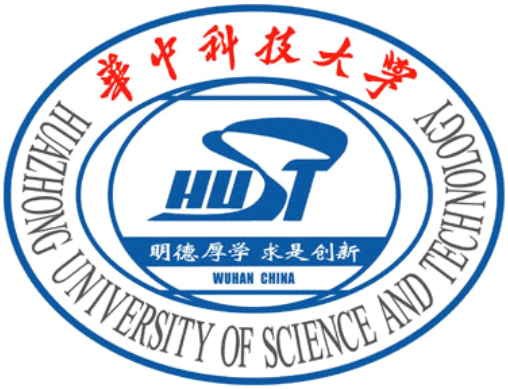
名称	密级
本地文件搜索	开源
版本	共54页
内测版	



本地文件搜索
概要设计说明书

拟制	孙经东	日期	2012-11-12
评审		日期	

修订记录

日期	修订版本	修改章节	修改描述	作者
2012-11-12	Alpha	1-2	完成概要设计基本框架	孙经东
2012-12-12	Alpha	1-2	完成各个模块的分解描述	孙经东
2012-1-2	Alpha	2.1-2.2.1	完成零层设计描述和 Shell 模块	孙经东
2012-1-7	Alpha	2.2.2	完成二层 Engine 模块	孙经东
2012-1-8	Beta	1-2	升级为 Beta 版本	孙经东
2012-1-12	Beta	1-2	根据代码修改部分重写	孙经东

目录

表目录

图目录

1 简介

1.1 目的

本文在需求文档的基础上，具体阐述了本地文件搜索软件的概要设计，用于指导本项目开发人员进行编码和测试工作，是项目后续工作的基础。

1.2 范围

1.2.1 软件名称及功能

本地文件搜索引擎

1.2.2 软件应用

本软件针对中小型文件目录，建立其文件/文件夹的名称索引，及文件的内容索引。提供针对名称和内容的搜索服务功能。适用于多平台。

2 概要设计

2.1 第零层设计描述

2.1.1 软件系统上下文定义

本软件可以工作在预装有Perl和Tcl解释器的操作系统中，通过解释器完成对代码的解析，并执行结果。其模块架构图如下图所示。

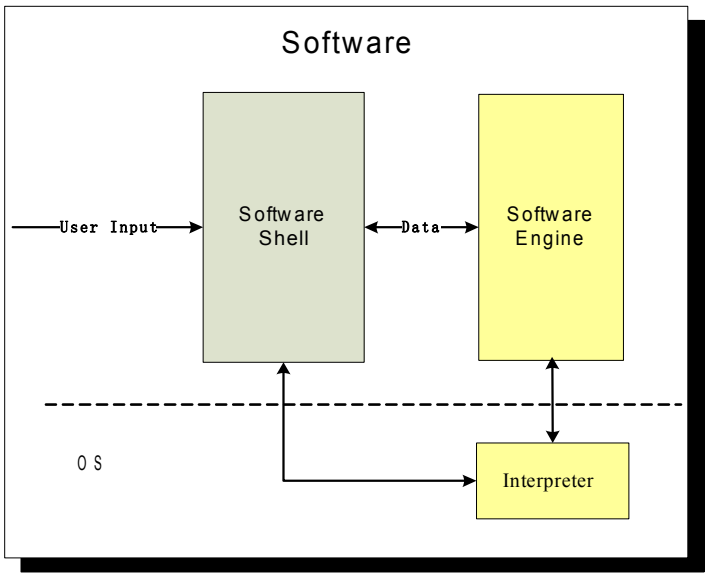


图1 文件搜索软件模块架构图

搜索引擎分为前端和后台两部分，前端负责与用户交互，获取及呈现数据，后台负责运行数据的搜索、计算等处理。但实际上，前端和后台跑在同一个进程中，其分界仅仅是功能上的区分，没有物理上的界限。

Search Shell代指前端，通过控件完成与用户的交互。其中采用插件的思想，实现多样化的主题更换功能。

Search Engine代指后台，将在一层分解中划分为多个功能独立的模块，向Shell提供不同接口，完成相应任务。

此外，整个系统需要数据库及相关外部模块的支持。

本地文件搜索引擎的顶层分解如下图所示，

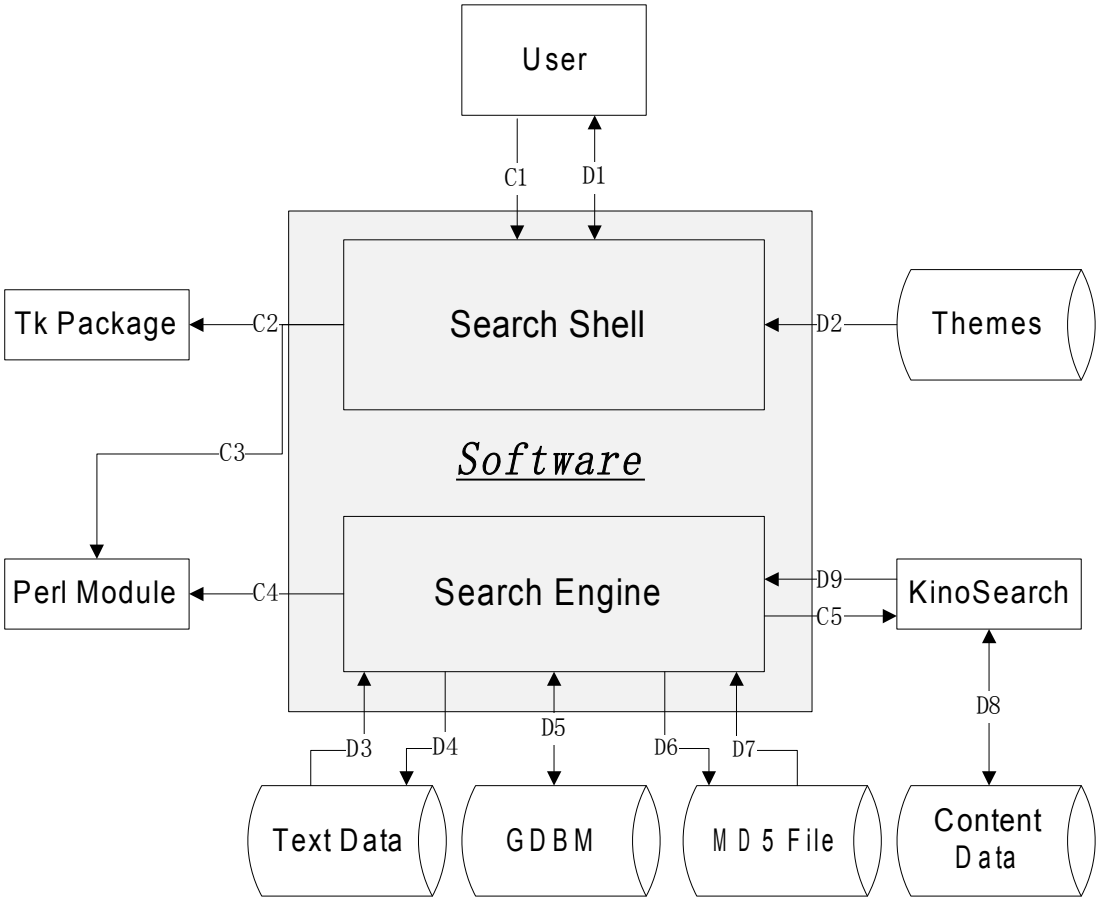


图2 零层分解图

控制流说明：

C1：用户通过软件上的控件，对整个搜索过程进行控制，输入关键词，观察结果。

C2：软件调用Tk的外部Package，实现Gif动画播放。

C3：软件调用Perl外部模块，实现编码、文件信息头读取等辅助功能。

C4：软件调用Perl外部模块，完成例如MD5校验等辅助功能。

C5：软件借助工具KinoSearch，调用其接口实现全文索引和关键词搜索。

数据流说明：

D1：用户输入以及从软件界面上获取的信息。

D2：软件从主题库中获取素材和配置信息，完成主题装载。

D3：索引信息从文本数据库恢复。

D4：建立或更新索引时，向文本数据库备份。

D5：运行数据中哈希结构与GDBM绑定，即时存取二进制数据库中的内容。

D6：索引建立后保存MD5校验数据。

D7：载入数据库信息时读取校验文件，查看原始数据是否损坏。

D8：KinoSearch存取全文检索数据

D9：KinoSearch将检索到的全文数据结果反馈给软件。

2.1.2 设计思路

1.设计方法

本软件采用自顶向下，逐层分解的设计方法。

2.设计可选方案

1） 基本数据组织

请参见各层分解数据依赖关系和数据组织章节

2） 数据库设计

本软件设计两大类数据库：

- 文件目录数据库

  基于GDBM，并将原始的单个数据库文件均分为七个小库，形成数据库群，由重定向哈希结构指引索引和搜索时的目标数据库。提高搜索效率。

- 全文检索数据库

  由KinoSearch建立，KinoSearch属于Apache下的Lucy项目，向Perl提供全文检索接口。速度大大优于Plucene。Plucene为Lucene以Perl语言的实现，其效率对比如下。

KinoSearch Benchmarks

RESULTS A: Documents indexed, but full text not stored			
Engine	Environment	Time	Memory
Lucene 1.9.1	JVM 1.4	40.40 secs	80 MB
Lucene 1.9.1	JVM 1.5	42.17 secs	94 MB
KinoSearch 0.09	Perl 5.8.8	58.01 secs	28 MB
KinoSearch 0.09	Perl 5.8.6	68.23 secs	29 MB
Plucene 1.24	Perl 5.8.8	1914.07 secs	skipped
Plucene 1.24	Perl 5.8.6	1951.02 secs	skipped

RESULTS B: Documents indexed, stored, and "vectorized"			
Engine	Environment	Time	Memory
Lucene 1.9.1	JVM 1.4	57.71 secs	161 MB
Lucene 1.9.1	JVM 1.5	59.99 secs	193 MB
KinoSearch 0.09	Perl 5.8.8	62.91 secs	28 MB
KinoSearch 0.09	Perl 5.8.6	74.77 secs	29 MB
Plucene 1.24	Perl 5.8.8	1931.49 secs	skipped
Plucene 1.24	Perl 5.8.6	1963.82 secs	skipped

Test Corpus

19043 news articles drawn from the Reuters 21578 collection, Distribution 1.0.

图3 KinoSearch Benchmarks

可以看到KinoSearch的效率大大优于Plucene，接近原声Lucene的速度。实测结果差距更大，对于超过30MB的文本文件，Plucene保持卡死状态持续超过30分钟，KinoSearch在30秒内完成该文件全文索引的建立。所以我们选用KinoSearch。

3） 多线程设计

无。

最初几个版本中曾添加多线程支持，子线程搜索，主线程可以执行其他任务。但是发现，作为一个本地文件搜索软件，除了搜索之外，没有其他任务可做……

考虑多线程搜索，从Lucene的经验来看，多线程的搜索效率低于单线程，部分原因是IndexReader实例的创建和销毁消耗较大，另外磁盘IO也有限制，使得多线程搜索效果不理想。

考虑到可以利用多线程加快索引建立的速度。

3.设计约束

想象力

4.其他

无

2.2 第一层设计描述

根据零层分解的描述，本地文件搜索引擎第一层分解为2个模块：

Shell

通过控件完成与用户的交互，不同控件有各自明确的职能。其中采用插件的思想，实现多样化的主题更换功能。

Engine

以多个自定义的模块的形式组织，向Shell及功能脚本提供多种接口，完成诸如 索引建立、搜索、分词、获取网络信息等功能。其中模块间允许调用以完成综合性的功能，但是不允许相互调用。

一层分解示意图如下。

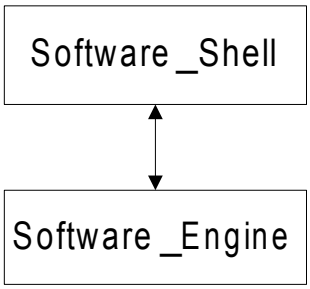


图4 一层模块分解图

2.2.1 分解描述

1.模块/子系统分解

1） Shell模块描述

标识：Software_Shell

类型：一级子模块

目的：用户交互

功能列表：

- 界面元素初始化及加载
- 管理及组织各个控件
- 维护结果信息及相关数据结构
- 响应操作事件并进行处理
- 调用Engine相关接口完成对应功能
- 主题管理

要实现的需求ID：

表1 Shell模块需求ID

需求点序号	需求点名称
SRS.SHELL.GENERAL.001	交互界面整体设计
SRS.SHELL.CMD.001	命令行调试程序设计

子模块：

- Software_Shell_Specific
- Software_Shell_Fuzzy
- Software_Shell_More
- Software_Shell_Exit
- Software_Shell_Cmd
- Software_Shell_Theme

2） Engine模块描述

标识：Software_Engine

类型：一级子模块

目的：索引建立、文件搜索、关键词分词、网络信息提取

功能列表：

- 针对文件/目录名建立倒排索引，对特定格式的文件进行全文索引
- 维护二进制数据库、文本库、MD5校验文件及全文检索数据库

- 针对输入字符串进行分词处理
- 针对文件/目录及文件内容进行关键词搜索
- 对搜索结果的关联度给予评估并给出量化分数
- 历史记录及关键词搜索建议

要实现的需求ID：

表2 Engine模块需求ID

需求点序号	需求点名称
SRS.INDEX.DB.001	初始化本地数据库
SRS.INDEX.DB.002	读取本地数据库
SRS.INDEX.DB.003	存储本地数据库
SRS.INDEX.DB.004	删除本地数据库
SRS.INDEX.DB.005	重建本地数据库
SRS.INDEX.DB.006	载入本地数据库
SRS.INDEX.DATA.001	绑定索引与数据库
SRS.INDEX.DATA.002	添加新索引
SRS.INDEX.DATA.003	清空索引数据
SRS.INDEX.DATA.004	索引关键词的合并
SRS.INDEX.DATA.005	建立压缩文件索引
SRS.SPLIT.WORD.001	分词处理
SRS.SPLIT.KEYWORD.001	针对数字分词
SRS.SPLIT.KEYWORD.002	针对英文分词
SRS.SPLIT.KEYWORD.003	针对中文分词
SRS.SPLIT.EXTEND.001	针对扩展名处理
SRS.SEARCH.SPECIFIC.001	精确搜索模式
SRS.SEARCH.FUZZY.001	模糊搜索模式
SRS.SEARCH.SCORE.001	搜索结果关联度评估
SRS.PLU.BUILD.001	文件内容索引的建立
SRS.PLU.BUILD.002	文本和PDF文件内容索引处理
SRS.PLU.BUILD.003	HTML文件内容索引处理

SRS.PLU.SEARCH.001	文件内容的搜索

子模块：

- Software_Engine_Index
- Software_ Engine _Search
- Software_ Engine _Split
- Software_ Engine _Net
- Software_ Engine _Kino

2.并发进程处理

Shell和Engine模块运行于同一进程，不存在并发进程。

3.数据分解

请参见一层分解相关描述。

2.2.2 依赖性描述

1.运行设计

1） 一层模块搜索过程

顺序图如下：

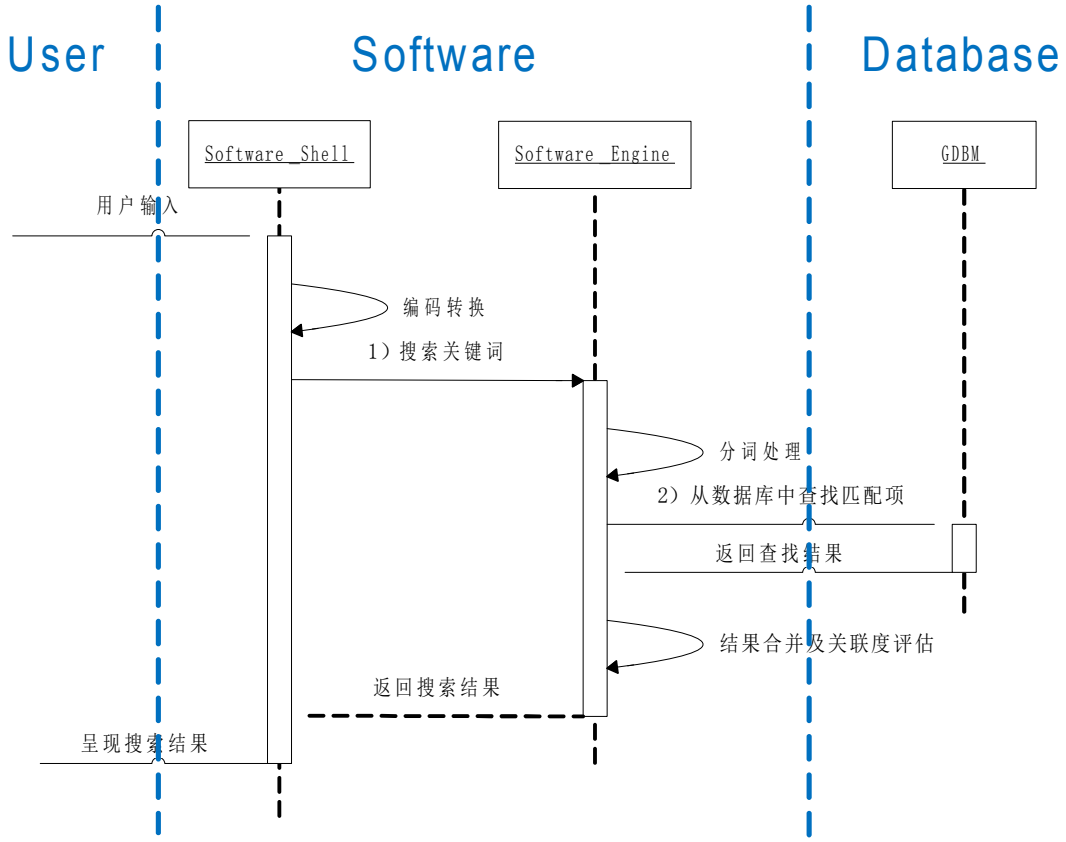


图5 一层模块搜索运行设计

说明：

1) Software_Shell收到用户输入的搜索指令后，首先判断是否输入为空，如果不为空，则对字符转化为统一编码，调用接口Search::SearchData，将用户输入的字符串及搜索配置传递给Engine处理。

2) Engine获取到数据后，首先进行分词处理，之后通过接口DBM::Deep::Tie绑定二进制库，搜索索引数据查找匹配项。经后续处理后返还给Shell显示。

注：如果为模糊搜索模式，1）中调用接口为Search::FuzzySearchData；如果是搜索文件内容，1）中调用接口为Kino::SearchContentData。

2） 一层模块获取网络信息

顺序图如下：

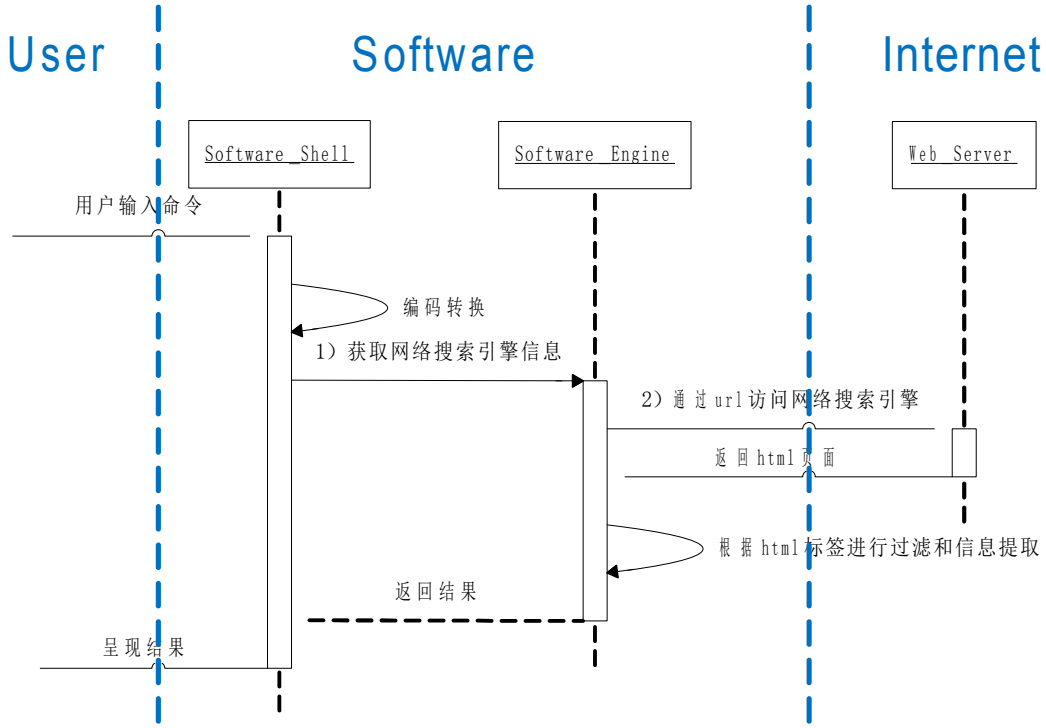


图6 一层模块搜索运行设计

说明：

1) Software_Shell收到用户输入的获取网络信息指令后，对之前的搜索字符转化为统一编码，调用接口Net::BaiduIt，将用户输入的字符串传递给Engine处理。
2) Engine获取到数据后，直接将搜索字符串与url合并，调用接口HTML::TreeBuilder::new_from_url访问网络搜索引擎，以html页面的形式获取搜索结果，解析后返回给Software_Shell呈现给用户。

注：如果采用百度搜索引擎，1）中调用接口为Net::BaiduIt；如果采用雅虎，1）中调用接口为Net::YahooIt。谷歌总是被墙，于是不用了。

3） 其他

一层模块间的其他交互与上述两个例子相似，不再一一描述。具体处理流程可以参见二层分解中相关运行设计。

2.数据依赖关系

2.2.3 接口描述

1.模块/子系统接口

1） Shell模块外部接口描述

Software_Shell模块总是在用户的驱动下，主动调用其他模块完成处理，不提供外部接口供其他模块调用。

2） Engine模块外部接口描述

Index::

@EXPORT = qw(LoadData StoreData RebuildData GetDataBase GetDataBaseKind);

Index::LoadData 载入数据库信息

Index::StoreData 保存数据库信息

Index::RebuildData 重建数据库信息

Index::GetDataBase 根据指定关键词获取对应数据库

Index::GetDataBaseKind 根据指定关键词获取对应的数据库大类，包含多个同属性的库

Search::

@EXPORT = qw(SearchData FuzzySearchData StrictSearchData);

Search::SearchData 精确搜索，限定首字匹配

Search::FuzzySearchData 模糊搜索，无限定

Search::StrictSearchData 严格精确搜索，限定全字匹配

Kino::

@EXPORT = qw(BuildContentIndex SearchContentData IsIndexExist);

Kino::BuildContentIndex 建立全文索引数据库

Kino::SearchContentData 搜索全文索引

Kino::IsIndexExist 判断全文索引数据库是否存在

Net::

@EXPORT = qw(BaiduIt YahooIt);

Net::BaiduIt 获取百度搜索引擎的网络搜索结果

Net::YahooIt 获取雅虎搜索引擎的网络搜索结果

2.进程间接口

无

2.3 第二层设计描述

2.3.1 Shell 模块

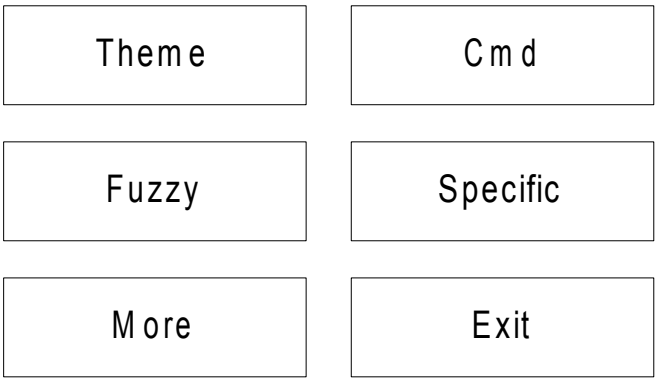


图7 二层Shell模块分解

Shell是UI模块，各个子模块之间没有调用关系，用户根据需要切换到相应视图下即可。

1.分解描述

1） Specific精确搜索界面

标识：Software_Shell_Specific

类型：二级子模块

目的：提供精确搜索功能及选项

功能列表：

- 启动与载入界面

- 管理并响应精确搜索相关控件，模式切换控件
- 支持手势操作切换到帮助界面
- 支持精确搜索选项以动画效果弹出与缩回
- 显示精确搜索结果及文件内容搜索结果
- 搜索结果初步过滤
- 结果文件/文件夹双击打开
- 保存并提供历史记录
- 输入搜索字符串时给出输入建议

涉及的需求ID：

表3 Specific模块涉及的需求ID

需求点序号	需求点名称
SRS.SEARCH.SPECIFIC.001	精确搜索模式
SRS.SHELL.GENERAL.001	交互界面整体设计

子模块：

无

2） Fuzzy模糊搜索界面

标识：Software_Shell_Fuzzy

类型：二级子模块

目的：提供模糊搜索功能及选项

功能列表：

- 管理并响应模糊搜索相关控件，模式切换控件
- 显示模糊搜索结果及文件内容搜索结果
- 搜索结果初步过滤
- 结果文件/文件夹双击打开
- 保存并提供历史记录
- 输入搜索字符串时给出输入建议

涉及的需求ID：

表4 Fuzzy模块涉及的需求ID

需求点序号	需求点名称
SRS.SEARCH.FUZZY.001	模糊搜索模式
SRS.SHELL.GENERAL.001	交互界面整体设计

子模块：

无

3） More更多结果显示界面

标识：Software_Shell_More

类型：二级子模块

目的：针对搜索结果显示更多信息，并提供排序、过滤、联网查询等辅助功能

功能列表：

- 搜索结果显示名称、地址、大小、修改时间、标识图标等更详细的信息
- 支持文件预览，根据文件类型有不同预览方式
- 支持搜索结果即时过滤并高亮匹配结果
- 支持按照大小、文件名、修改时间、关联度升序或降序排序
- 支持联网获取搜索引擎的搜索结果信息
- 按照文件类型显示/隐藏相应文件/文件夹

涉及的需求ID：

表5 More模块涉及的需求ID

需求点序号	需求点名称
SRS.SHELL.GENERAL.001	交互界面整体设计

子模块：

无

4） Exit退出界面

标识：Software_Shell_Exit

类型：二级子模块

目的：退出软件并完成相关资源回收处理

功能列表：

- 提示用户是否确认退出
- 结束进程并退出程序
- 检查是否需要更新MD5校验文件或文本库

涉及的需求ID：

表6 Exit模块涉及的需求ID

需求点序号	需求点名称
SRS.SHELL.GENERAL.001	交互界面整体设计

子模块：

无

5） Cmd命令行调试界面

标识：Software_Shell_Cmd

类型：二级子模块

目的：调试程序，测试软件稳定性，显示调试信息

功能列表：

- 覆盖软件主要功能点
- 通过命令字切换模式
- 显示调试信息

涉及的需求ID：

表7 Cmd模块涉及的需求ID

需求点序号	需求点名称
SRS.SHELL.CMD.001	命令行调试程序设计

子模块：

无

6） Theme主题切换

标识：Software_Shell_Theme

类型：二级子模块

目的：管理、切换、载入软件主题

功能列表：

- 识别主题包并提取路径和名称
- 读取主题包内素材并根据配置文件载入主题

涉及的需求ID：

表8 Theme模块涉及的需求ID

需求点序号	需求点名称
SRS.SHELL.GENERAL.001	交互界面整体设计

子模块：

无

2.界面设计

作为一款轻量级的本地文件搜索引擎，我们不希望将界面设计得如传统软件一样，左上角“File、Edit、Help……”一排菜单页，下方一列状态栏，万年灰白色的背景上面整整齐齐地罗列静态控件。作为一款本地文件搜索引擎，File、Edit等菜单中的条目几乎没有，很多搜索相关的设置选项，也并非实时都在使用，本质而言，对于搜索程序，显示一个搜索框就够了，这也是Google Desktop等的追求极致简洁的设计方式。

一些很成功的搜索软件，如Everything、Google Desktop和网络搜索引擎Baidu、Google、Yahoo等，都有一个共同的特点，就是在他们默认的（或者说主要的）搜索界面上，会在很明显的地方摆放上唯一的输入框，界面上没有其他作为搜索可选项的复选框、输入框等等。这就给用户一个很明确的信号：往这里输入你想要的。用户不会感到迷茫或不知所措，同时，这种简洁的设计也会在潜意识里鼓励用户输入一点东西试试，引导他们使用产品。

我们也希望能够借鉴这种设计思路，严格控制界面上的信息量，同时保持美观，提供方便、有效的交互体验。

说明时，以主题“iPhone”为例，其他主题控件与此保持一致。

1） Specific精确搜索

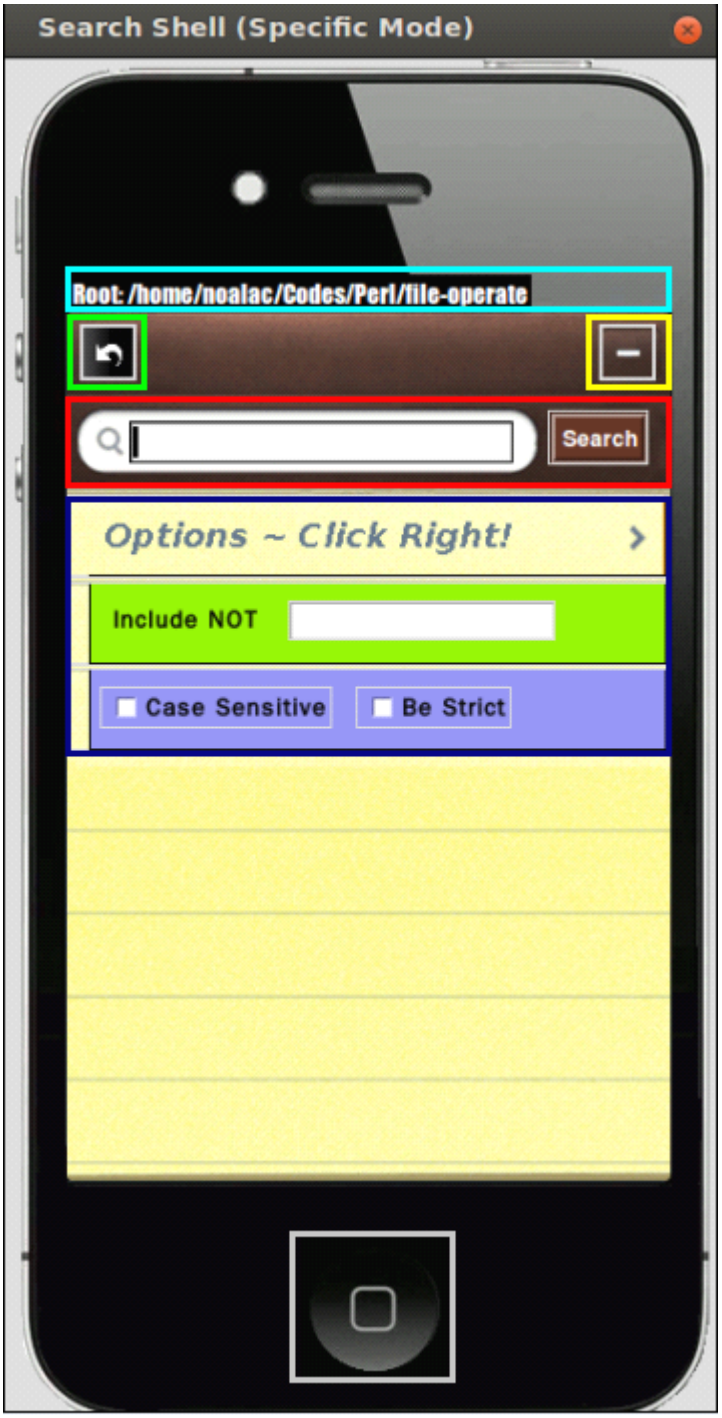


图8 精确搜索主界面设计

上图为精确搜索界面的示意图，其主要包括六大部分（以不同颜色标识说明）：

1) **输入框（及搜索按钮）**：用户可以输入希望搜索到的字符

2) **状态栏**：显示当前处于何种状态，并给出索引目录、搜索结果数量等必要信息。

3) **模式切换按钮**：精确搜索下可以一键切换至模糊搜索模式

4) **清空按钮**：完全清空当前输入的信息及结果，恢复到默认空闲状态

5) **搜索可选项**：针对精确搜索的高级搜素可选项，平时隐藏，需要时动态弹出

6) 更多信息按钮：显示搜索结果的更多更详细的信息，并提供更多辅助功能

注：搜索可选项部分默认不显示，当用户有相关需求时，鼠标从一侧移入，或点击右侧箭头，可选项则会以动态方式滑入，呈现给用户。用户如果没有输入任何东西，鼠标移出时，可选项自动以动态方式滑出隐藏。如果用户输入了信息，则可选项不再隐藏，而是一直显示（因为现在已经有意义了）。

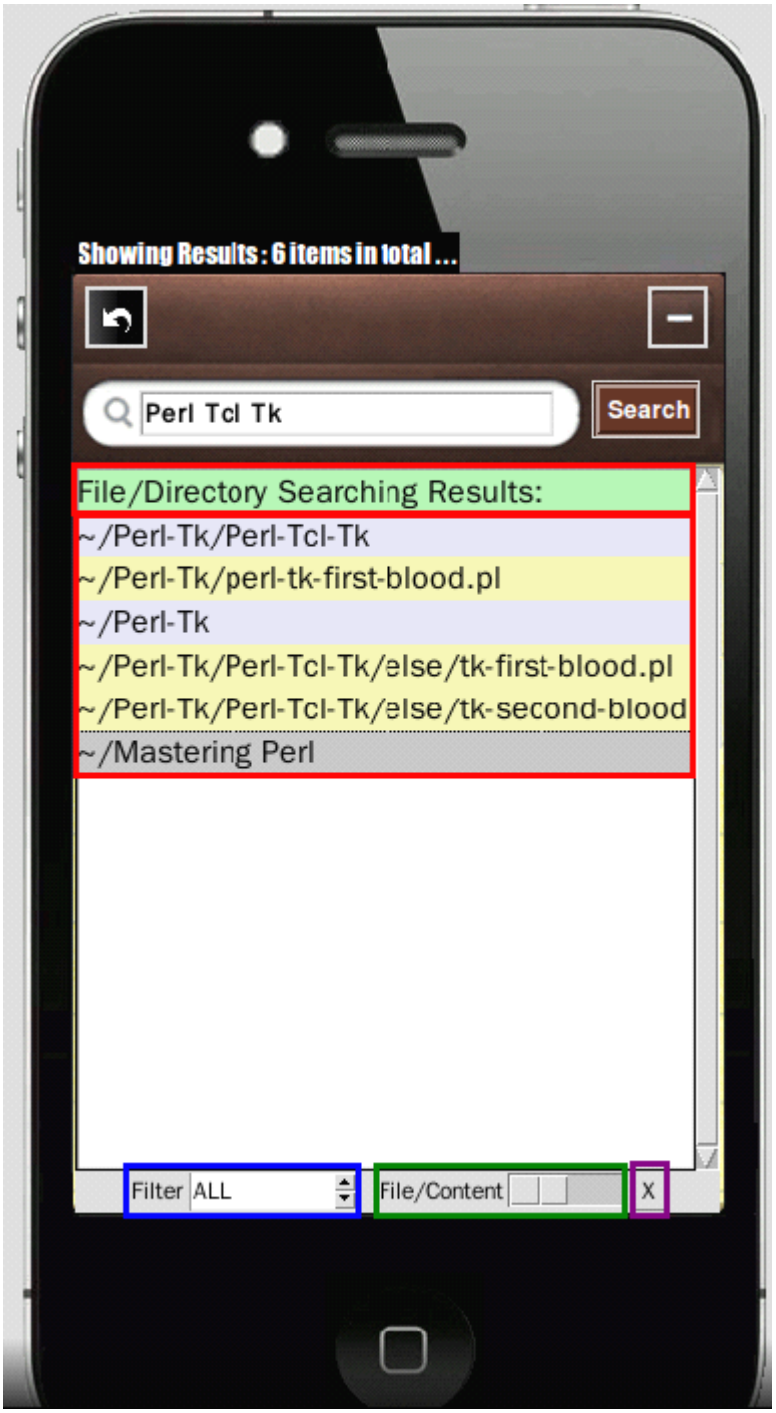


图9 精确搜索结果显示界面设计

上图为精确搜索结果显示界面的示意图，其主要包括四部分（以不同颜色标识说明）：

1) **搜索结果**：结果地址，默认按照关联度由高到低排列，索引根目录用前缀～替代

2) **状态栏**：简单过滤器。根据选择可以直接过滤出文件夹、文件、二进制或文本文件

3) **文件名/内容搜索结果切换**：可以一键在文件名搜索结果和文件内容搜索结果中切换

4) **关闭按钮**：仅仅关闭结果显示界面，而不清空搜索时输入的信息（与全部清空不同）

注：搜索结果中的文件以黄色底色显示，文件夹以蓝色底色显示，结果框中的第一行为提示信息，注明当前显示的结果类别。双击文件名可以打开相应文件，双击第一行提示信息可以打开索引根目录。如果没有结果信息，则输出“No Information Available”，并以红色底色显示。

2） Fuzzy模糊搜索

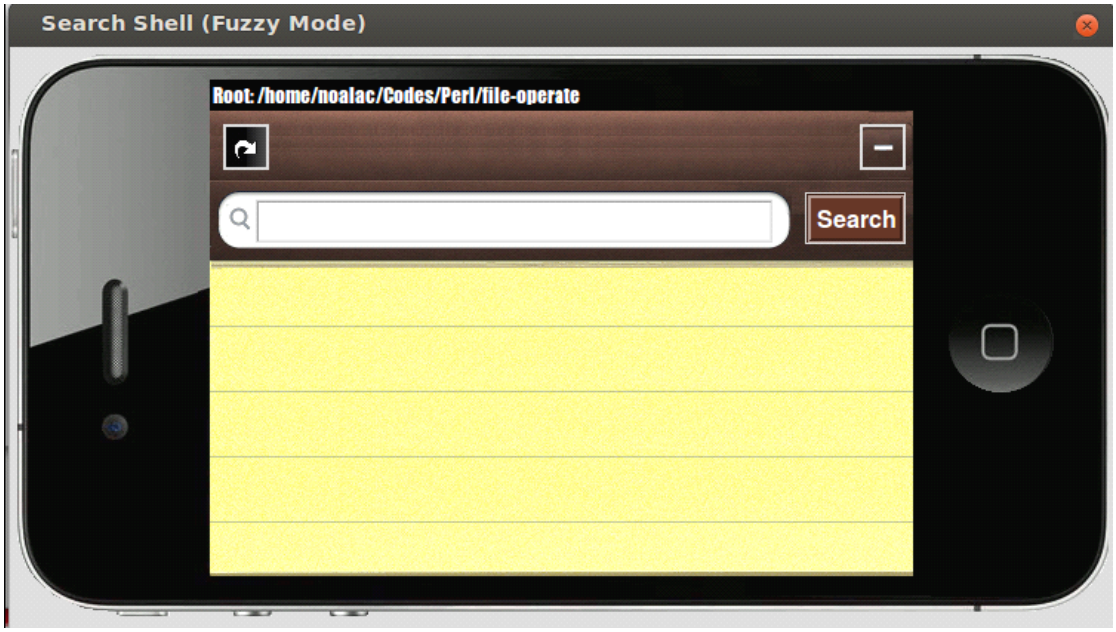


图10 模糊搜索界面设计

与精确搜索界面类似，保持界面UI的统一性。但是模糊搜索仅仅提供搜索框，不提供高级搜索的可选项。

3） More更多信息

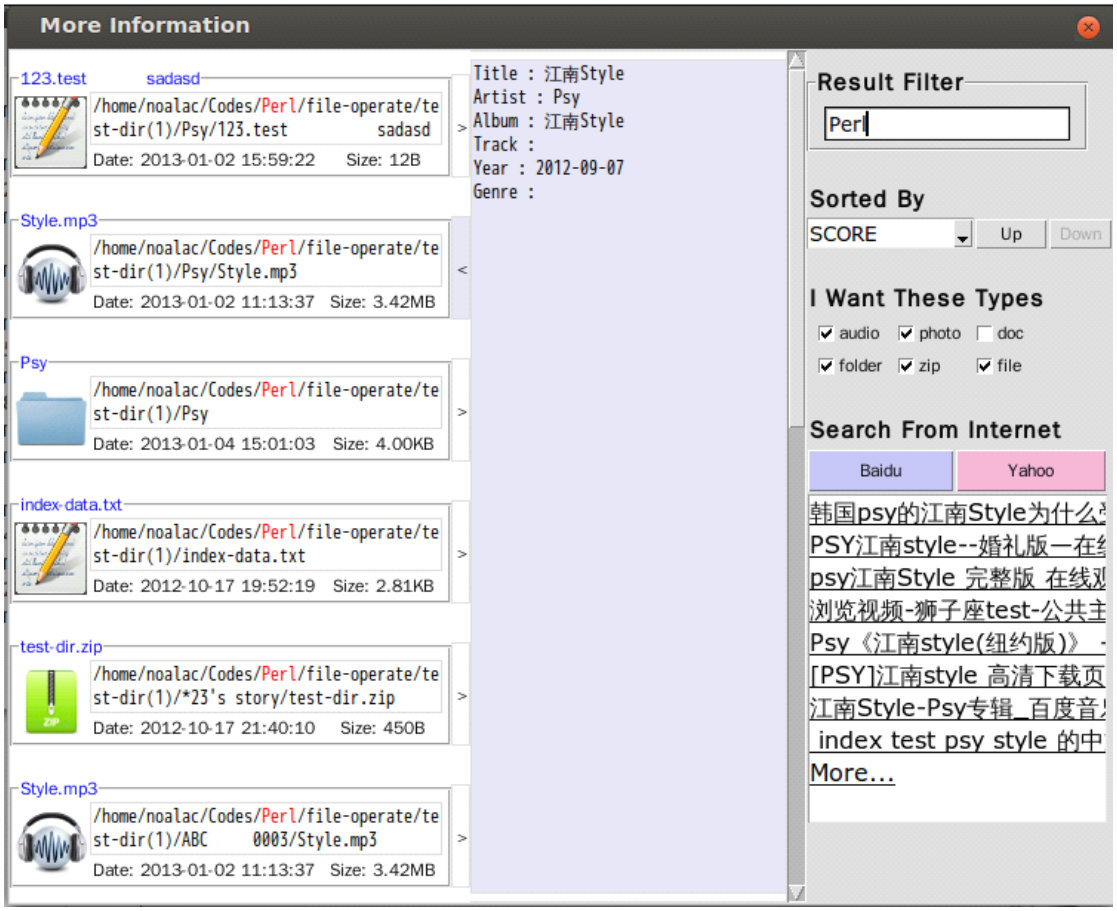


图11 更多信息界面设计

上图可以很清晰地分为三大部分：

1) 左侧：结果信息。具体请见下方详细分解。

2) 中间：预览框。默认为白色底色与整体UI底色相同，当用户预览内容时，变为浅蓝色底色，与扩展箭头底色相同。

3) 右侧：结果控制器。

   a) Result Filter：即时过滤器，匹配结果以红色高亮显示

   b) Sorted By：排序器，可以根据关联度等进行升序或降序排序

   c) I Want These Types：结果类型过滤器，显示/隐藏某一大类的文件/文件夹

   d) Search From Internet：网络分析器，获取网络搜索引擎的内容作为参考

注1：预览框中的信息根据不同文件类型有不同内容。对于audio类型文件，显示ID3标签信息（音轨名、专辑名、艺术家、时间等）；对于photo类型的文件，可以支持小图预览显示；对于doc类型的文件，显示结果为首行显示总页码，之后显示文档前十页；对于folder类型的文件夹，显示文件夹下的所有文件/文件夹；对于zip类型的压缩包，显示压缩包内文件内容；对于file类型的文件，直接显示前100行内容。

注2：网络分析器中的内容可以双击打开，会直接通过浏览器定位到相应网页，若点击最下方的“More…”，则通过浏览器打开搜索引擎的搜索结果页面。

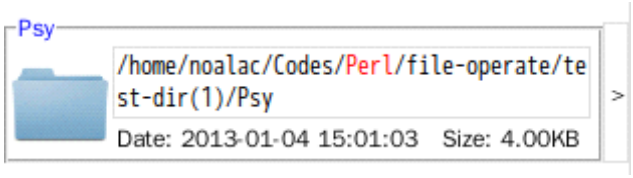


图12 更多信息界面单个条目设计

更多信息界面左侧为结果信息，其中每一个条目的具体显示方式也是由我们自己定义的（默认的Listbox不可能支持这么精美复杂的元素）。包括文件名、直观的标识图标、文件路径、修改日期和大小，右侧的按钮点击可以预览内容。双击图标可以直接打开文件。

每个条目都包含3个Frame、1个LabelFrame、1个装载图片的Label、2个文本Label、1个Text、一个Button。通过Pack打包显示。各个控件的树形结构图如下所示。

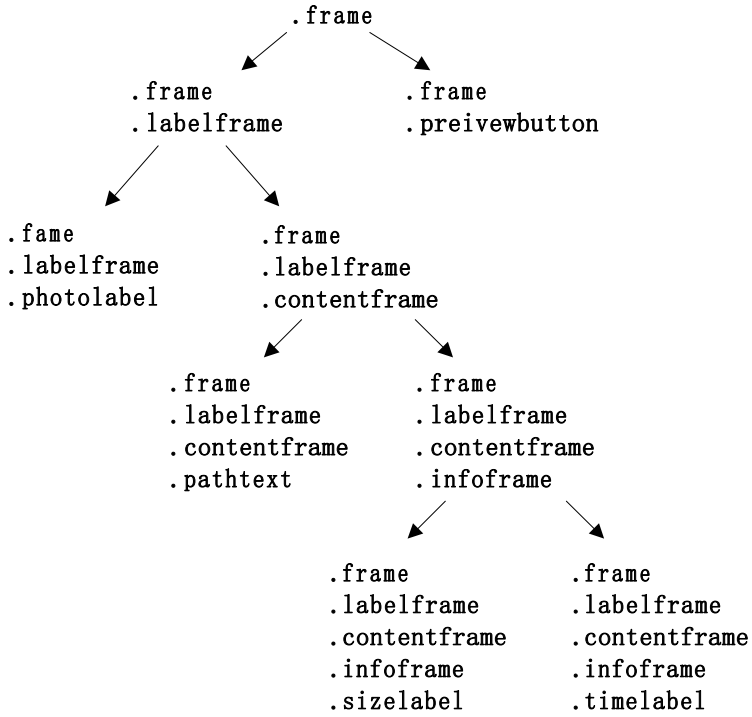


图13 更多信息界面单个条目控件树形图

使用多个Frame作为框架，是为了将各个控件自动对齐，而不必自己按照像素或其他单位长度给控件进行限制，人工手动排齐。每个条目及其下的控件通过多级匿名数组管理，具体请参见本节数据组织部分。

3.数据组织

1） 搜索结果数据结构

搜索结果有三个全局数据结构保存，分别为一个哈希表和两个列表。

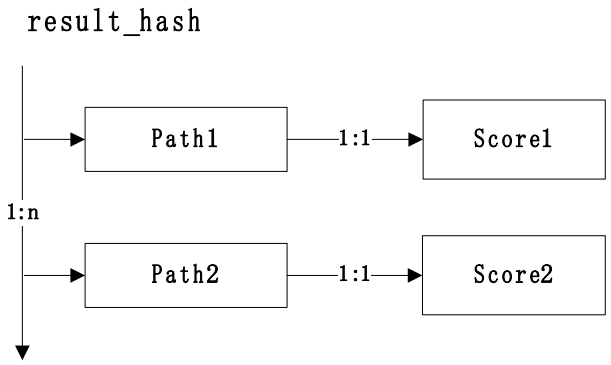


图14 全局结果哈希

结果哈希表result的Key为结果地址，Value为该地址对应的关联度分数。两个列表分别为按照关联度降序排列的结果sorted_file，以及文件内容搜索结果content_file。

2） 更多信息结果数据结构

更多信息的结果保存在另一个独立的列表more_file中，不与sorted_file复用是因为我们会根据过滤需要直接增删more_file中的数据，也即，more_file中的内容就是显示在更新信息界面的内容。所以more_file在需要时会从sorted_file中恢复数据。

3） 更多信息显示条目数据结构

更多信息界面的条目及其下的控件通过多级匿名列表保存。

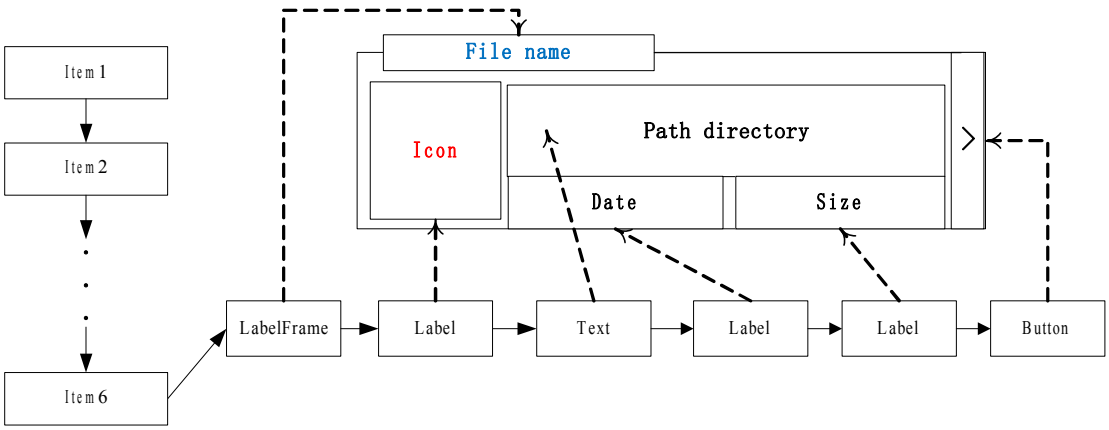


图15 更多信息条目数据组织

一级列表保存每一个Item的地址，不需要知道具体地址是什么，只要保证能通过头节点顺序访问到即可。每个Item指向一个列表，列表记录六个控件的地址，分别为文件名（LabelFrame）、标识图标（Label）、结果地址（Text）、修改日期（Label）、文件大小（Label）和预览按钮（Button）。同样不需要知道具体名称或地址，只需要保证能按照下标访问到即可。这样就能统一管理更多信息界面的主要控件并进行修改。

4.功能实现

1） 排序处理

本软件涉及到多种排序需求，例如文件名、大小、日期、关联度排序等。但实际上可以分为两大类。

a） 第一类排序

需求：已知需要排序的目标，直接进行升序或降序排列。

实现：这是最基本的需求，包括对日期、大小和文件名排序。其实每种排序本质上都是通过两个数之间进行比较，根据结果（正、负或零）在进行新一轮的两两比较。其核心算子可以表示为

其中$a为参数1，$b为参数2。中间的运算符<=>定义如下：如果$a>$b，则返回1；如果$a<$b，则返回-1；如果$a=$b，则返回零。Perl中内建有sort函数进行排序，但我们可以自己定义上述算子实现自己的需求。

对于文件名排序，则直接将文件名作为入参即可，之后sort便会按照字典序排序。

对于文件大小和日期，注意，这里我们无法直接操作这些数据，因为我们的根本目标是排**列**文件名，而不是排**列**大小或日期，大小或日期只是我们排**序**的依据。对于这种情况，我们的实现方法是，创建一个临时哈希，Key为文件名，Value为对应的大小或日期，数据即时获取（Everything就是即时获取的，速度很快）。重新定制算子为：

为了让算子内部可以看到外层临时哈希，既可以把哈希作为全局变量（不推荐，临时使用，全局变量开销大不安全），也可以将算子定义为匿名内联函数（推荐）。这个

算子的意义是，实际比较由文件名确定的大小或日期，但是最后排列的对象是文件名。

以上是升序的排序算子，降序的话，不必按照升序排列完再reverse，太慢了，直接把$a和$b的位置调换一下就完了。降序算子如下。

b） 第二类排序

需求：排序整个过程均不知道排序目标，但是已知一个按照目标排好序的超集，现要对其子集排序。

实现：这个需求可能没描述清楚，举个例子：在完全不知道电信系全体同学分数的情况下，给你全体同学的名次，要求你排出你们班同学的名次。

这个需求出现在更多信息界面对关联度的排序中，more_file是显示用的列表，sorted_file是按照关联度降序排列的列表，但是其中没有记录具体的关联度分数。

最慢的方法显然是，依次从sorted_file列表中开始找，每个均在more_file里查是否存在，如果存在则记录一下，设sorted_file长度为m，more_file长度为n（n<=m），这这种方法的时间复杂度为**$o(m^n)$**，显然比较慢。经测试，如果结果文件量级上千，就能在排序过程中明显感受到延迟。

我们采用的方案为，用空间换时间，对more_file建立一个临时哈希，以其中的地址为Key，对应的Value不重要。然后遍历sorted_file，这样如果遍历到的文件存在于临时哈希中，则可以直接定位出来，压入排序好的列表，如果该文件在临时哈希中没有定义，则跳过。这样建立临时哈希用时n，遍历用时m，总时间复杂度为**o(m+n)**。

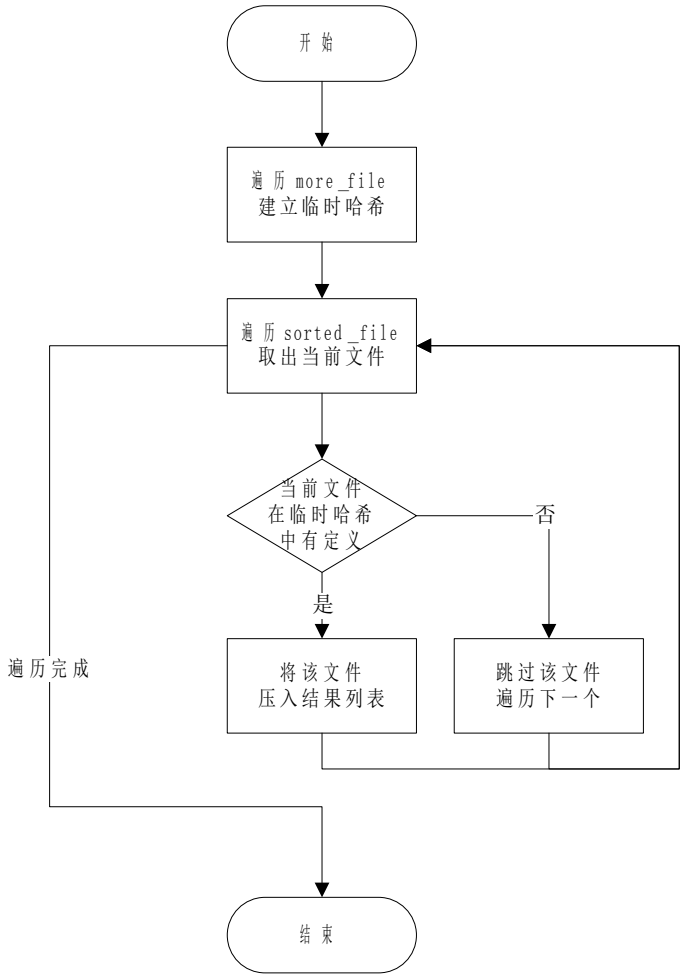


图16 第二类排序流程图

2） 更多信息

更对信息界面是对主界面列表框的扩展。默认的Listbox只能插入一行字符串，不能同时加入图标、多行文本、高亮关键词等功能，于是我们就自己实现了一个。

根据上面界面设计和数据组织的介绍，单个Item的结构和更新方式都已经确定，现在要考虑的是如何排放这些Item。

最初的方案是，新建一个画布Canvas，将每个Item作为对象嵌入Canvas，这样还有个好处，就是可以自然而然地通过画布自带的滚动条完成整个信息的滚动功能。但是结果失败了，原因是我们定义的Item解构复杂，每个Item创建的过程都比较长（人可以感受到），那么几十条、上百条Item依次创建就要等几秒钟，上千条结果要几十秒，这显然无法接受。

改进的方案思路源自于新浪博客的图片功能。众所周知，图片载入也需要一定时间，新浪每次载入固定数量的图片，用户在浏览完一屏后可以点击“获取下N条”显示下一屏。我

们发现如果创建六七个Item还是很快的，于是仿照此思路实现了一下，结果类似：

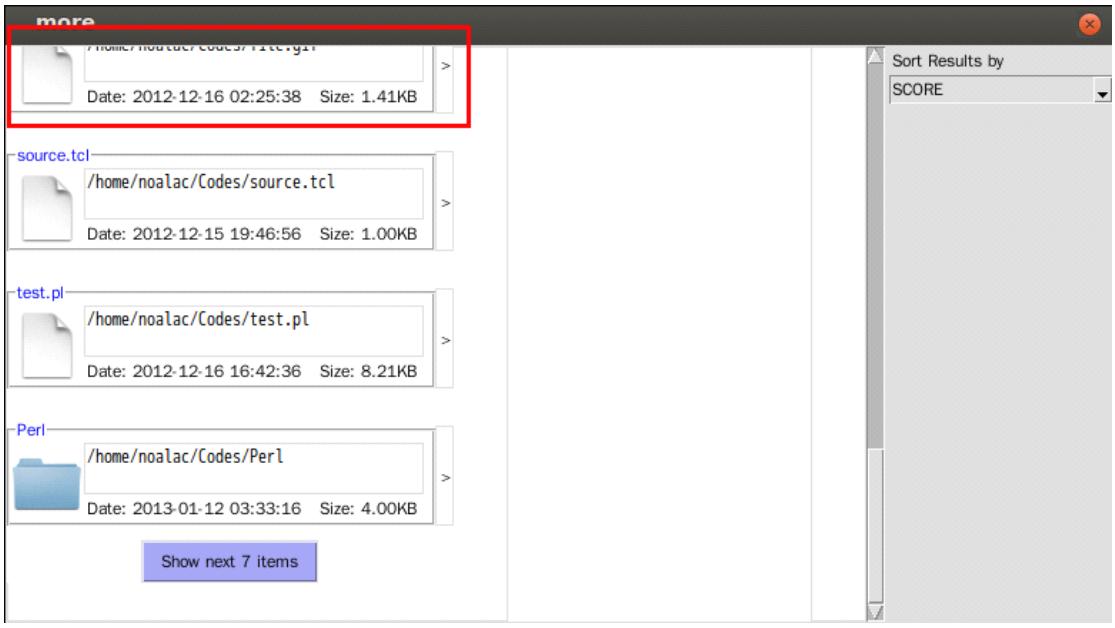


图17 更多信息失败显示方案

通过下方蓝色的 Show Next 7 items按钮，可以载入之后的七条Item。注意图中红色框中的内容，可以发现当前窗口下最上面的Item只显示了一半，这说明这是真真正正把Item嵌入到画布中，然后通过滚动条可以任意定位。

但是这个方案最后也失败了，原因有很多，例如Item创建起来还是不够快，比较占资源，排序后的结果无法显示完全等，但是最致命的问题出在销毁时的速度。在进行新一轮搜索前，需要把之前的信息销毁，但是Item创建慢销毁也很慢，导致下一轮搜索需要为了清空上次的搜索数据而等待几十秒，这也是不可接受的。

最后我们采用的方案也还算是老思路，就是在初始化时便创建好6个Item备用，不再创建更多Item，每次更新都仅仅把各个Item中的内容更新即可。这样直接结果是不能利用已有的滚动条了，于是我们重载了滚动条的操作和事件，原来的滚动条失去自身功能，实际上已经变为一个上下滑动的长方形而已。

首先根据Item个数（即6）与搜索结果数量的比值，计算出滚动条的长度。

如果是在搜索结束后第一次打开更多信息界面，还需把滚动条坐标值为0。

将鼠标左键释放和鼠标左键移动事件绑定到滚动条上，事件处理中记录滚动条当前位置占总长的比例，将该比例乘以结果总数量，取整后，即为更新起始的下标Index，将Index及之后的5个结果信息更新入Item，完成操作。这样每次滚动的最小单位为一个Item。

其示意图如下。

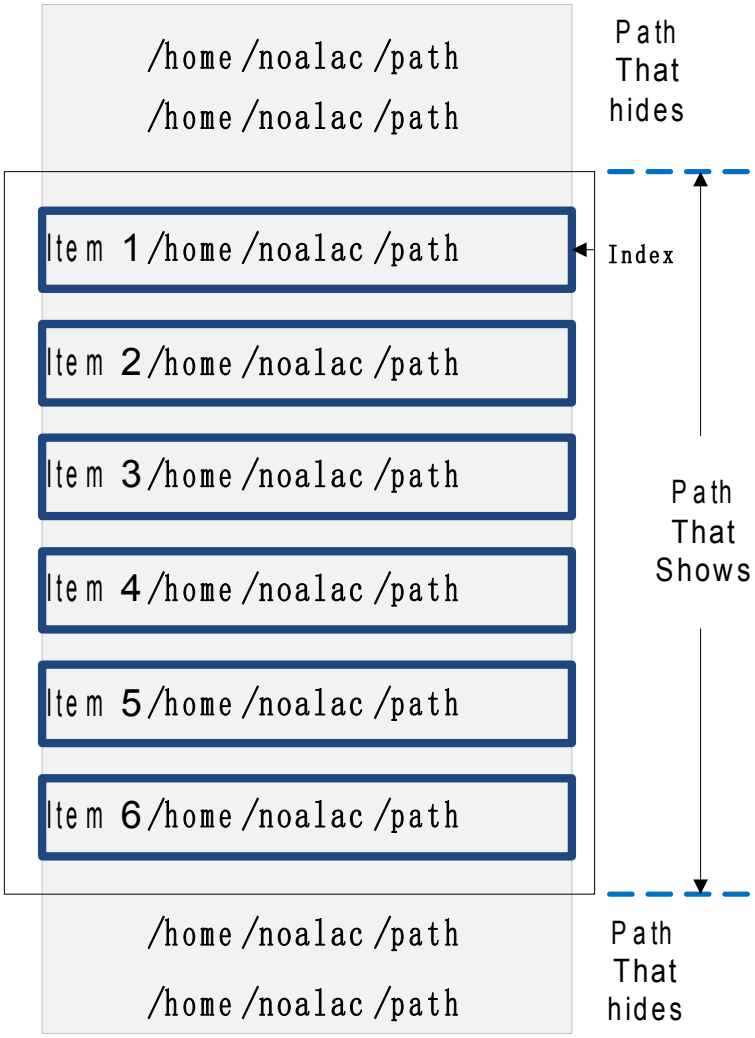


图18 更多信息现行显示方案示意图

3） 预览功能

预览功能的思路来自于谷歌搜索引擎，在打开网页之前就可以事先预览部分内容，十分方便。虽然本软件允许用户双击直接打开文件/文件夹，但是可能因为打开时间较长、或用户仅仅希望预览一点信息之后进行下一次搜索，不需要完全打开文件。基于这些需求，我们开发了预览功能，同时，对于不同类型的文件，尽量提供多样化的信息，帮助用户更好地完成搜索。

根据不同类型提取相关信息：

- Audio：音频文件提取ID3标签信息（支持部分格式）
- Photo：图片支持小图读取预览
- Doc：提供总页数及前10页内容（仅支持PDF格式）
- Folder：显示该文件夹下的文件和文件夹

- Zip：显示压缩包内容（仅支持ZIP格式）
- File：其他文件类型读取前100行内容预览（因格式不明，可能有乱码）

如果该文件或文件夹已被删除或移动，则在预览框中给出相应提示信息：

“This file/folder has been moved or deleted!”

信息的提取要借助相关模块，以文本内容提取出来后进行相关处理十分普通，没什么特别。预览功能麻烦的地方在一个很小的地方：打开/关闭预览的按钮。

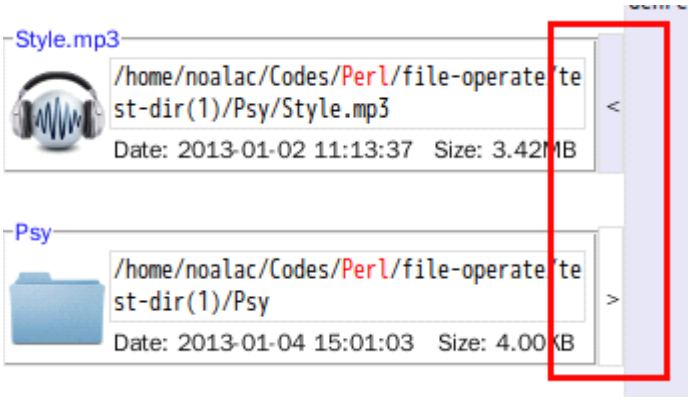


图19 预览按钮示意图

预览按钮有两种状态，激活状态（’>’）和非激活状态（’<’），且打开或关闭的状态是持续的，必须记忆，这样在拖动滚动条时，预览状态也会同步移动。当前预览Item移出当前显示窗口时，预览框内容保持（因为预览还未关闭！），但是预览按钮已全部处于非激活状态。当拖动滚动条返回预览Item所在位置时，该Item对应的预览按钮应该显示为激活状态

我们的实现方法是，设置一个标志位，last_prev记录激活预览的文件地址，因为文件地址是唯一的，类似身份ID，这样可以唯一标识结果。

在滚动过程中（即更新过程中），6个Item会逐一与last_prev比较，如果相等，则将该Item的预览按钮激活，否则非激活。

同时，每次在打开另一个文件的预览功能时，都需要遍历6个Item，查找是否同窗口下已经存在有激活的预览按钮，若有将其置为非激活。这个细节很重要。

4） 搜索建议

搜索建议功能也来自于网络搜索引擎，例如谷歌、百度等。在用户输入搜索关键词时，搜索引擎会即时给用户一些建议，帮助用户完成搜索过程。

由于本软件建立的是倒排索引，所以关键词的提取是天然实现的，我们将索引关键词作为建议的词库。难点在于给出建议的时机。

我们将特定按键的释放事件绑定给输入框，绑定按键包括A-Z二十六个字母、0-9十个数字、Backspace、Del和空格功能键。每次事件响应时提取当前输入框内的最后一部分单词进行匹配，匹配结果输入建议框，并将建议框置于顶层显示，如果没有匹配结果或输入最后一部分为空白字符（或空），则清空建议框并下降。

双击建议框中的某个条目，直接将最后一部分单词替换为建议内容。

具体实现细节请参见search-shell.pl中give_suggestion函数。

图20 建议功能示意图

5） 历史记录

历史记录复用了建议框，区别在于，历史记录在输入框的内容为空（或全为空白字符）时发生，此时修改状态栏为Show Search history…，若没有历史记录，则修改状态栏为No history…，历史记录显示后，用户输入任何可见字符都会使历史记录消失。

注意历史记录显示时要消除重复项，注重细节！历史记录按照时间顺序由新到旧排列。

双击历史记录中的某个条目，直接将输入框的内容插入该条历史记录。

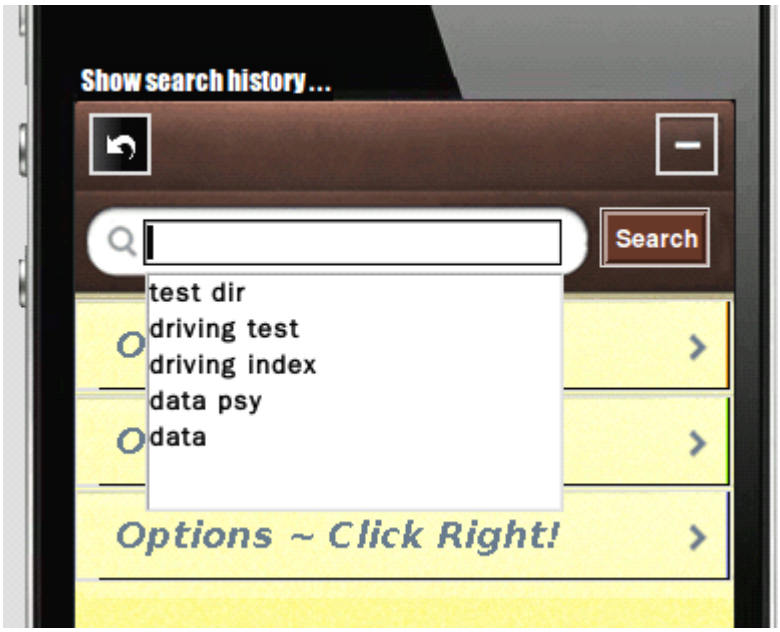


图21 历史记录功能示意图

6） 选项动画

选项动画是很酷并且很实用的功能。当用户移动鼠标从右侧进入选项时，该选项会以渐变的颜色动态弹出，用户把鼠标移出选项时，该选项又会以渐变的颜色动态隐去。

如果用户在弹出的选项用进行了输入或修改，则该选项不再隐去，一直保留直到清空。

选项在最初隐藏的目的是不给用户带来困扰，希望主界面以尽可能简洁的形式呈现给用户，并仅提供唯一的明确的输入框（即原始搜索输入框）。

遗憾的是，Tk并没有提供任何动画功能，所以我们必须自己实现动画效果。首先创建16个等宽但不等长的矩形，并按照长度赋予不同深度的颜色，依次排放在界面底层（不可见）。给16个矩形顺序打上标签方便调用。当需要弹出动画时，按照长度由短到长、颜色从深到浅依次将一个矩形置于顶层，每次操作间隔20毫秒（人眼视觉暂留的极限），其效果便是非常酷的动画效果。动态隐去的过程与弹出相反。

选项动画示意图如下。

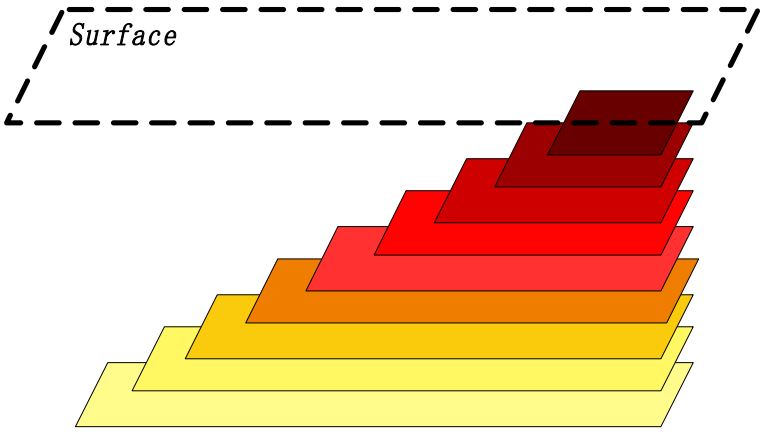


图22 选项动画初始状态示意图

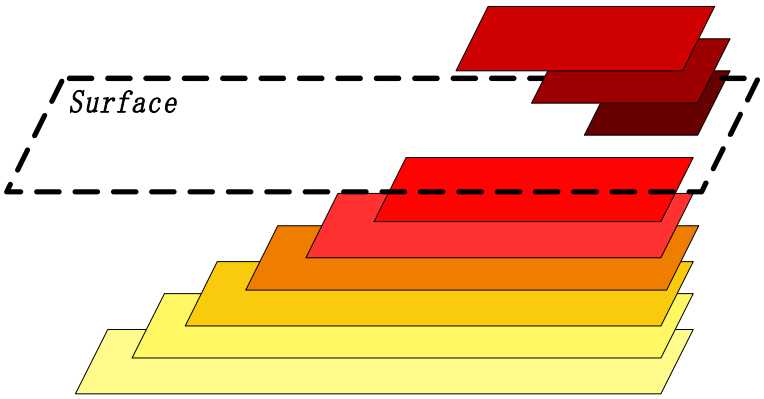


图23 选项动画渐变示意图

7） 即时过滤

即时过滤是另一个很酷的功能，用户的搜索结果可能很多，但是用户又不希望更改关键词重新搜索，此时用户可以使用即时过滤功能，在Result Filter框中输入希望存在的字符串，软件会即时把不符合条件的结果从显示的结果中去掉，并把剩下的、符合条件的结果中的匹配项用红字高亮。当用户回退Result Filter框中的字符串时，之前被过滤掉的结果又会重新恢复入显示结果中，就像从没发生过过滤操作一样。

即时过滤的功能依赖于三个数据结构more_filtered_file列表（记录已经被过滤的文件）、more_filter_string字符串（记录过滤字符串）、more_filter_curlen标量（记录上次过滤字符串长度），将按键释放事件绑定到Result Filter框，实现方法比较巧妙。

核心思路是一个互斥三种操作：

互斥。more_filtered_file与more_file中的内容互斥，即一个文件要么存在于more_filtered_file，要么存在与more_file。

三种操作。用户释放按键时，当前过滤框中内容的长度与上次记录的长度相比，只可能有三种情况，每种情况都对应相互不覆盖的操作。

- 若当前长度大于上次长度：意味着用户在过滤框中输入了字符，此时要遍历more_file列表，把不匹配的文件取出，压入more_filtered_file
- 若当前长度小于上次长度：意味着用户删除了过滤字符，此时要遍历more_filtered_file列表，把当前匹配的文件取出，压入more_file，并重新排序。
- 若当前长度等于上次长度：意味着用户进行了其他控件的操作，例如隐藏了某一类型的文件，这个操作会影响到过滤功能。原因是，隐藏操作仅仅会处理more_file列表中的文件，某些文件可能由于过滤到了more_filtered_file中，导致没有隐藏，这样在用户回退过滤字符串时，该文件就会出现（本不应该出现！）。所以当长度相等时，需要自行遍历more_filtered_file，去掉隐藏类型的文件。这是一个很重要的细节！

实现细节可以参考search-shell.pl中的more_filter_result函数

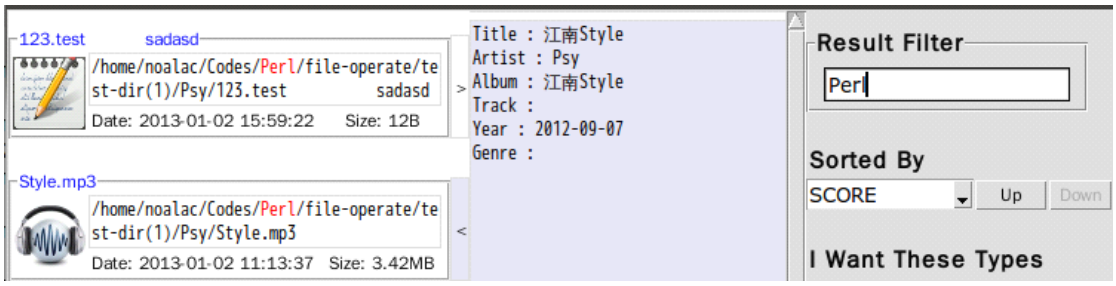


图24 即时过滤示意图

8） 字符转码

字符转码问题是跨平台软件面临的最大难题之一，现在编码格式多且杂，并且每个操作平台都有各自的默认编码格式，更加大了字符转码的难度。

字符转码的本质很简单，就是从一种编码格式解码（Decode），然后再编码为另一种格式（Encode），其中的过程各个语言都有相应的接口和模块完成。难点在找出两头的编码格式具体是什么。

每个操作系统都有默认的编码格式，如Ubuntu/Linux默认为UTF-8，Windows默认为GBK（gb2312）等等，我们根据平台设定各自的默认编码，这

样能解决大部分情况下的转码问题。但是，当用户将系统默认的编码格式修改，或者在一个未知的平台上操作时（虽然这几乎不可能），设定默认编码的方法便会失效。

根据本软件自身的设计，我们想了一个比较灵活又略显智能的自动检测编码格式的方法。

首先需要说明的是，检测编码的工作交由外部模块Encode::Detect::Detector完成，这部分功能大多数语言也都有相应接口。但是仅有编码检测功能没太大用，因为在软件真正开始遇到转码任务前，必须已知该操作系统当前的编码格式，如果每次读取文件内容或文件目录名前都进行编码检测，开销又过大，且大多是情况下没有必要。

软件自身有两个特点：

- 搜索前必须建立索引（文件搜索引擎大都有的特点）
- 任务均以某个功能脚本启动，以命令行参数的形式传递参数（几乎是本软件独有）

建立索引时，根目录地址以命令行参数形式传入功能脚本。

2.3.2 Engine 模块

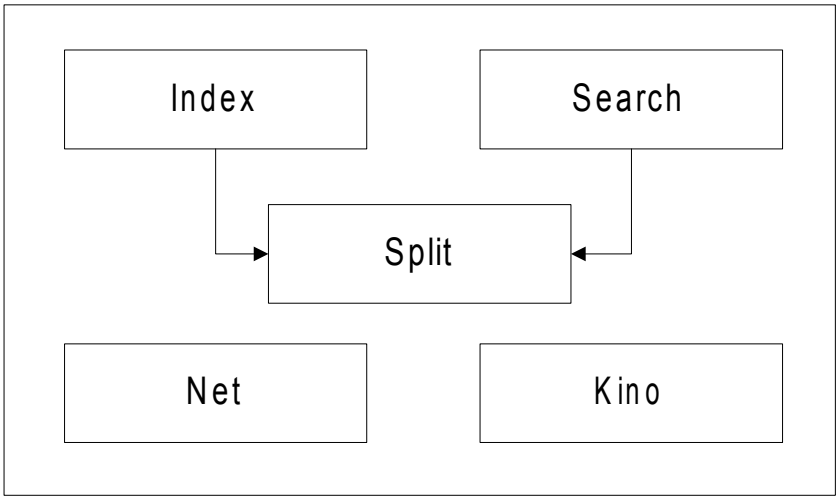


图25 二层Engine模块分解

Index和Search模块均需要调用Split模块完成分词功能，Net和Kino模块功能独立，Engine模块整体对外提供接口的模块为Index、Search、Net、Kino，完全的内部模块为Split。

1.分解描述

1） Index模块

标识：Software_Engine_Index

类型：二级子模块

目的： 建立索引及数据库基本信息

功能列表：

- 针对文件名及文件夹名建立倒排索引
- 二进制数据库、文本库和MD5校验文件的创建、更新及管理
- 数据库基本信息的管理db-info
- 数据库群的选择与重定向

涉及的需求ID：

表9 Index模块涉及的需求ID

需求点序号	需求点名称
SRS.INDEX.DB.001	初始化本地数据库
SRS.INDEX.DB.002	读取本地数据库
SRS.INDEX.DB.003	存储本地数据库
SRS.INDEX.DB.004	删除本地数据库
SRS.INDEX.DB.005	重建本地数据库
SRS.INDEX.DB.006	载入本地数据库
SRS.INDEX.DATA.001	绑定索引与数据库
SRS.INDEX.DATA.002	添加新索引
SRS.INDEX.DATA.003	清空索引数据
SRS.INDEX.DATA.004	索引关键词的合并
SRS.INDEX.DATA.005	建立压缩文件索引

子模块：

无

2） Search模块

标识：Software_Engine_Search

类型：二级子模块

目的： 搜索关键词并对结果的关联度评估

功能列表：

- 精确搜索

- 精确搜索严格模式
- 模糊搜索
- 搜索结果关联度评估打分

涉及的需求ID：

表10 Search模块涉及的需求ID

需求点序号	需求点名称
SRS.SEARCH.SPECIFIC.001	精确搜索模式
SRS.SEARCH.FUZZY.001	模糊搜索模式
SRS.SEARCH.SCORE.001	搜索结果关联度评估
未分解	精确搜索的严格模式

子模块：

无

3） Kino模块

标识：Software_Engine_Kino

类型：二级子模块

目的： 全文数据索引的建立及文件内容搜索

功能列表：

- 根据文件类型提取文件内容并全文索引
- 文件内容关键词搜索
- 创建并维护全文数据库

涉及的需求ID：

表11 Kino模块涉及的需求ID

需求点序号	需求点名称
SRS.PLU.BUILD.001	文件内容索引的建立
SRS.PLU.BUILD.002	文本和PDF文件内容索引处理
SRS.PLU.BUILD.003	HTML文件内容索引处理
SRS.PLU.SEARCH.001	文件内容的搜索

子模块：

无

4） Split模块

标识：Software_Engine_Split

类型：二级子模块

目的： 中英文及数字分词

功能列表：

- 中文分词
- 英文分词
- 数字分词

涉及的需求ID：

表12 Split模块涉及的需求ID

需求点序号	需求点名称
SRS.SPLIT.WORD.001	分词处理
SRS.SPLIT.KEYWORD.001	针对数字分词
SRS.SPLIT.KEYWORD.002	针对英文分词
SRS.SPLIT.KEYWORD.003	针对中文分词
SRS.SPLIT.EXTEND.001	针对扩展名处理

子模块：

无

5） Net模块

标识：Software_Engine_Net

类型：二级子模块

目的： 获取网络搜索引擎的搜索结果

功能列表：

- 获取百度搜索引擎的搜索结果并解析
- 获取雅虎搜索引擎的搜索结果并解析

涉及的需求ID：

表13 Net模块涉及的需求ID

需求点序号	需求点名称
SRS.SHELL.GENERAL.001	交互界面整体设计

子模块：

无

2.数据组织

1） 索引数据结构

本软件采用倒排索引（InvIndex）的方式，在符合倒排索引的基本格式前提下，记录了自身关联度评估需要的参数。具体而言，本软件的索引结构为“哈希的哈希”（HoH）。首先，与GDBM数据库进行绑定，得到外层哈希。向该哈希写入数据即向数据库写入数据。外层哈希的Key为经过分词处理后的关键字Key，Value为内层哈希的引用。内层哈希的Key为某文件/文件夹的地址Path，Value为关键词的词频Frequency，将用于关联度评估。

其结构示意图如下

$$db \rightarrow \begin{cases} Key1 \rightarrow \begin{cases} Path1 \rightarrow Frequency1 \\ Path2 \rightarrow Frequency2 \\ ..... \\ PathM \rightarrow FrequencyM \end{cases} \\ Key2 \rightarrow ..... \\ ..... \\ KeyN \end{cases}$$

图26 索引HoH结构

说明：一个关键词Key对应多个地址Path，每个Path中Key出现的次数记为Frequency。

2） 多数据库的选择与重定向结构

多数据库的目的是加快搜索的速度。GDBM为单文件数据库形式，文件体积过大会影响读写速度，影响搜索效率；同时，精确搜索（非严格模式）和模糊搜索由于搜索范围的需要，都会遍历整个库的关键词，精确搜索（非严格模

式）在已经附加首字母匹配的限定条件下，没有必要再去遍历其他字母开头的关键词（因为肯定不会匹配）。所以，需要适当对原单个数据库进行拆分，缩小遍历范围。

数据库拆分的思想源自于词典，一个词典中，以各个字母开头的单词数相差很大，我们在背单词的时候，为了平均工作量，往往把26个字母开头的单词均分成几个部分，这样背每部分单词时花费的精力既不会太多也不会过少。文件名也有相同的特性。我们首先对**Ubuntu /usr**目录下的文件进行了索引，将收集到的关键词按照首字母分类，中文和数字单独分为一类。根据关键词数量，分成以下七类：

- 中文fallback
- 数字number
- 英文字母garvk开头
- 英文字母cmbnq开头
- 英文字母dlhxzy开头
- 英文字母pieuj开头
- 英文字母sftwo开头

分成七个类的原因很大程度上受到了西方小说人物伏地魔的影响。

建立一个重定向哈希负责各个关键词去往数据库的分发。

```
my %redirect = (
    g => 'garvk', a   => 'garvk', r => 'garvk', v   => 'garvk', k => 'garvk',
    c => 'cmbnq', m => 'cmbnq', b => 'cmbnq', n => 'cmbnq', q => 'cmbnq',
    d => 'dlhxzy', l => 'dlhxzy', h => 'dlhxzy', x => 'dlhxzy', z => 'dlhxzy', y => 'dlhxzy'
    p => 'pieuj', i => 'pieuj', e => 'pieuj', u => 'pieuj', j => 'pieuj',
    s => 'sftwo', f => 'sftwo', t => 'sftwo', w => 'sftwo', o => 'sftwo',
    0 => 'number', 1 => 'number', 2 => 'number', 3 => 'number', 4 => 'number',
    5 => 'number', 6 => 'number', 7 => 'number', 8 => 'number',9 => 'number',
);
```

如果在重定向哈希中没有定义，则进入fallback外文数据库。

3） 数据库信息结构

软件把数据库索引信息与数据库的基本信息分开存放，因为使用对象不同。数据库索引信息由Index或Search调用，用于增删索引数据或检索关键词。而数据库基本信息（db-info）存储索引根目录、索引创建时间、索引地址数量、关

键词列表等信息，几乎Shell和Engine下模块都会按需取用。

处于加密性考虑，数据库基本信息采用独立的DBM文件为载体，不使用明文保存，其中的Key-Valu对应关系如下：

$$db\_info \rightarrow \begin{cases} ROOTDIR \rightarrow 根目录 \\ MODTIME \rightarrow 文件索引创建时间 \\ CONTENT \rightarrow 全文内容索引创建时间 \\ TOTALNUM \rightarrow 索引地址总数 \\ ALLKEY \rightarrow ref(关键词) \end{cases}$$

图27 数据库基本信息结构

ALLKEY对应的Value为一个列表的引用，列表内容为全部关键词。

3.功能实现

1） 多模式搜索

软件设置了两个搜索模式页面——精确搜索和模糊搜索。其中精确搜索包含严格模式（Be Strict）选项，共三种分级。

- 精确搜索严格模式

限定全字匹配。对输入字符串分词处理后，以关键词为Key直接从相应数据库中取值。搜索速度最快，要求最严格。直接以词频为关联度得分，父目录关联度加分比例为0.5。支持“^$?*”通配符搜索（^$通配符无效）。

- 精确搜索模式

限定首字匹配。对输入字符串分词处理后，以遍历相应数据库中的元素，限定首字母开头必须匹配，搜索速度适中，要求适中，为默认搜索模式。父目录关联度加分比例为1.0。支持“^$?*”通配符搜索（^通配符无效）。

- 模糊搜索模式：

无限定。对输入字符串进程原子化处理，分离为尽可能小的碎片，并试图还原英语单词的词根，再分词处理后，遍历大类数据库中的元素，搜索速度最慢，要求最宽松。父目录关联度加分比例为1.0。支持“^$?*”通配符搜索。

搜索过程流程图如下。

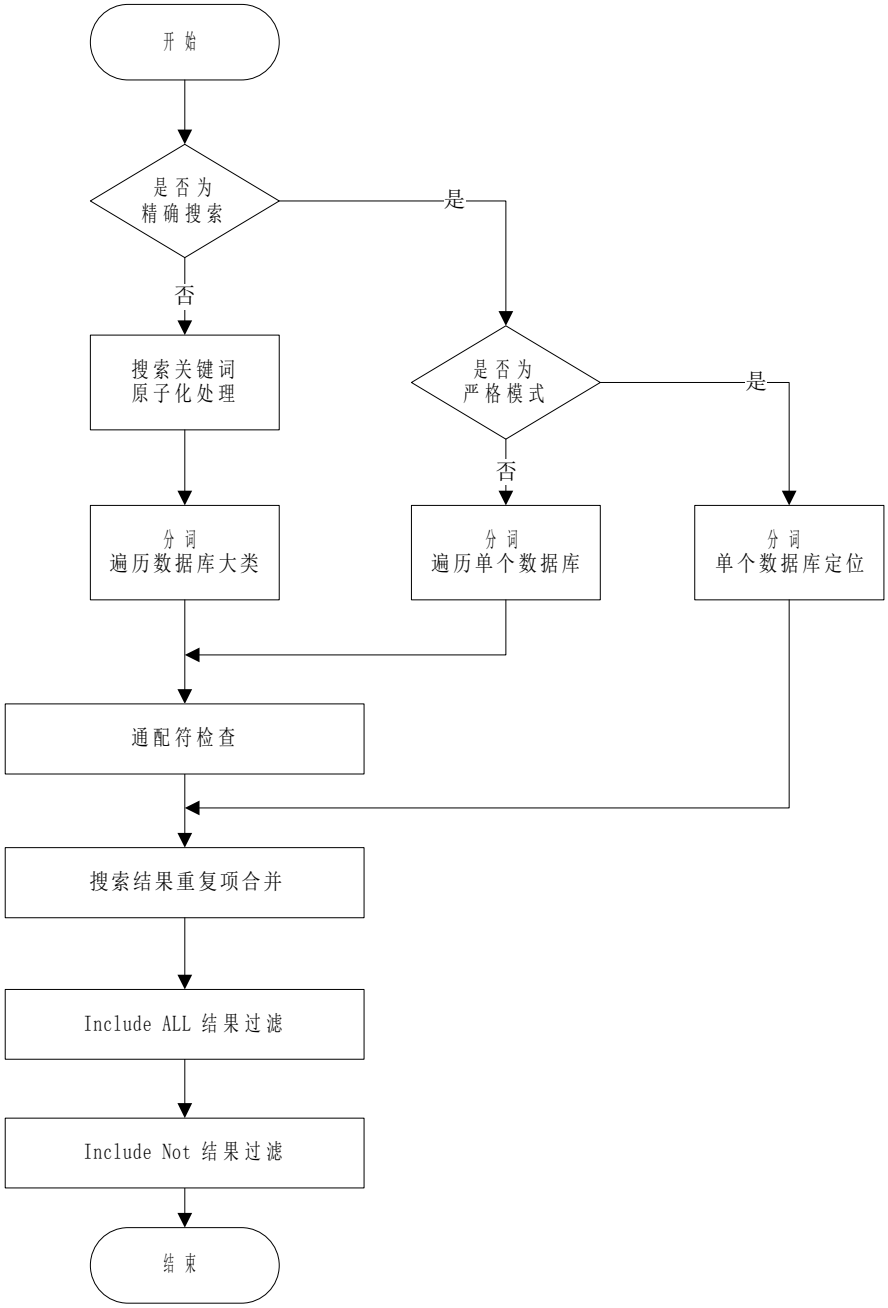


图28 搜索模式流程图

2） 搜索输入分级解析与结果合并

针对用户输入，软件采用了二层解析的方式，将原始输入字串Input逐层解析为可用于倒排索引查找的关键词Name。第一层解析为天然分词，根据空格和逗号将Input分解为多个Word，每一个Word会进行Glob通配处理，生成的Glob_Word，可以直接作为正则表达式的匹配项，在最后通配检查结果时使用。之后利用Split模块的分词接口，将Word转化为Name，此步骤的处理与Index建

立索引时使用的分词处理相同。每个Name在对应数据库中查找到的地址均是多个（倒排索引一个关键词对应多个地址），此时先对同Word下的Name对应的结果地址进行合并操作，Merge时注意关联度分数的叠加。之后根据之前得到的Glob_Word对每个Word下对应的搜索结果地址列表进行通配符检查操作。最后将通过检查的结果地址合并为一个总哈希表，每个地址（Key）对应一个关联度分数（Value）。之后进行重复项过滤、匹配父地址关联度叠加，Include All/Not搜索选项过滤等操作，在此不一一详细说明，请参见Search.pm模块。

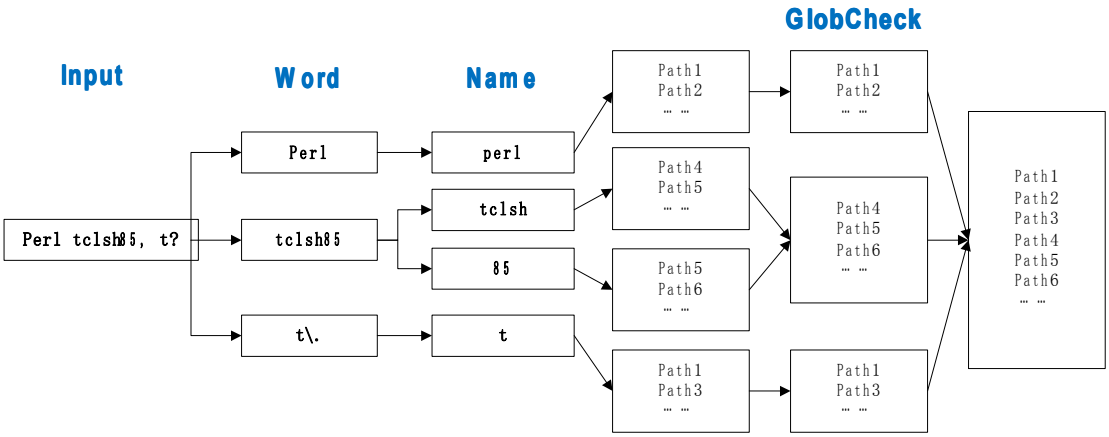


图29 分级解析与结果合并示意图

3） 关联度计算

搜索结果关联度的评估是搜索过程中的重要过程，也是体现搜索软件质量优劣的重要指标。关联度体现的是搜索结果与用户输入之间的匹配度，其计算素材的来源主要是用户输入的字串以及数据库中的索引。比较显而易见的关联因素，如词频等；另外，索引地址的总数等隐含在索引数据中的变量也会间接影响关联度。若某个地址自身匹配搜索，同时其父地址也匹配搜索，那么父地址的关联度会以一定比例加乘到子地址中。

在参考了相关文献，并经过大量实验之后，总结出如下关联度计算公式：

其中，

为整个地址的综合关联度

为单个关键词给地址贡献的关联度

为该地址包含的且匹配搜索关键词的父地址的关联度

为父地址关联度对地址总关联度的贡献比例

为整个索引数据中关键词的总个数

□为包含有该关键词的地址数

为该索引关键词在地址中出现的次数，即词频

为搜索串与索引关键词的匹配度（）

其直观表达为：如果某个关键词在该地址中出现的次数越多，关联度越大；如果该关键词包含在过多地址中，则说明比较普通常见，关联度要降低；在搜

索过程中，该关键词与倒排索引的Key值相差的字符数越少，关联度越高；一个地址的综合关联度为其包含的各个关键词关联度之和。

在实现中， 即为数据库基本信息的 TOTALNUM 对应值，即为倒排索引数据库中内层哈希的Value值，即为内层哈希的Key的个数，可由Key列表放置于标量上下文直接转换得到。三个关键影响因素都直接给出（时间花在索引建立过程中），以提高搜索时的速度。

4） HTML文件内容索引

HTML是一类特殊的文本文件，其通过比较松散的规则组织各式静态元素，可以被浏览器解析，从而呈现出丰富美观的信息。然而，HTML文件中并非所有数据都是有效且应该被索引的，例如，HTML文件中的各种标签，如<p>、<tr>、<li>、<href>、等，仅仅为了组织排列数据或强调某个字串，并没有明显意义，而且HTML头部的信息很多也没有特征，例如编码格式是UTF-8或gb2312。

对于HTML类型文件索引，需要提取出其直观的信息，即解析后呈现给用户的信息。这一原则也适用于其他类似文件类型，如XML。

在实现中，我们调用了外部模块的接口HTML::TreeBuilder::Obj->guts()。当以标量上下文调用此接口时，会返回HTML中可视文字段落。这正是我们希望索引的内容。原本打算自己实现，不过实在太好用了，就一直用下去了。

5） 网络信息获取

网络信息获取是十分有用的辅助功能，用户在看到搜索结果后，如果希望能参考网络上的相关信息，可以直接通过一键点击Baidu或Yahoo按钮完成。

第一步是拼装url并发送。

雅虎和百度均有固定url格式用于查询

http://www.yahoo.cn/s?q=$word"
http://www.baidu.com/s?wd=$word"

其中的$word替换为希望查询的字符串，发送url到对端服务器便可获取包含搜索结果HTML文件。

第二步是解析HTML结果。

获取到的原始HTML十分杂乱，标签元素众多。通过阅读分析HTML格式，找出包含目标结果的顶层标签。把原始HTML解析为树形结构，查找该标签并过滤出目标字符串及对应网站链接。

百度将结果盛放在<table>标签中，雅虎和谷歌都将其盛放在<li>标签中，实际解析过程中，由于雅虎的HTML很干净，直接提取出全部anchor（标签<a>），

就可以找到目标字串，再提取出该anchor下<href>部分的链接即为目标网址链接（而且还是原始链接没重定向）。而百度的HTML解析过程曲折一些，需要逐个分析table，并谨慎过滤掉广告。

第三步是过滤广告。

主要是百度，广告比较多，而且数量不固定且都放在前面。用了比较笨的方法，忽略第一个table（永远是e.baidu.com推广链接），之后根据背景色判断是否为广告，广告table的背景色为灰色（#f5f5f5），正常搜索结果的背景色为白色。

最后呈现给用户的是字串内容，及网络信息搜索结果的文本大标题，用户点击相应标题可以通过浏览器进入对应网址。最下方提供More选项，点击则通过浏览器打开网络搜索引擎结果显示页面。

6） 关键词分词

分词原则详见需求文档：

4.1.13 SRS.SPLIT.KEYWORD.001 针对数字分词

4.1.14 SRS.SPLIT.KEYWORD.002 针对英文分词

4.1.15 SRS.SPLIT.KEYWORD.003 针对中文分词

在实现中，除了完成需求中的分词处理外，还针对多个连续的英文单词进行了分词处理。根据观察，很多文件和文件夹的名称通过大小写交替表示间隔，例如TixBook.pdf，MeanShift.cpp。通过正则匹配进行分词。另外，可以根据'分词并且扔掉剩下一个长度的字头，例如he's的s，Rock'n'Roll的n。另外，对于模糊搜索的原子数据，如果某个搜索的关键词，例如ir这种，去掉r就只剩下一个字母的，不会去掉r，类似的ing，ed同样处理。

7） 文本索引数据恢复

本地数据库根据数据库文件形式分为两种，分别为：二进制文件数据库和文本文件数据库。其对比与优劣可参见下表。

表14 二进制与文本数据库对比表

数据库类型	稳定性	体积	读取速度	存储速度	跨平台性
二进制数据库	差	大	快	快	一般，根据类型有特定平台支持
文本数据库	好	小	慢	慢	好，但需要注意换行符等细节

两类数据库互有优劣，只有结合两者的优点才能提供最稳定、快速的数据服务。

索引文件需同时保存至二进制和文本数据库两种形式，并均可进行读取。从二进制库读取索引，速度快，优先选用。但二进制文件容易损坏，所以还需通过文本库进行备份，并创建一份MD5校验文件，保存上次退出时索引文件的校验和，当校验不通过时，清空二进制库的内容，从文本库进行索引恢复。

在实际测试过程中，发现加入文本库后，建立索引数据的时间增加了约20%，文本库的体积约为二进制库体积的1/3。在索引根目录不是特别庞大的前提下，这些代价是可以接受的。而本软件的定位即使中小型文件目录搜索，故采用此种二进制库备份与恢复方案。

参考资料清单：

[1] Tck/Tk入门经典（第2版）, J.K.Ousterhout, K.Jones著，张元章译，清华大学出版社

[2] Lucene in Action, E.Hatcher, O.Gospodnetic, M.McCandless著，人民邮电出版社

[3] Advanced Perl Programming, Simon Cozens, O'Relly Publisher

[4] Tk Manual Page http://www.tcl.tk/man/tcl8.4/TkCmd/

[5] Perl CPAN Website http://search.cpan.org/

[6] Perl Programming Documentation http://perldoc.perl.org/

[7] Stack Overflow Community http://stackoverflow.com/

[8] Apache Lucene Home Page http://lucene.apache.org/