Emacspeak:
全功能音频桌面

《代码之美》 第31章

桌面是用来组织各种办公软件的工作区域

- 图形桌面为我们的日常工作提供了丰富的可视化交互方式，而音频桌面的目的则是为了在免视环境中也能获得同样的效果。因此，音频桌面的主要目标就是通过听觉输出（包括语音输出和非语音输出）的形式来帮助终端用户完成各种计算任务：
- 通过各种电子消息服务来通信。
- 为客户端的本地文档和Web上的全局文档提供即时访问服务。
- 能够在免视环境中高效的开发软件。

从信息的原本内容入手，而不是从信息的视觉表示入手

- 开发Emacspeak音频桌面的动机是基于以下的想法：
- 在为信息提供有效的听觉表现时，我们应该从信息的原本内容入手，而不是从信息的视觉表示入手。正是基于这种想法，我开发出来AsTer技术读物语音系统。后来的主要动机是为了把在听觉文档环境中学到的经验应用于用户界面——也就是说，在界面上实现与文档同样的效果。

- **Emacspeak**的主要目标不是在视觉界面上补充听觉形式，而是创建一个免视的用户界面，不仅用起来很舒适而且还能提高生产效率。
- 在传统的屏幕阅读器中，像滑块和树形控件等界面控件可以直接被翻译为语音输出。虽然这种直接翻译也可以提供完全的免视访问功能，但这种形式的听觉用户界面在使用上却是很低效的。

这些先决条件意味着在音频桌面环境中需要包括：

- 一组核心的语音和非语音的音频服务。
- 丰富的内置应用程序以实现朗读功能。
- 访问应用程序的上下文以产生相应的反馈。

简单的初次实现

- Emacspeak的第一个开发任务是，当用户按上下箭头键时，让Emacs自动朗读位于光标下面的文本行。
- 在Emacs中，用户的所有动作都会调用相应的Emacs Lisp函数。在标准的编辑模式中，按下箭头将调用next-line函数，按上箭头则将调用previous-line函数。为了使这些命令支持语音，在版本0的Emacspeak中实现了下面这个非常简单的advice：

```
(defadvice next-line (after emacspeak)
  "Speak line after moving."
  (when (interactive-p) (emacspeak-speak-line)))
```

- Emacspeak-speak-line函数用来获得位于光标下面的一行文本并把它发送到语音服务器。在前面的定义后，作者开发出来Emacspeak 0.0，它为构建实际的系统提供了基础。

简单的advice教程

- Lisp语言中的advice功能是Emacspeak实现中的关键技术。advice功能可以对现有的函数进行修改而无需改变函数的原始实现。而且，如果advice m修改了函数f，那么所有对该函数的调用都会受到m的影响。
- advice有三种实现形式：
- before：advice代码在原始函数调用之前运行。
- after：advice代码在原始函数调用之后运行。
- around：advice代码代替原始函数运行。around形式的advice可以在必要时调用原始函数。
- 所有形式的advice都可以访问原始函数的参数；此外，around和after这两种形式的advice还能够访问原始函数的返回值。

增加听觉图标

- 丰富的视觉用户界面包括文本和图标。同样，在Emacspeak能够智能的朗读后，接下来的步骤就是通过听觉图标来扩展语音输出，从而提高听觉通信的带宽。
- Emacspeak中的听觉图标都是简短的声音片段，用于表示在用户界面中频繁发生的事件。例如，当用户保存文件时，系统将播放一个确认声音。同时，在打开或者关闭某个对象时将播放相应的听觉图标。这组听觉图标是被陆续实现的，例如打开、关闭或者删除对象等。本节将描述如何将这些听觉图标插入到Emacspeak的输出流中。

- 听觉图标是以下的用户交互中产生的：
- 提示明确的用户动作。
- 在语音输出中增加额外的提示。
- 确认用户动作的听觉图标——例如，文件被成功的保存——是通过把一个after-advice增加到不同的Emacs内置函数中来实现的。
- 为了在整个Emacspeak桌面中提供一致的声音和感觉，在Emacs中许多调用这段代码的地方都要添加这种扩展。

- 以下是通过advice来实现的一个扩展实例：

```
(defadvice save-buffer (after emacspeak pre act)
  "Produce an auditory icon if possible."
  (when (interactive-p) (emacspeak-auditory-icon 'save-object)
    (or emacspeak-last-message (message "Wrote %s" (buffer-file-name)))))
```

- 此外，还可通过Emacs提供的钩子来实现扩展。
- 我们在前面给出的advice简短介绍中已经解释过，advice能够使我们在扩展或者修改现有软件的行为时无需修改软件的源代码。Emacs本身就是一个可扩展的系统，而那些写的很好的Lisp代码也能够在通常情况中提供合适的扩展钩子。

- 例如，在Emacspeak中把函数Emacspeak-minibuffer-setup-hook增加到Emacs的minibuffer-setup-hook，从而将听觉反馈添加到Emacs的默认提示机制（Emacs中的minibuffer）中：

```
(defun emacspeak-minibuffer-setup-hook ( )
  "Actions to take when entering the minibuffer."
  (let ((inhibit-field-text-motion t))
    (when emacspeak-minibuffer-enter-auditory-icon
      (emacspeak-auditory-icon 'open-object))
    (tts-with-punctuations 'all (emacspeak-speak-buffer))))
(add-hook  'minibuffer-setup-hook 'emacspeak-minibuffer-setup-hook)
```

- 这是最大限度使用内置扩展能力的一个很好示例。
- 然而，Emacspeak在许多情况中使用的都是advice，这是因为Emacspeak需要为Emacs中的所有函数都增加听觉反馈，而在当初实现Emacs的时候并没有考虑到这种需求。
- 这样，Emacspeak实现就很好证明了advice是一种发现扩展点的强大工具。

- 由于在普通的编程语言中没有像advice这样的功能，因此难以进行一些实验，尤其是在寻找有用的扩展点时。因为软件工程师需要考虑以下的权衡因素：
- 使系统能够实现任意的扩展（以及任意的复杂）。
- 在某些合理的扩展点上进行猜测，并且把这些猜测用硬编码的方式来实现。
- 在实现了扩展点后，用新的扩展点来试验就需要重新编写现有的代码，这种做法所带来的时间开销通常意味着像大多数这样的扩展点在很长的时间内无法被发现。Lisp中的advice及Java中的aspects，都可以使软件工程师在进行试验时无须担心会对现有源代码造成负面影响。

在朗读内容时产生听觉图标

- 除了使用听觉图标来提示用户交互的结果外，Emacspeak还使用听觉图标来扩充所朗读的内容。这种听觉图标的示例用法包括：
- 在段落开始之处朗读一个短听觉图标。
- 在越过带有断点的源代码行时朗读一个听觉图标标记对象。
- 听觉图标是通过在文本属性Emacspeak-auditory-icon上添加一个值来实现的，这个值是将在相关的文本上播放的听觉图标的名字。

日历：通过上下文敏感的语义来增强语音输出

- 总结到目前为止所讨论的内容，Emacspeak有以下的能力：
- 在应用程序的上下文中产生听觉输出。
- 实现音频格式输出以提高口语通信的带宽。
- 用听觉图标来扩充语音输出。
- 在本节中将解释这个设计的一些增强功能。

- 如果想要说明在视觉媒介和传输信息上都进行了优化的视觉布局，那么日历无疑是一个很好的示例。
- 在描述日期时，我们在直觉上会按照日期和月份的形式来进行思考；通过表格布局将日期组织成网格形式，每个星期都出现在表格的一行，这种表示方式很符合我们的直觉。
- 通过这种形式的布局，人眼可以快速的在日历中按照日期、星期和月份来浏览日历，并且可以很容易回答“明天是几号”以及“我在下个月的星期三是否有空”等这些问题。

- 然而，如果只是简单的把这个二位布局朗读出来，那么无法在听觉交互中获得在视觉上下文中的信息获取效率。
- 这个结论很好的印证了这个观点：良好的听觉反馈应该直接从传输信息的本质内容中产生出来，而不是从信息的视觉表示中产生。
- 当需要从视觉格式信息中产生听觉输出时，软件必须在朗读信息之前重新找到在信息中包含的基本语义。

- 相反，如果通过扩展底层应用程序的advice定义来产生语音反馈，那么软件就能够完全访问应用程序的运行时上下文。这样，软件不仅是基于视觉布局来进行推测，而是可以指导底层应用程序朗读正确的内容！
- Emacspeak-calendar模块通过定义朗读日历信息的工具函数以及对调用这些函数的日历导航命令进行advice修改来实现朗读的功能。因此，emacs日历通过把箭头按钮绑定到日历导航命令而不是普通编辑模式中的默认光标导航来产生专门的行为。Emacspeak对日历特定的命令使用advice，从而在日历的上下文中朗读相关的信息。

- 这种做法的结果就是，从终端的用户观点来看，一切都工作正常。在普通的编辑模式中，按上下箭头将会朗读当前行；在日历中按上下箭头则会按照星期来导航，并且朗读当前日期。

对于在线信息的简单访问

- 在有了能够产生丰富听觉输出的所有必要功能后，在支持语音的Emacs LISP的advice功能只需要非常少的代码。TTS层和Emacspeak内核将处理产生高质量语音输出的细节内容，而支持语音功能的扩展只需关注独立应用程序的具体语义；这使我们能够实现简单而漂亮的代码。
- 在本节中，我将从Emacspeak丰富的信息访问软件工具包中选择一些实例来阐述这个概念。

- 就在作者着手开发Emacspeak的时候，在计算世界中发生了一次更为深刻的革命：万维网从研究机构中的工具变成了日常工作的主要模式。这是发生在1994年，当时编写浏览器仍然是一个相对简单的任务。在之后的12年中，在Web中不断增加的复杂性经常会模糊这个事实：Web仍然是非常简单的设计，其中
- 内容的创建者通过**URL**来发布网页资源。
- 通过公开的协议来获取**URI**上的内容。
- 可获取的内容是用**HTML**语言来编写的，这是一种易于理解的标记语言。

- 注意，刚才粗略介绍的基本框架并没有提到如何使内容对于终端用户是有效的。在20世纪90年代中期，我们看到了Web中日益复杂的视觉交互。在商业网页上越来越多华而不实的视觉交互，使其逐渐远离了早期网站中简单的面向数据交互。
- 到1998年，作者发现在Web上出现了许多有用的交互站点；但令人沮丧的是，作者所访问的站点正逐步减少，因为在语音输出时，访问这些站点需要很长的时间。

- 这使得作者在Emacspeak中创建了一组面向网页的工具软件已重新回到网页交互的基本方式。
- Emacs已经能够把简单的HTML重新处理为交互的超文本文档。
- 随着 Web变得日益复杂，Emacspeak得到了一组构建在Emacs中HTML表现功能上的交互向导，这组向导能够逐渐的去掉网页交互中的复杂性，从而创建一个听觉界面，使用户能快速的听到想要的信息。

使用Emacs W3和Aural CSS的基本HTML

- Emacs W3是一个简单的网页浏览器，最初是在20世纪90年代中期实现的。Emacs W3在早期实现了CSS，而这正是最初ACSS实现的基础，这个ACSS实现是我在1996年2月份编写Aural CSS草稿时发布的。Emacspeak通过Emacspeak-w3模块在Emacs W3中支持语音功能，在这个模块中实现了以下扩展：

- 在Aural CSS的默认样式列表中有一个Aural media字段。
- 在所有的交互命令中都增加了advice以产生听觉反馈。
- 使用特殊的模式来识别网页上的修饰性图片，并且不朗读这些图片。
- 为HTML表单域的听觉表现定义相关联的标签，用来支持HTML4中label元素的设计。
- 为HTML表单控件定义上下文敏感的听觉表现。
- 为Emacs w3的函数w3-parse-buffer定义了一个before形式的advice，这个函数用来把用户请求的XSLT转换应用到HTML页面上。

面向任务搜索的Emacspeak-websearch模块

- 到1997年，Web上的交互网站，无论是用于搜索的Altavista还是在线方位指示信息的Yahoo！Maps，都要求用户进行高度视觉化的操作，包括：
- 1、填充一组表单域。
- 2、提交结果表单。
- 3、在复杂的HTML网页中找到结果。
- 其中第一个和第三个操作在使用语音输出时需要使用很多时间。首先需要在一个琳琅满目的网页上找出各种表单域，并且在获得答案之前，还要在结果页面上浏览很多复杂的信息。

- 注意，从软件设计的角度来看，这些步骤可以很巧妙的映射为前置动作和后置动作等钩子。
- 因为网页交互遵循URI的简单架构，提示用户正确输入的前置动作步骤可以从网站上提取出来，并放到一小段局部运行的代码中；这样，用户就不需要打开网页并且找出各种输入域了。
- 同样，在结果网页中找出实际结果的后置动作步骤同样能够用相应的程序来实现。

- 最后，我们注意到，即使每个前置动作和后置动作都是特定于某些网站的，但可以对整体的设计模式进行一般化。基于这个观点，我开发了Emacspeak-websearch模块，这是一组面向任务的网页工具：
- 1、提示用户。
- 2、构造一个合适的URI并且提取出其中的内容。
- 3、在通过Emacs W3呈现相关内容之前，对结果进行过滤。

- 以下是Emacspeak-websearch中的一个工具，用于从Yahoo！Maps中访问方向提示信息：

```
(defsubst emacspeak-websearch-yahoo-map-directions-get-locations ()
  "Convenience function for prompting and constructing the route component."
  (concat
   (format "&newaddr=%s"
           (emacspeak-url-encode (read-from-minibuffer "Start Address: ")))
   (format "&newcsz=%s"
           (emacspeak-url-encode (read-from-minibuffer "City/State or Zip:")))
   (format "&newtaddr=%s"
           (emacspeak-url-encode (read-from-minibuffer "Destination Address: ")))
   (format "&newtcsz=%s"
           (emacspeak-url-encode (read-from-minibuffer "City/State or Zip:")))))
(defun emacspeak-websearch-yahoo-map-directions-search (query )
  "Get driving directions from Yahoo."
  (interactive
   (list (emacspeak-websearch-yahoo-map-directions-get-locations))
   (emacspeak-w3-extract-table-by-match
    "Start"
    (concat emacspeak-websearch-yahoo-maps-uri query))))
```

- 下面给出了对上述代码的简单解释：
- 前置动作:
- emacspeak-websearch-yahoo-directions-get-locations函数将向用户提示起始位置和结束位置。
- 内容提取：
- 用于提取方向指示信息的URL是通过把用户的输入连接到Yahoo！Maps的基URI来构造的。
- 后置动作：
- 结果URL以及搜索模式Start被传递给Emacspeak-W3-extract-table-by-match函数：
- a.使用Emacs W3来获取内容。
- b.使用XSLT转换来提取包含Start的表格。
- c.用Emacs W3的HTML格式器来表示这个表格。

- 在这个工具编写出来7年之后，Google于2005年2月份发布了Google Maps。网络上许多博客都对Google Maps进行了仔细的研究并很快找到了这个应用程序所使用的查询参数。
- 作者使用这些查询参数在Emacspeak中构建了相应的Google Maps工具来提供类似的功能。用户能很快的适应Google Maps工具，因为在相同的参数内可同时指定起始位置和结束位置。以下是Google Maps向导的代码：

```
(defun emacspeak-websearch-emaps-search (query &optional use-near)
  "Perform EmapSpeak search. Query is  in plain English."
  (interactive
   (list
    (emacspeak-websearch-read-query
     (if current-prefix-arg
         (format "Find what near  %s: "
                 emacspeak-websearch-emapspeak-my-location)
       "EMap Query: "))
    current-prefix-arg))
  (let ((near-p ;; determine query type
         (unless use-near
           (save-match-data (and (string-match "near" query) (match-end 0)))))
        (near nil)
        (uri nil))
    (when near-p ;; determine location from query
      (setq near (substring query near-p))
      (setq emacspeak-websearch-emapspeak-my-location near))
    (setq uri
          (cond
           (use-near
            (format emacspeak-websearch-google-maps-uri
                    (emacspeak-url-encode
                     (format "%s near %s" query near))))
           (t (format emacspeak-websearch-google-maps-uri
                      (emacspeak-url-encode query)))))
    (add-hook 'emacspeak-w3-post-process-hook 'emacspeak-speak-buffer)
    (add-hook  'emacspeak-w3-post-process-hook
               #'(lambda nil
                   (emacspeak-pronounce-add-buffer-local-dictionary-entry
                    "&#240;mi" " miles ")))
    (browse-url-of-buffer
     (emacspeak-xslt-xml-url
      (expand-file-name "kml2html.xsl" emacspeak-xslt-directory)
      uri))))
```

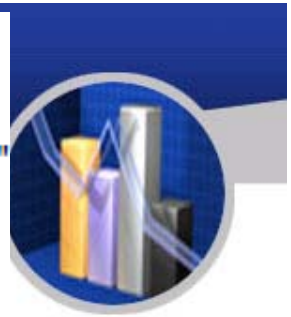
- 对代码的解释：
- 1、对输入进行解析以确定是一个方向指示查询还是搜索查询。
- 2、当为搜索查询时，缓存用户的位置以便在将来使用。
- 3、构造用于获取查询结果的URI。
- 4、通过XSLT过滤器对URI内容进行过滤并浏览过滤的结果，并把获取的内容转换为简单的html。
- 4、在结果中建立起定制的发音来把“mi”发音为“Miles”

网页命令行和URL模板

- 随着网络上日益增多的服务，在2000年早期出现了另一种有用的模式：网站开始通过Javascript创建智能的客户端交互。这种脚本的一个典型应用就是在客户端构造URL，用来根据用户的输入访问特定部分内容。例如：美国职业棒球大联盟通过把比赛日期以及主队和客队的名字组合在一起来构造URL，从而获得特定比赛的比分。
- 为了实现对这些服务的快速访问，作者在2000年下半年增加了Emacspeak-URL-template模块。这个模块成为了Emacspeak-Websearch模块的一个功能强大的辅助工具。这些模块共同把Emacs的minibuffer变成了一个功能强大的网页命令行，用来提供对网页内容的快速访问。

- Emacspeak中的URL模板将使用以下的数据结构实现：

```
(defstruct (emacspeak-url-template (:constructor emacspeak-ut-constructor))
  name                                  ;; Human-readable name
  template                              ;; template URL string
  generators;; list of param generator
  post-action                     ;; action to perform after opening
  documentation                         ;; resource  documentation
  fetcher)
```



```
(emacspeak-url-template-define
 "NPR On Demand"
 "http://www.npr.org/dmg/dmg.php?prgCode=%s&showDate=%s&segNum=%s&mediaPref=RM"
 (list
  #'(lambda () (upcase (read-from-minibuffer "Program code:")))
  #'(lambda ()
      (emacspeak-url-template-collect-date "Date:" "%d-%b-%Y"))
  "Segment:")
 nil; no post actions
 "Play NPR shows on demand.
Program is specified as a program code:
ME              Morning Edition
ATC             All Things Considered
day             Day To Day
newsnotes       News And Notes
totn            Talk Of The Nation
fa              Fresh Air
wesat           Weekend Edition Saturday
wesun           Weekend Edition Sunday
fool            The Motley Fool
Segment is specified as a two digit number --specifying a blank value
plays entire program."
 #'(lambda (url)
     (funcall emacspeak-media-player url 'play-list)
     (emacspeak-w3-browse-xml-url-with-style
      (expand-file-name "smil-anchors.xsl" emacspeak-xslt-directory)
      url)))
```

在这个示例中，自定义的获取器执行两个动作

1、启动一个媒体播放器来播放音频流。

2、通过XSLT对相应的文档进行过滤。

- 用户通过Emacspeak命令Emacspeak-URL-template-fetch来调用URL模板，这个命令将提示输入URL模板并且：
- 1、查找命令模板。
- 2、通过调用指定的生成器来提示用户。
- 3、将Lisp函数格式应用到模板字符串，并把所收集的参数用来创建最终的URL。(翻译不准确）
- （3、将Lisp函数format应用到模板字符串和收集的参数，创建最终的URL。）
- 4、建立起在内容被表现之后执行的任意后置动作。
- 5、应用指定的获取器来表现内容。

应用模板获取URL的过程

```
(defun emacspeak-url-template-url (ut)
  "Instantiate URL identified by URL template."
  (declare (special emacspeak-url-template-current-ut))
  (setq emacspeak-url-template-current-ut nil)
  (apply 'format
         ( emacspeak-url-template-template ut)
         (mapcar
          #'(lambda (g)
              (let ((input nil))
                (setq input
                      (cond
                       ((stringp g)
                        (emacspeak-url-encode (read-from-minibuffer g)))
                       (t (funcall g))))
                (setq emacspeak-url-template-current-ut
                      (list (emacspeak-url-template-name ut) input))
                input))
          (emacspeak-url-template-generators ut))))
```

小结

- Emacspeak在构思时是一个功能完备的、能够用于日常工作的免视用户界面。为了实现完备的功能，系统需要为桌面工作站上计算任务的各个方面都提供直接访问。为了实现流畅的免视交互，系统需要把语音输出和听觉媒介视为系统中的一等公民——也就是说，仅仅朗读在屏幕上显示的信息是不够的。
- 为了提供全功能的音频桌面，目标环境需要是一个交互框架，既可以被广泛的部署，也可以被完全的扩展。为了能够实现比朗读屏幕更多的功能，系统需要把交互式的语音能力构建到不同的应用程序中。

- 最后，在实现这个功能时不能修改任何底层应用程序的源代码；这个项目无法为了增加免视交互的能力而对一组应用程序进行修改，因为我希望所有的工作仅限于维护语音扩展。
- 为了满足所有这些设计需求，我选择了Emacs作为用户的交互环境。作为一个交互框架，Emacs的优势在于它拥有大量的开发人员群体。与我在1994年开始这个项目时的其他交互框架不同，它的重要优势之一就是它是一种免费的软件环境。

- 在各种应用程序中实现语音功能时，Emacs Lisp作为一种可扩展语言，它所提供的巨大灵活性是一个重要的先决条件。这个平台的开源特性也是很重要的；虽然我决定不去修改任何源代码，但分析各种应用程序的实现方式将更容易使这种程序支持语音。最后，Emacs Lisp（注意，Lisp中的advice功能是面向方向编程最主要的推动力）中对于advice的高质量实现使得用Emacs Lisp编写的应用程序在实现语音时无需修改源代码。
- Emacspeak是把前面描述的需求与Emacs作为用户交互环境所提供的功能结合起来的结果。

长期以来管理代码的复杂性

- Emacspeak代码已经发展了12年的时间了。除了第一次开发花了六个星期以外，代码库的开发和维护都是在Emacspeak中来完成的。本节将总结在管理代码复杂性时学到的一些经验。
- 在Emacspeak的开发过程中，它始终是作为一个业余时间的开发项目。看着代码库经历了这么长的时间，我相信这对它今后的发展有着重要的影响。当全职开发一个大型并且复杂的系统时，开发人员能够在一段时间内把他的所有注意力都集中在代码库上——例如，6到12个星期。这就导致了创建深代码库的紧密开发代码。

- 尽管开发人员有着美好的愿望，但随着时间的过去，代码很可能变得难以阅读。那些只有一位工程师长期投入于某个项目的大型软件系统几乎是不存在的；这种忠诚的关注通常会很快的消耗殆尽！
- 与之相反，Emacspeak是一个只有一位工程师在这上面工作了12年的大型软件系统，但仅在他的业余时间里。多年来独自开发这个系统的结果就是，代码库自然而然的是“独立的”。注意，在下表中总结了代码行数和文件的分布情况。

TABLE 31-1. Summary of Emacspeak codebase

Layer	Files	Lines	Percentage
TTS core	6	3866	6.0
Emacspeak core	16	12174	18.9
Emacspeak extensions	160	48339	75.0
Total	182	64379	99.9

- 在上表中强调了以下要点：
- 负责高质量输出的TTS内核分布在182个文件的6个文件中，并且占代码库总量的6%。
- Emacspeak内核——为Emacspeak扩展提供了高级语音服务，此外还使得所有基本的Emacs功能都支持语音——分布在16个文件中，并且占代码库总量的19%。
- 系统的其他部分分布在160个文件中，对这些文件可以单独进行修改（或者拆分）而不会影响系统的其他部分。许多模块，例如Emacspeak-URL-template——本身就是茂密的（bushy）——也就是说，可以在修改某个URL模板不会影响到任何其他的URL模板。
- advice减少了代码量。在Emacspeak代码库中大约有60000行Lisp代码，这与底层需要支持语音的系统代码数量相比是非常小的；此外，Emacspeak还使得其他大量应用程序也支持语音，而这些程序在默认情况下并没有捆绑到Emacs中。

结论——技术篇

- 以下在开发和使用Emacspeak中的一些心得：
- Lisp advice及其在面向方面编程方面的类似技术，是实现横切关注点的有效方式——例如使视觉界面支持语音。
- Advice是一种在复杂软件系统中发现潜在的扩展点的功能强大的方法。

结论——设计篇

- 关注基本的网页架构并且依靠有标准的协议和格式所支撑的面向数据的网络，它带来了功能强大的语音网页访问。
- 关注最终用户的体验，这与像滑块和树形控件这样的独立交互widget控件不同，它带来了高效而且免视的环境。

结论——设计篇

- 视觉交互非常依赖人眼快速扫描视觉显示的能力。高效的免视交互需要把这种能力中的一部分转移到计算机上，因为听取大量的信息是非常耗时的。这样，在每种形式上进行搜索对于提供有效的免视交互来说就是非常重要的，这包括从最小量级（例如Emacs的增量搜索，用来在本地查找正确的文档）到最大量级（例如Google搜索，用来在全球Web上快速查找正确的文档）的搜索。

结论——前景篇

- 视觉复杂性可能只会刺激用户使用复杂视觉交互，它是实现免视交互的阻碍之一。当复杂视觉界面所带来的坏处超过一定限度时，在早期免视环境中出现的工具软件最终会成为主流软件。
- 下面给出了在Emacspeak中获得的两个经验：
- ——RSS和Atom Feed代替了网页内容查找，这样在检索像文章标题这样的基本信息时无需查找所有网页内容。
- ——Emacspeak在2000年使用了XSLT来过滤内容，这与2005年出现的将自定义客户端Javascript应用于Web的Greasemonkey技术是同等重要的。