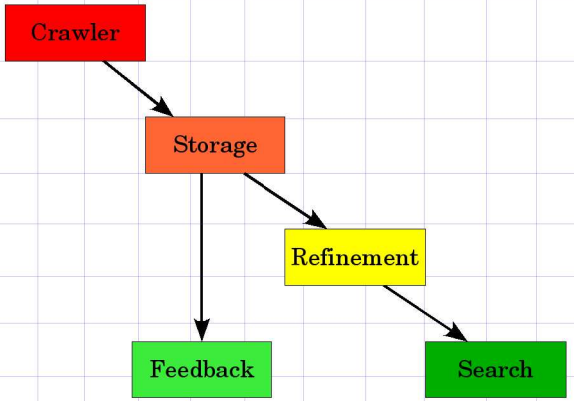
Implementierung einer Suchmaschine am Beispiel von ht://Dig

Georg Wittenburg

22.1.2003 – Netzbasierte Informationssysteme – FU Berlin

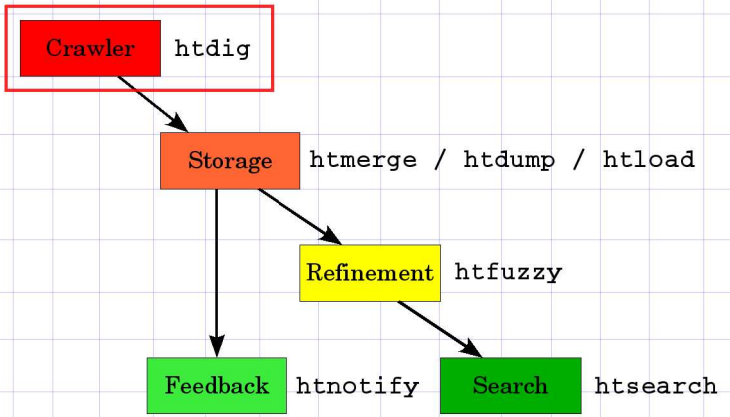
Konzeptioneller Aufbau



ht://Dig

- komplettes System zur Indizierung und Suche in einer Domain oder Intranet
- entwickelt von der San Diego State University, lizensiert unter GPL
- geschrieben in C++, modular aufgebaut
- spricht `file://` und `http://`
- über 500 “high-profile” Installationen
- mehr Information unter `www.htdig.org`

Interner Aufbau von ht://Dig



Crawler - htdig

- sammelt Dokumente per `file://` oder `http://`
- unterstützt mehrere Formate: Plain Text, HTML, PDF
- externe Filter lassen sich einbinden
- Crawler kann sich mit Usernamen und Passwort am Server anmelden

htdig - Retriever::Start()

```
while (more && noSignal)
{
    more = 0;
    //
    // Go through all the current servers in sequence.  We take only one
    // URL from each server during this loop.  This ensures that the load
    // on the servers is distributed evenly.
    //
    servers.Start_Get();
    while ( (server = (Server *)servers.Get_NextElement()) && noSignal)
    {
        ref = server->pop();
        if (!ref)
            continue;               // Nothing on this server
        more = 1;
        //
        // Deal with the actual URL.
        // We'll check with the server to see if we need to sleep()
        // before parsing it.
        //
        server->delay();   // This will pause if needed and reset the time
        parse_url(*ref);
        delete ref;
    }
}
```

htdig - Retriever::parse_url()

```
url.parse(urlRef.URL());
doc->Reset();
doc->Url(url.get());
doc->Referer(urlRef.Referer());
base = doc->Url();
// Retrive document, first trying local file access if possible.
Document::DocStatus status;
server = (Server *) servers[url.signature()];
StringList *local_filenames = GetLocal(url.get());
if (local_filenames) {
    status = doc->RetrieveLocal(date, local_filenames);
    if (status == Document::Document_not_local) {
        if (local_urls_only)
            status = Document::Document_not_found;
        else if (server && !server->IsDead())
            status = doc->RetrieveHTTP(date);
        else
            status = Document::Document_no_server;
    }
    delete local_filenames;
} else if (server && !server->IsDead())
    status = doc->RetrieveHTTP(date);
else
    status = Document::Document_no_server;
```

htdig - Retriever::parse_url() (cont.)

```
switch (status) {
    case Document::Document_ok:
    if (old_document)
    {
        if (doc->ModTime() == ref->DocTime()) {
            words.MarkScanned();
            break;
        }
        words.MarkModified();
        int backlinks = ref->DocBackLinks();
        delete ref;
        current_id = docs.NextDocID();
        words.DocumentID(current_id);
        ref = new DocumentRef;
        ref->DocID(current_id);
        ref->DocURL(url.get());
        ref->DocState(Reference_normal);
        ref->DocAccessed(time(0));
        ref->DocHopCount(currenthopcount);
        ref->DocBackLinks(backlinks);
    }
    RetrievedDocument(*doc, url.get(), ref);
    break;
[...]
```

htdig - Retriever::RetrievedDocument()

```
n_links = 0;
current_ref = ref;    current_anchor_number = 0;
current_title = 0;    current_time = 0;
current_head = 0;     current_meta_dsc = 0;
// Create a parser object and let it have a go at the document.
// We will pass ourselves as a callback object for all the got_*() routines.
// This will generate the Parsable object as a specific parser
Parsable *parsable = doc.getParsable();
if (parsable)
    parsable->parse(*this, *base);
else { // If we didn't get a parser, then we should get rid of this!
    ref->DocState(Reference_noindex);
    return;
}
ref->DocHead(current_head);
ref->DocMetaDsc(current_meta_dsc);
if (current_time == 0)
    ref->DocTime(doc.ModTime());
else
    ref->DocTime(current_time);
ref->DocTitle(current_title);
ref->DocSize(doc.Length());
ref->DocAccessed(time(0));
ref->DocLinks(n_links);
ref->DocImageSize(ref->DocImageSize() + doc.Length());
```

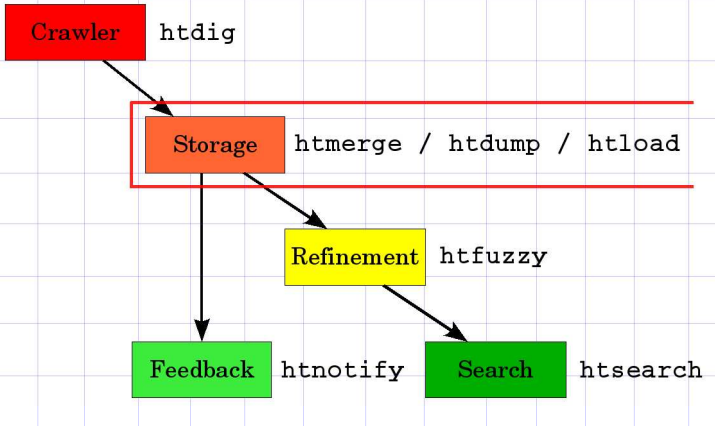
htdig - Retriever::got_word()

```
if (heading > 11 || heading < 0)  // Current limits for headings
    heading = 0;                  // Assume it's just normal text
if (trackWords)
{
    String w = word;
    HtStripPunctuation(w);
    if (w.length() >= minimumWordLength)
        words.Word(w, location, current_anchor_number, factor[heading]);
    if (strcmp(word, w.get()) != 0)    // have punctuation that was stripped
    {
        [...]    // Check for compound words...
    }
}
```

htdig - Retriever::got_href()

```
ref = docs[url.get()];
if (!ref) [...]
ref->DocBackLinks(ref->DocBackLinks() + 1);
ref->DocURL(url.get());
ref->AddDescription(description);
if (currenthopcount + 1 > max_hop_count) {
    delete ref;
    return;
}
docs.Add(*ref);
if (Need2Get(url.get())) {
    server = (Server *) servers[url.signature()];
    if (!server)    {
        String robotsURL = "http://";
        robotsURL << url.host() << "/robots.txt";
        StringList *localRobotsFile = GetLocal(robotsURL.get());
        server = new Server(url.host(), url.port(), localRobotsFile);
        servers.Add(url.signature(), server);
        delete localRobotsFile;
    }
    server->push(url.get(), ref->DocHopCount(), base->get(),
            IsLocalURL(url.get()));
    String   temp = url.get();
    visited.Add(temp, 0);
}
delete ref;
```

Interner Aufbau von ht://Dig



Storage – **htmerge**

- erzeugt Index und Wortliste zur späteren Suche
- Möglichkeit, Daten aus unabhängigen Crawling-Läufen zu verschmelzen
- Backend ist Sleepycat Software's Berkeley DB (www.sleepycat.com)

htmerge – mergeDB()

```
while ((url = (String *) urls->Get_Next())) {
    DocumentRef *ref = merge_db[url->get()];
    DocumentRef *old_ref = db[url->get()];
    if (!ref) continue;
    if (old_ref) {
        if ( old_ref->DocTime() > ref->DocTime() ) {
            // Cool, the ref we're merging is too old, just ignore it
            char str[20];
            sprintf(str, "%d", ref->DocID());
            merge_dup_ids.Add(str, 0);
        } else {
            // The ref we're merging is newer, delete the old one and add
            char str[20];
            sprintf(str, "%d", old_ref->DocID());
            db_dup_ids.Add(str, 0);
            db.Delete(url->get());
            ref->DocID(ref->DocID() + docIDOffset);
            db.Add(*ref);
        }
    } else {
        ref->DocID(ref->DocID() + docIDOffset);
        db.Add(*ref);
    }
    delete ref;
    delete old_ref;
}
```

htmerge – mergeDB() (cont.)

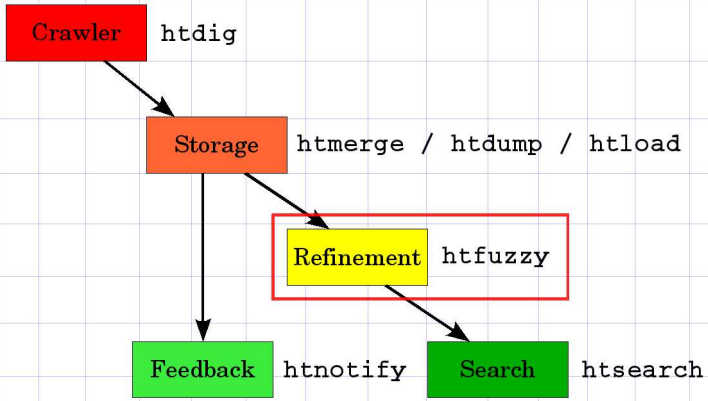
```
while (fgets(buffer, sizeof(buffer), dbwords)) {
    word = good_strtok(buffer, '\t');
    pair = good_strtok(NULL, '\t');
    wr.Clear();  // Reset count to 1, anchor to 0, and all that
    sid = "-";
    while (pair && *pair) {
        name = strtok(pair, ":");
        value = strtok(0, "\n");
        if (name && *name && value && *value) {
            switch (*name) {
                case 'c': wr.count = atoi(value); break;
                case 'l': wr.location = atoi(value); break;
                case 'i': sid = value; wr.id = atoi(value); break;
                case 'w': wr.weight = atoi(value); break;
                case 'a': wr.anchor = atoi(value); break;
            }
        }
        pair = good_strtok(NULL, '\t');
    }
    if (db_dup_ids.Exists(sid)) continue;
    fprintf(wordlist, "%s", word.get());
    fprintf(wordlist, "\ti:%d\tl:%d\tw:%d", wr.id, wr.location, wr.weight);
    if (wr.count != 1) fprintf(wordlist, "\tc:%d", wr.count);
    if (wr.anchor != 0) fprintf(wordlist, "\ta:%d", wr.anchor);
    putc('\n', wordlist);
}
```

htmerge – db.wordlist

```
linux   i:103   l:782   w:399    c:2
linux   i:104   l:561   w:1093   c:4
linux   i:11    l:60    w:9346   c:20
linux   i:25    l:734   w:487    c:2
linux   i:26    l:969   w:31
linux   i:29    l:664   w:336    a:2
linux   i:3     l:435   w:565
linux   i:4     l:567   w:828    c:2
linux   i:82    l:648   w:352
linux   i:84    l:949   w:51     a:6
linux   i:86    l:0     w:152282 c:7
linux   i:88    l:59    w:9368   c:20
linux   i:9     l:0     w:304094 c:12
```

- i: Wort – ID (db.words.db – Primary Key)
- l: Location – ID (db.docdb – Primary Key)
- w:Weight – Relevanz des Eintrags
- c: Count – Anzahl an der Location

Interner Aufbau von ht://Dig



Refinement – `htfuzzy`

- bearbeitet den Datenbestand, sodass erweiterte Suchenanfragen möglich werden
- verbessert Toleranz bei falschen Anfragen
- verschiede Algorithmen stehen zur Auswahl

`htfuzzy` - Algorithmen

- **soundex:** matcht ähnlichen Klang
- **metaphone:** soundex optimiert auf Englisch
- **accents:** entfernt Akzente
- **endings:** matcht häufige Wortendungen (Regel / Wörterbuch von ispell)
- **synonyms:** matcht Synonyme (externe Synonymdatei)

`htfuzzy` - `main()`

```
//
// Go through all the words in the database
//
char *key;
Fuzzy    *fuzzy = 0;
String   word, fuzzyKey;

worddb->Start_Get();
while ((key = worddb->Get_Next())) {
    word = key;
    wordAlgorithms.Start_Get();
    while ((fuzzy = (Fuzzy *) wordAlgorithms.Get_Next()))
        fuzzy->addWord(word);
}

//
// All the information is now in memory.
// Write all of it out to the individual databases
//
wordAlgorithms.Start_Get();
while ((fuzzy = (Fuzzy *) wordAlgorithms.Get_Next()))
    fuzzy->writeDB(config);
worddb->Close();
delete worddb;
if (fuzzy)
    delete fuzzy;
```

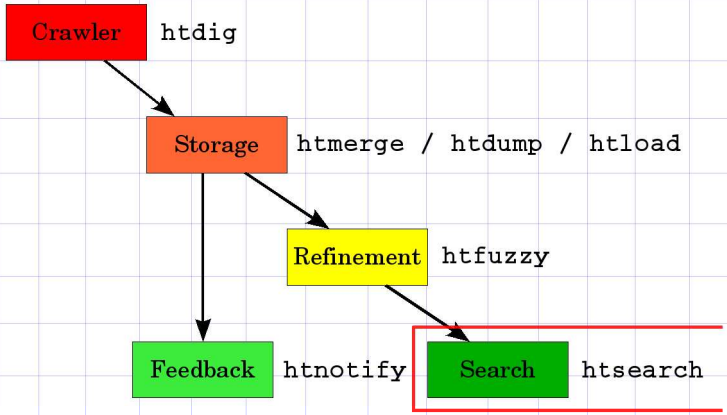
htfuzzy – Soundex::addWord()

```
void Soundex::addWord(char *word)
{
    if (!dict)
        dict = new Dictionary;

    String  key;
    generateKey(word, key);

    String  *s = (String *) dict->Find(key);
    if (s)
        (*s) << ' ' << word;
    else
        dict->Add(key, new String(word));
}
```

htfuzzy – Soundex::generateKey()

```
void Soundex::generateKey(char *word, String &key)
{
    int code = 0;
    int lastcode = 0;
    key = 0;
    if (!word) {key = '0'; return;}
    while (!isalpha(*word)) word++;
    if (word) key << *word++;
    else {key = '0'; return;}
    while (key.length() < 6) {
        switch (*word) {
        case 'b': case 'p': case 'f': case 'v':    code = 1; break;
        case 'c': case 's': case 'k': case 'g': case 'j': case 'q': case 'x': case 'z':
                                                   code = 2; break;
        case 'd': case 't':                        code = 3; break;
        case 'l':                                  code = 4; break;
        case 'm': case 'n':                        code = 5; break;
        case 'r':                                  code = 6; break;
        case 'a': case 'e': case 'i': case 'o': case 'u': case 'y': case 'w': case 'h':
                                                   code = 0; break;
        default:                                   break;
        }
        if (code && code != lastcode) {key << code; lastcode = code;}
        if (*word) word++;
        else break;
    }
}
```

Interner Aufbau von ht://Dig



Search – htsearch

- führt Suche durch und erstellt Resultatliste
- wird von einem HTML Formular aus aufgerufen
- CGI-Programm
- verarbeitet POST und GET

htsearch - main()

```
// [...]

ResultList *results = htsearch(word_db, searchWords, parser);

// [...]

Display display(index, doc_db);
display.setOriginalWords(originalWords);
display.setResults(results);
display.setSearchWords(&searchWords);
display.setLimit(&limit_to);
display.setExclude(&exclude_these);
display.setAllWordsPattern(&searchWordsPattern);
display.setCGI(&input);
display.setLogicalWords(logicalWords);
if (parser->hadError())
    display.displaySyntaxError(parser->getErrorMessage());
else
    display.display(pageNumber);
```

htsearch - htsearch()

```
ResultList *
htsearch(char *wordfile, List &searchWords, Parser *parser)
{
    //
    // Pick the database type we are going to use
    //
    ResultList *matches = new ResultList;
    if (searchWords.Count() > 0)
    {
        Database *dbf = Database::getDatabaseInstance();

        dbf->OpenRead(wordfile);

        parser->setDatabase(dbf);
        parser->parse(&searchWords, *matches);
        dbf->Close();
        delete dbf;
    }

    return matches;
}
```

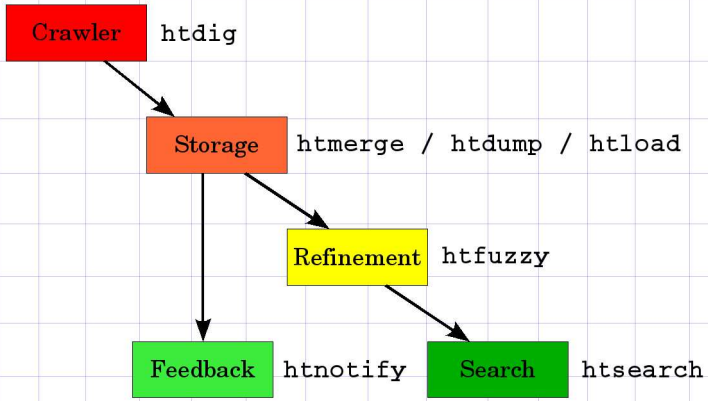
htsearch - parse()

```
void
Parser::parse(List *tokenList, ResultList &resultMatches)
{
    tokens = tokenList;
    fullexpr(1);

    ResultList *result = (ResultList *) stack.pop();
    if (!result) {    // Ouch!
        if (!valid)
            return;
        valid = 0;
        error = 0;
        error << boolean_syntax_errors[0] << ' ' << boolean_syntax_errors[1];
        return;
    }
    List *elements = result->elements();
    DocMatch *dm;
    for (int i = 0; i < elements->Count(); i++) {
        dm = (DocMatch *) (*elements)[i];
        resultMatches.add(dm);
    }
    elements->Release();
    result->Release();
    delete elements;
    delete result;
}
```

htsearch - perform_and()

```
stack.push(result);
elements = l2->elements();
for (i = 0; i < elements->Count(); i++)
{
    dm = (DocMatch *) (*elements)[i];
    dm2 = l1->find(dm->id);
    if (dm2 ? isand : (isand == 0))
    {
        //
        // Duplicate document.  We just need to add the scored together.
        //
        dm3 = new DocMatch;
        dm3->score = dm->score + (dm2 ? dm2->score : 0);
        dm3->id = dm->id;
        dm3->anchor = dm->anchor;
        if (dm2 && dm2->anchor < dm3->anchor)
            dm3->anchor = dm2->anchor;
        result->add(dm3);
    }
}
elements->Release();
delete elements;
delete l1;
delete l2;
```

Interner Aufbau von ht://Dig



Konfiguration - `htdig.conf`

- **start_url:** Startseite des Crawlers
- **limit_urls_to:** Welche Domains soll der Crawler besuchen dürfen?
- **exclude_urls:** Welche auf keinen Fall?
- **bad_extensions:** zu ignorierende Dateien
- **maintainer:** Admin dieser Installation
- **search_algorithms:** Liste gewichteter Algorithmen

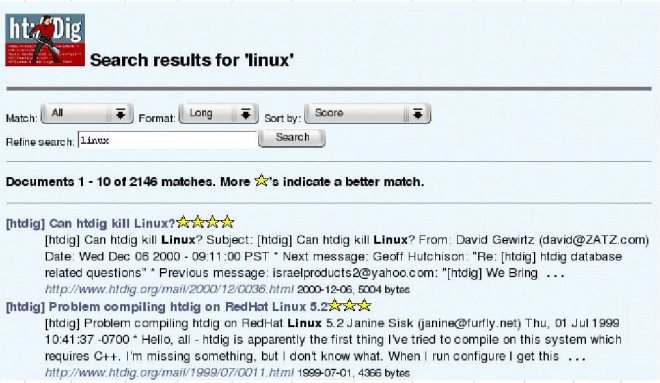
Admin Interface – `rundig`

1. **htdig:** Neue Daten sammeln
2. **htmerge:** Daten in (meist vorhandene Datenbank aufnehmen)
3. **htnotify:** ggf. Admin fehlerhafter Seiten verständigen (z.B. bei broken links)
4. **htfuzzy:** Datenbank für Anfragen optimieren

User Interface

- **search.html:** Suchformular
- **header.html:** Anfang der Resultatseite
- **footer.html:** Ende der Resultatseite
- **wrapper.html:** Resultatseite, mit Platzhalter `$(HTSEARCH_RESULT)`
- **nomatch.html:** keine Search-Hits
- **syntax.html:** Syntaxbeschreibung der boolschen Ausdrücke

Summa Summarum



Ende

Vielen Dank!