在 Emacs 中集成 Recoll 全文搜索

h:1 num:t toc:t

html-postamble:nil

# 需求

时间一长，平常收集的资料就多了，于是使用了 recoll 全文搜索，但是在Emacs 中工作时 间多，搜索要在 Emacs 和 Recoll 图形界面中来回切换，很不方便。而且，如果本地文件 中找不到，要用搜索引擎到网上搜索的话就更麻烦了，于是想能不能把这需要整合一下，平 时找东西直接开 Recoll 搜索，如果没有的话可以一键切到 Google 搜索结果。

# 解决办法

## 原来的办法

这是在网上搜索到的办法，最大的问题就是你看不到摘要，只有在 minibuffer 中看文件名 猜其中有啥，如果不是你想要的东西，还得再输入搜索字符串，重来一回，比在 recoll 图形界面中还要麻烦。

(defun counsel-recoll-function (string &rest \_unused)  
 "Issue recallq for STRING."  
 (if (< (length string) 3)  
 (counsel-more-chars 3)  
 (counsel--async-command  
 (format "recollq -b '%s'" string))  
 nil))  
  
  
(defun counsel-recoll (&optional initial-input)  
 "Search for a string in the recoll database.  
You'll be given a list of files that match.  
Selecting a file will launch `swiper' for that file.  
INITIAL-INPUT can be given as the initial minibuffer input."  
 (interactive)  
 (ivy-read "recoll: " 'counsel-recoll-function  
 :initial-input initial-input  
 :dynamic-collection t  
 :history 'counsel-git-grep-history  
 :action (lambda (x)  
 (when (string-match "file://\\(.\*\\)\\'" x)  
 (let ((file-name (match-string 1 x)))  
 (find-file file-name)  
 (unless (string-match "pdf$" x)  
 (swiper ivy-text)))))))

## 我的解决办法

使用 "recoll -t -A query" 输出查询结果到 Emacs 缓冲中，包含摘要等内容，然后构 建成 html 文档，再用 `eww-display-html' 来显示缓冲。

(defvar recoll-to-html-temmplate "~/Templates/recoll-to-html-template.html")  
(defun recoll-to-html(query)  
 "Use recoll search local file as eww, require `org',`eww',and recoll installed and indexed.  
You can press `n' to call `eww-next-url' google the QUERY."  
 (interactive "sSearch Words:")  
 (require 'org)  
 (require 'eww)  
 (let ((source nil)  
 (google-search-url (format "http://www.google.com/search?q=%s" (url-hexify-string query)))  
 (content-tail-marker nil)  
 (buffer (get-buffer-create "\*recoll\*")))  
 (with-current-buffer buffer  
 (setq inhibit-read-only t )  
 (display-buffer buffer)  
 (erase-buffer)  
 ;; insert template and goto the body insert point  
 (insert-file-contents (expand-file-name recoll-to-html-temmplate))  
 (if (not (re-search-forward "<body>\n</body>" nil t))  
 (message "Invalid html template")  
 )  
 (goto-char (- (point) 7))  
 (setq marker-beg (point))  
 ;; insert the search result  
 (insert  
 (shell-command-to-string  
 (format "recoll -t -A '%s'" query)))  
 (setq content-tail-marker (point))  
 ;; change the keyword display color  
 (goto-char (point-min))  
 (while (re-search-forward "\nABSTRACT\n\\(.+\\)\n/ABSTRACT" nil t)  
 (save-restriction  
 (narrow-to-region (match-beginning 1) (match-end 1))  
 (goto-char (point-min))  
 (replace-string query (concat "<span style=\"color:#F00\">" query "</span>" ))  
 )  
 )  
  
 ;; insert paragraph tags  
 (narrow-to-region marker-beg (point-max))  
 (goto-char (point-min))  
 (insert "<h1>")  
 (goto-char (point-min))  
 (forward-line 2)  
 (insert "</h1>" "\n<p>Search with Google: <a href=\"" google-search-url "\">" query "</a></p>" "\n<p>")  
 (goto-char (point-min))  
 (replace-string "\nABSTRACT\n" "</p>\n<p>")  
 (goto-char (point-min))  
 (replace-string "\n/ABSTRACT\n" "</p>\n<p>")  
 (goto-char (point-min))  
 (replace-string "\n/ABSTRACT" "</p>\n<p>")  
 (goto-char (point-max))  
 (insert "</p>")  
 (goto-char (point-min))  
 (while (re-search-forward "bytes\n\\([\n]+\\)" nil t)  
 (delete-region (match-beginning 1) (match-end 1))  
 )  
 ;; construct html url href  
 (goto-char (point-min))  
 (while (re-search-forward "\\[\\(.\*\\)\\] \\[\\(.\*\\)\\]" nil t)  
 (let ((match-len (length (match-string 0)))  
 (esc-str (org-link-escape (match-string 1)))  
 (display-str (match-string 2))  
 (marker-max (point))  
 )  
 (delete-region (- marker-max match-len) marker-max)  
 (insert "<a href=\"" esc-str "\">" display-str "</a>")  
 )  
 )  
 (goto-char (point-max))  
 (insert "\n<p>Search with Google: <a href=\"" google-search-url "\">" query "</a></p>")  
 (widen)  
 (setq source (buffer-string))  
 ;; display as html by eww  
 (goto-char (point-min))  
 (eww-display-html 'utf-8 nil nil (point-min) buffer)  
 (linum-mode t)  
 (read-only-mode t)  
 (eww-mode)  
 )  
 (with-current-buffer buffer  
 (plist-put eww-data :url "Recoll") ;;here any string make `eww-next-url' work  
 (plist-put eww-data :source source) ;; used for debug  
 (plist-put eww-data :next google-search-url) ;; you can press `n' google the query  
 (plist-put eww-data :title "RECOLL SEARCH RESULTS")  
 (eww-update-header-line-format)  
 (let ((old-data eww-data))  
 (eww-save-history)  
 (setq eww-history-position 0)  
 (dolist (elem '(:source :url :title :next ))  
 (plist-put eww-data elem (plist-get old-data elem)))  
 (run-hooks 'eww-after-render-hook)))  
 ))

但有个问题就是回退的时候，退不到我们的索引页面，原因是 `eww-follow-link' 中 检查了访问 url 的构成，只有同一个页面的才会调用 `eww-save-history'，我不知道 作者是怎么考虑的，反正我给加个补丁，不管到什么 url 都调用 `eww-save-history' 保存当前页面。

第二是他可能会使用外部浏览器打开页面。其实，Emacs 支持打开的页面已经完全可以 满足我的需要了，于是将 `browse-url-browser-function' 设置为 `eww-browse-url'。

(setq browse-url-browser-function 'eww-browse-url)  
(setq shr-external-browser (lambda (url &rest args) (apply 'browse-url-xdg-open url args)))  
(eval-after-load 'browse-url  
 '(defun browse-url-default-browser (url &rest args)  
 " Use EWW as the default browser for search web package. But  
you should set the variable `shr-external-browser' to  
browse-url-xdg-open to make `eww-browse-with-external-browser' to  
work as expected."  
 (apply 'eww-browse-url  
 url args)))  
  
(defun eww-follow-link-before-advice (&optional external mouse-event)  
 (eww-save-history)  
 )  
(advice-add 'eww-follow-link :before #'eww-follow-link-before-advice)

本作品采用[知识共享署名-非商业性使用-禁止演绎 3.0 未本地化版本许可协议](http://creativecommons.org/licenses/by-nc-nd/3.0/deed.zh) 进行许可。