪起...⊥	收了...⊥
急掣...⊥	
使...⊥	
取出...來⊥	

額外記錄的目標詞彙			
□鋼刀斧	抡撻藤	蟠龍枴杖	長槍
七星劍	方天戟	誅龍的寶劍	降妖寶杖
三尖兩刃神鋒	楮白槍	赤銅刀	降妖真寶杖
三股鋼叉	烏油黑棒子	金箍鐵棒	雕翎箭
九齒鈀	生金棒	金鋼套	青銅劍
四明鏟	硬弩	鉞斧	青鋒寶劍
大刀	蒺藜骨朵	鉤子	風刃宣花斧
大捍刀	藤紇褡	銅錘	
如意棒	蘸鋼刀	鋼刀	
寶劍	虎眼鞭	鋼槍	
彎弓	蛇矛	鐵杵	
悶棍	蟠龍拐	鐵蒺藜	

接著，以這些種子詞夾計算出 382 個候選詞彙，從中挑選出 44 個目標詞彙，並以「新增詞彙」的方式將手動記錄的 43 個目標詞彙補上。由於挑選的詞彙與

手動記錄的詞彙有 11 個重覆，因此在第一次迭代結束後，總共已找出 4 + 44 + (43-11) = 80 個目標詞彙。

接下來進行第二次迭代，得到 237 個候選詞夾。謝同學從中挑選出 8 個有效詞夾，並在參閱詞夾前後文的過程中手動記錄 29 個目標詞彙：

挑選出的有效詞夾
一桿...⊥
撇了...⊥
一柄...⊥
⊥挺...⊥
手執...⊥
各執...⊥
一口...⊥
一把...⊥

額外記錄的目標詞彙			
火尖槍	強弓	混鐵棍	藥杵
七星寶劍	寶刀	畫戟	搗藥杵
鉤子	九瓣銅錘	輕軟狼牙棒	金剛琢
三股叉	九瓣赤銅錘	齊眉棍	敲磬槌兒
軟柄槍	青鋒寶劍	釘把	渾鐵棒
鐵簡	霜刃青鋒	黑纓槍	
短劍	飛龍寶杖	如意金鉤子	
利刃	降妖杖	如意鉤	

以這些挑選出的詞夾對文本進行比對，系統找出 207 個候選詞彙，而謝同學從中挑選出 33 個目標詞彙。由於這些目標詞彙中，有 17 個先前已經出現[39]，因此實際上新增了 14 個目標詞彙。另外，額外記錄的 29 個詞彙中也有 12 個重覆，扣除後相當於新擷取出 17 個詞彙。換句話說，在第二次迭代結束，謝同學已經找出 80 + 14 + 17 = 111 個目標詞彙。

接著再進行第三次迭代，得到 20 個候選詞夾，但從中並沒有發現有效的詞夾。因此擷取過程結束，謝同學總共擷取出以下 111 個詞彙：

謝同學利用詞夾子系統所擷取出的《西遊記》兵器詞彙						
□鋼刀斧	圈子	悶棍	狼牙棒	虎眼鞭	釘把	降妖寶杖
□鋼斧	大刀	戒刀	生金棒	蛇矛	釘鈀	降妖杖
七星劍	大捍刀	扇子	畫戟	蟠龍拐	鉞斧	降妖棒
七星寶劍	如意棒	扎撻藤	畫桿方天戟	蟠龍枴杖	鉤子	降妖真寶杖
三尖兩刃槍	如意的鐵棒	搗藥杵	短劍	誅龍的寶劍	銅錘	雕翎箭
三尖兩刃神鋒	如意金箍棒	敲磬槌兒	短棍	赤銅刀	鋼刀	霜刃青鋒
三楞簡	如意金鉤子	方天戟	硬弩	軟柄槍	鋼叉	青銅劍
三股叉	如意鉤	月牙鏟	磬槌	輕軟狼牙棒	鋼槍	青鋒寶劍
三股鋼叉	如意鉤子	柳棍	神鋒	金剛套	鐵杵	風刃宣花斧
九瓣赤銅錘	宣花斧	棍子	芭蒲扇	金剛琢	鐵棍	飛龍寶杖
九瓣銅錘	宣花鉞斧	楮白槍	芭蕉扇	金弓	鐵棒	飛龍杖
九齒滲金釘鈀	寶刀	混鐵棍	芭蕉扇子	金箍如意棒	鐵簡	黑纓槍
九齒釘耙	寶劍	渾鐵棍	蒺藜骨朵	金箍棒	鐵蒺藜	點鋼大叉
九齒釘鈀	寶杖	渾鐵棒	藤紵褡	金箍鐵棒	鐵鈀	點鋼槍
九齒鈀	強弓	火尖槍	藥杵	金鋼套	鐵錘	齊眉棍
四明鏟	彎弓	烏油黑棒子	蘸鋼刀	金鋼琢	長槍	

這份列表的前兩個詞彙，各包含了一個 □ 字元，它代表文本在打字時無法正確輸入的中文字。另外，「釘把」應該是「釘鈀」在文本打字時的疏失；而「金剛套」與「金鋼套」可能是出自於打字錯誤，也有可能在打字時所依循的印刷版本

[39] 由於實作上的效能考量，系統並沒有在每一個可能的步驟都嘗試從候選詞彙移除「已經被判定屬於目標詞彙」者，因此時還是會出現重覆的詞彙。

就已經出現此變異了，在此我們不加考訂。

值得一提的是，雖然日常用語中「挨了一記悶棍」的「悶棍」並非兵器名稱，但這個詞彙在《西遊記》出現了三次，都是以兵器名稱的樣貌出現：

- 只見那裡森森兵器擺列著：弓箭刀槍甲與衣，干戈劍戟並纓旗。剽槍月鏟兜鍪鎧，大斧團牌鐵蒺藜。長悶棍，短窩槌，鋼叉銃刨及頭盔。（第七十回）
- 這裡孫大聖使金箍棒架住群精，狻猊使悶棍，白澤使銅錘，搏象使鋼槍，伏狸使鉞斧。（第九十回）
- 眾賊歡喜，齊了心，都帶了短刀、蒺藜、枴子、悶棍、麻繩、火把，冒雨前來，打開寇家大門，吶喊殺入。（第九十七回）

這提醒我們，在判斷詞彙是否屬於目標詞彙時，我們並不能僅依賴個人對詞彙的理解與想像，許多時候還是得參看詞彙在文本出現的前後文。有趣的是，這些很容易被人們誤判爲「非目標詞彙」的詞彙，我們卻有機會利用詞夾子系統的「同位詞」特性將它們擷取出來。

在論文中曾提到，詞彙的相關性判斷，有時並不客觀，而會相依於使用者本身的認知與判斷。利用詞夾子系統所擷取到的「敲磬槌兒」就提供一個機會讓我們省思這個問題。「敲磬槌兒」在正常狀況下應該不算兵器[40]，但在《西遊記》中，黃眉怪的狼牙棒兵器卻是這個「敲磬槌兒」所變成。[41]那麼，「敲磬槌兒」究竟算不算兵器呢？

另外還有一點也值得提醒。謝同學在擷取的過程中曾發現「彈弓」這樣的詞彙。他認爲在文本中「彈弓」是分開的兩項武器（類似「刀槍劍戟」分指四樣兵器），因此並沒有將「彈弓」列入目標詞彙。

最後，觀察剛才所列的「悶棍」前後文，我們可發現《西遊記》中許多兵器名稱（例如「剽槍」和「大斧」），都出現在類似韻文的段落中。由於詞夾子無法有效擷取韻文中的兵器名稱，使用者必須手動從韻文中將這些詞彙提取出來。在這次實驗中，謝同學並沒有進行這樣的處理。

---

[40] 根據漢語詞典(http://tw.18dao.net)，磬為佛寺中使用的一種缽狀物，用銅鐵鑄成，既可作念經時的打擊樂器，亦可敲響集合寺眾。

[41] 《西遊記》第六十六回：彌勒上前一把揪住，解了他的後天袋兒，奪了他的敲磬槌兒，叫：「孫悟空，看我面上，饒他命罷。」