

本日のお品書き

- hack #1 - MyBlogSearch
 - 前回からの進歩

- MyBlogSearch Scheduler
 - 設計方針
 - データベースのスキーマ
 - Tasking & Threading

- JSpider hacks
 - JSpider に施した改造について
 - blog 記事だけをクローリングする - 設計 -
 - blog 記事だけをクローリングする - 設定 -
 - blog 記事だけをクローリングする - 実装 -
 - RSS に対応する - 前置き -
 - RSS ってななに
 - Informa
 - RSS に対応する - 設計 -
 - RSS に対応する - 設定 -
 - RSS に対応する - 実装 -

- これからの予定

- 付録
 - 尾内 blog の RSS

- ・ hack #1 - MyBlogSearch

- 前回からの進歩

- 前回の勉強会で見たものから、次のような進歩がありました。

- ・ スケジューラを作成中

- MyBlogSearch で使っているクローラ(MyBlogSearch crawler)は JSpider に手を加えたものです。実行の仕組みとしては JSpider そのままで、バッチファイル(or シェルスクリプト) から、ダウンロードしたい blog の URL を指定して起動し、その blog の記事データをクローリングしてくる、というものになっています。翻って blog とは日々新しい記事が追加されていくものであり、クローリングしてきたデータを新鮮な状態に保つには、定期的なクローリングをして、新しく追加された記事を集めてきてやる必要があります。J Spider にはこのような機能がないため、これを実現するためには、手ずからスケジューラを実装してやらなくてはなりません。

- そこで、C#を使ってスケジューラを書いてみました。MyBlogSearch として、新鮮なデータを提供するためには、blog の記事データをクローリングするだけでなく、その後のインデクシング等の処理までを行わなくてはなりません。今回作成したスケジューラは、クローリングからその後の処理までを一貫して管理し、またそれらをなるべく能率的に実行できるようなものになっていますというかそうなる予定ですたぶん。スケジューラ的设计と実装については、これから簡単に説明します。

- ・ データベース(SQL Server)を導入

- blog の URL やユーザの情報を一括して管理するのにデータベースが欲しくなったので、『SQL Server』という Microsoft 製のデータベースを買ってきて、適当にスキーマを作成し、スケジューラに組み込みました。

- ・ blog 記事の HTML から本文を抽出するコードを実装

- blog の記事をクラスタリング(後述)しようとするときに、タイトルやサイドバーなどの定型的部分がとてつもなく邪魔になるので、これをなるべく除外すべく、blog 記事の HTML から本文を抽出するコードを書きました。

- ・ JSpider の RSS 対応

- データを新鮮に保つためには継続的かつ定期的なクローリングが必要になりますが、毎回毎回 blog 全体をダウンロードしたのでは、実装としてあまりにもださいですし、クローリング先のサーバと回線に負担になります。よって、RSS から更新をチェックし、新着記

事だけをダウンロードしに行くようにしました。

・ MyBlogSearch Scheduler

設計方針

スケジューラを作る動機は、おおむね次のようなものです。

- ・ プログラムの実行を自動化したい
- ・ プログラムを能率的に実行したい

で、現状、MyBlogSearch は以下のような各個に独立したプログラムで構成されています。

- ・ MyBlogSearch crawler (もとい JSpider。blog 記事のダウンロード)
- ・ MyBlogSearch extractor(blog 記事の HTML の本文抽出)
- ・ namazu(インデクシング)
- ・ gnmz(namazu のインデックスを使って文書をクラスタリングするプログラム)

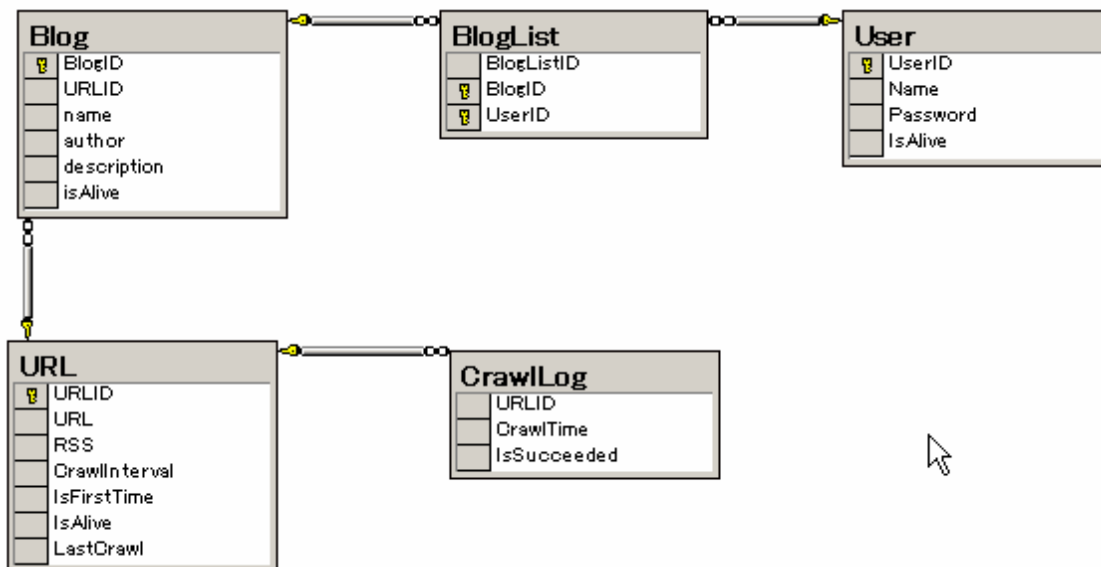
これらは以下のような流れで実行してやる必要があります。

ダウンロード 本文抽出 インデクシング クラスタリング

ここで、お互いの処理を同期させたいという考えが出てきます。つまり、ある blog のダウンロードが終わったら、その blog の記事の本文抽出を行えっていう指示を出し、それが終わったらその blog を登録しているユーザのインデックスを再構成しろっていう指示を出し……と、あるプログラムの実行が次のプログラムの実行のトリガーになるようにしたいってことです。慣れ親しんだ言葉だとイベントドリブンと言っても良いです。で、実際に MyBlogSearch のスケジューラは、そのあたりを意識した設計になっています。

データベースのスキーマ

現状、このようなものになっています。現時点で最低限必要なものだけを作成しているので、これから作り込んでいくようであれば、テーブルやフィールドが追加されていくとおもいます。



- User

ユーザを管理します。

いまのところ、フィールドはユーザ名とパスワードだけです。パスワードは暗号化やハッシングはされておらず、平文で入力されてます。

- BlogList

ユーザの登録している blog のリスト。User と Blog が多対多の関係になるので、間にこれを挟んでいます。

- Blog

blog を管理します。

URLID として URL への参照を持ちます。他に blog の名称、管理者、説明のフィールドがあります。

- URL

blog のトップページと RSS の URL を持ちます。

CrawlInterval として、その URL をクローリングする間隔を指定します。将来的には更新頻度によってこれを上下させるようにしたいな—と思ってますがたぶんやらないでしょう。

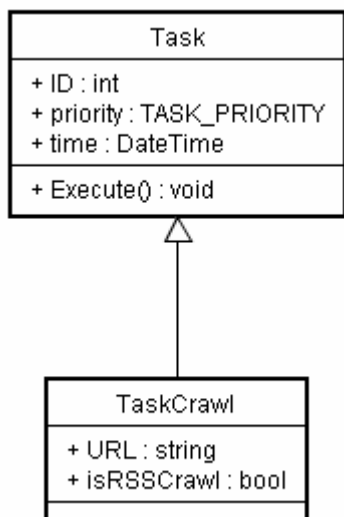
- CrawlLog

クローリングのログをここに書き込みます。

Tasking & Threading

個々のプログラムの実行を、タスクという単位で扱います。クローラなら、ある blog のデータをダウンロードする処理が、インデクシングなら、あるユーザ用のインデックスを作成する処理が1つのタスクに相当します。

タスクは、個々のプログラムに対してクラスとして実装されます。クローラならクローラのタスク、インデクサならインデクサのタスク、という具合で。UML で表現するとこのようになります。Task インターフェースを継承し、TaskCrawl を作成します。TaskCrawl の属性として、クローリングに必要な、URL などの情報が宣言されています。UML では継承先のクラスとして TaskCrawl のみを示していますが、実際には他のプログラムに対応したクラスも作成されます。



このタスクを、リスト（待ち行列）に積んでやります。また、その際、Task ごとに指定された優先度によって、実行する順番をソートします。優先度を表現する列挙型として、TASK_PRIORITY が宣言されており、以下のようになっています。

```
public enum TASK_PRIORITY
{
    UNKNOWN = 0,
    EMERGENCY,
    IMMEDIATE,
    TIME,
    BACKGROUND
};
```

EMERGENCY > IMMEDIATE > BACKGROUND の順番で優先度が低くなります。また、TIME では、指定された時刻にそのタスクが実行されます。

で、タスクを積むリストはスレッドに関連づけられます。個々のスレッドにチェックすべきリストが与えられており、リストを定期的にチェックし、処理すべきタスクがあればそれを取り出して実行します。スレッドは数を指定することができます。

contents

crawl

TaskName MyBlogSearch_Scheduler.TaskCrawl
TaskList MyBlogSearch_Scheduler.TaskList
Task Thread MyBlogSearch_Scheduler.Task Thread
ThreadCount 3

tasks

ID	Type	Priority	Time	Cra...	URL
40	MyBlogS...	TIME	2004/09/03 12:04:29	RSS	http://watcher.moe-ni...
41	MyBlogS...	TIME	2004/09/03 9:07:29	RSS	http://kokogiko.net/ ...
42	MyBlogS...	TIME	2004/09/03 10:34:29	RSS	http://blog.bulknews.n...
43	MyBlogS...	TIME	2004/09/03 8:57:29	RSS	http://12345.seesaa.n...
44	MyBlogS...	TIME	2004/09/03 8:52:29	RSS	http://www.wiredope...
45	MyBlogS...	TIME	2004/09/03 11:51:29	RSS	http://www.caeylogic...
46	MyBlogS...	TIME	2004/09/03 14:19:29	RSS	http://www.chikawata...
47	MyBlogS...	TIME	2004/09/03 10:28:29	RSS	http://www.dashiblog...
48	MyBlogS...	TIME	2004/09/03 9:26:29	RSS	http://uva.jp/dh/mt/ ...
49	MyBlogS...	TIME	2004/09/03 14:45:29	RSS	http://www.enatural.o...
5	MyBlogS...	TIME	2004/09/03 13:22:29	RSS	http://www.seman.cs...
50	MyBlogS...	TIME	2004/09/03 9:18:29	RSS	http://www.flowerloun...
51	MyBlogS...	TIME	2004/09/03 11:27:29	RSS	http://www.milkstand...
52	MyBlogS...	TIME	2004/09/03 12:38:29	RSS	http://www.goodpic.c...
53	MyBlogS...	TIME	2004/09/03 11:35:29	RSS	http://netafull.net/go...
54	MyBlogS...	TIME	2004/09/03 12:27:29	RSS	http://www.hirokiazu...
55	MyBlogS...	TIME	2004/09/03 12:05:29	RSS	http://hoya.jugem.cc/...
56	MyBlogS...	TIME	2004/09/03 11:52:29	RSS	http://d.hatena.ne.jp/...
57	MyBlogS...	TIME	2004/09/03 12:15:29	RSS	http://www.sfc.wide.a...
58	MyBlogS...	TIME	2004/09/03 10:49:29	RSS	http://shimizu.typepa...
59	MyBlogS...	TIME	2004/09/03 10:15:29	RSS	http://www.ntlabs.gr.j...
6	MyBlogS...	TIME	2004/09/03 12:10:29	RSS	http://www.seman.cs...
60	MyBlogS...	TIME	2004/09/03 10:50:29	RSS	http://www.tez.com/b...
61	MyBlogS...	TIME	2004/09/03 11:32:29	RSS	http://d.hatena.ne.jp/...
62	MyBlogS...	TIME	2004/09/03 12:01:29	RSS	http://livre.vis.ne.jp/w...
63	MyBlogS...	TIME	2004/09/03 13:10:29	RSS	http://knn.typepad.co...
64	MyBlogS...	TIME	2004/09/03 11:23:29	RSS	http://5net.com/blog/...
65	MyBlogS...	TIME	2004/09/03 13:40:29	RSS	http://blog.neoteny.co...
66	MyBlogS...	TIME	2004/09/03 8:58:29	RSS	http://kinshachi.ddo.j...
67	MyBlogS...	TIME	2004/09/03 14:00:29	RSS	http://www.tngar.com...
68	MyBlogS...	TIME	2004/09/03 9:33:29	RSS	http://metamemos.ty...
69	MyBlogS...	TIME	2004/09/03 11:48:29	RSS	http://www.multitrea...

また、このあたりの設定はXMLをベースにした設定ファイルで行います。

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>

    <content name="crawl">
      <thread type="MyBlogSearch_Scheduler.TaskThread" count="3" />
      <list type="MyBlogSearch_Scheduler.TaskList" />
      <task type="MyBlogSearch_Scheduler.TaskCrawl" />
    </content>

    <!--

    <content name="extract">
      <thread type="TaskThread" count=1 />
      <list type="TaskList" />
      <task type="TaskExtract" />
    </content>

    <content name="indexing">
      <thread type="TaskThread" count=1 />
      <list type="TaskList" />
      <task type="TaskIndexing" />
    </content>

    <content name="clustering">
      <thread type="TaskThread" count=1 />
      <list type="TaskList" />
      <task type="TaskClustering" />
    </content>

    -->

  </appSettings>
</configuration>
```

- ・ JSpider hacks

JSpider に施した改造について

JSpider は起動時に設定フォルダの場所を指定し、そこに格納された設定によって、異なる動作をさせることができます。今回、"myblogsearch"、"myblogsearch_rss"という2つの新しい設定を作成しました。

- ・ myblogsearch

blog のトップページの URL を指定する。

指定された URL からリンクを辿っていき、その blog の個々の記事の HTML だけを (カテゴリや、日付別のもは除いて) ダウンロードする。

- ・ myblogsearch_rss

RSS の URL を指定する。

RSS をダウンロード パースし、まだダウンロードしていない記事があれば、それをダウンロードする。

以下、それぞれの実装と設定ファイルの内容について解説します。

また、現時点での JSpider のコードは にあります。Eclipse でコンパイルが通るようになってます (jar のパスを通す設定が必要かもしれません)。実際に御覧になりたい方はこちらをどうぞ。

blog 記事だけをクローリングする - 設計 -

加えた修正は簡単なものです。デフォルトで提供されている「特定のサイトだけをダウンロードする」という機能に、以下の2点を加えました。

- ・ 起点となったディレクトリ以下だけをダウンロードする
- ・ URL から判断し、blog 記事と思われるものだけをダウンロードする

2点目ですが、blog 記事の URL には、ツールごとに特徴があり、割と単純なルールを設定することでフィルタリングが可能です。たとえば MovableType では、blog 記事の URL

は以下ようになります。

<http://www.seman.cs.uec.ac.jp/~shin/blog/archives/000962.html>

"000962"の部分が個々の記事にユニークに割り振られる識別子で、記事を書くたびに、これがインクリメントされていきます。よって、これを正規表現なりなんなりで判断してやれば、その URL が blog 記事かどうかを判断することができます。他の blog ツールについても同じようなことが云えます。

ただ、blog ツールによってはこの URL を自分でカスタマイズすることが可能なものもあり、これだけの処理では、カスタマイズされたものを取り落とすこととなります。よって、より完全なクローリングを目指すなら、もっと別のやり方で判断をしなくちゃなりません。URL や RSS に依存しない blog 記事クローリングの実装としては、森本システムや東工大のなんたら研究室（名前わすれた）で作っている"blogWatcher"などがあります。blogWatcher のものについては、論文として発表されているので、どんな手法を使っているかある程度知ることが可能です。たしか森本さんが持ってたとおもうので読みたい人は森本さんからゲットしましょう。

blog 記事だけをクローリングする - 設定 -

設定ファイルは以下のようになっています。デフォルトで付属している"download"が修正元です。手を加えた場所のみを順番に抜き出していきます。

jspider.properties

```
# -----  
# User Agent Configuration  
# -----  
  
jspider.userAgent=MyBlogSearch crawler/0.2 (powered by JSpider) (+http://www.seman.cs.uec.ac.jp/~shin/blog/archives/cat_myblogsearch.html, shin@seman.cs.uec.ac.jp)
```

user-agent の設定です。参照 URL としては、暫定的にぼくの blog の関連記事のものを載せています。暇なときにちゃんとしたページをどこかに作ります。たぶん。

```
# -----  
# Rules Configuration  
# -----  
  
jspider.rules.spider.count=2  
jspider.rules.spider.1.class=net.javacoding.jspider.core.rule.impl.BaseDirectoryOnlyRule  
jspider.rules.spider.2.class=net.javacoding.jspider.mod.rule.OnlyHttpProtocolRule
```

ダウンロード時の rule として、"BaseDirectoryOnlyRule"を指定しています。自分で実装したクラスです。起点となった URL 以下のディレクトリにあるリソースのみをダウンロードします。たとえば、起点に"http://www.seman.cs.uec.ac.jp/~shin/blog/"を指定したとすると、それ以下に存在するぼくの blog のデータはダウンロードの対象になりますが、それより上に位置する尾内研のトップページ等は対象外になります。

Plugins/diskwriter.properties

```
plugin.filter.spider=net.javacoding.jspider.mod.eventfilter.BlogArchiveOnlyEventFilter
```

修正前のものでは、すべての Event を通過させる AllowAllEventFilter が指定されていましたが、それを変更し、Blog 記事とおぼしきもののみを通す BlogArchiveOnlyEventFilter を設定しています。BlogArchiveOnlyEventFilter は自分で実装したクラスです。

blog 記事だけをクローリングする - 実装 -

URL が blog 記事かどうかを判断する BlogArchiveOnlyEventFilter クラスです。

```
public class BlogArchiveOnlyEventFilter implements EventFilter {

    /* (非 Javadoc)
     * @see net.javacoding.jspider.mod.api.EventFilter#filterEvent(net.javacoding.jspider.api.event.JSpiderEvent)
     */

    private ArrayList validators;

    public BlogArchiveOnlyEventFilter()
    {
        //validator のインスタンスを作成して、
        //リストに容れておく

        validators = new ArrayList();

        validators.add(new MovableTypeDefaultSettingValidator());
        validators.add(new MovableTypeYearMonthSettingValidator());
        validators.add(new HatenaDiaryValidator());
        validators.add(new TDiaryValidator());
    }

    public boolean filterEvent(JSpiderEvent event) {
        // TODO 自動生成されたメソッド・スタブ

        if (event instanceof ResourceFetchedEvent)
        {
            ResourceFetchedEvent rfe = (ResourceFetchedEvent)event;
            URL url = rfe.getURL();
        }
    }
}
```

```
//各 validator に対して、処理中の URL を
//accept するかどうか問い合わせる
Iterator iterator=validators.iterator();
while(iterator.hasNext())
{
    BlogArchiveValidator validator = (BlogArchiveValidato
r)iterator.next();

    if(validator.validate(url))
        return true;
}
return false;
}
```

filterEvent 関数の中でその URL を accept するかどうかの判断をしています。blog の URL を判断するコードですが、各ツールごとに別々のコードを書いてやる必要があります、これをスマートに管理するため、blog 記事かどうかを判断する Interface を作成しました。

```
public interface BlogArchiveValidator {

    public boolean validate(URL url);

}
```

validate 関数は、引数として判断対象の URL を渡し、それが blog 記事かどうかを boolean で返します。これを実装したクラスとしては、現在、

- ・ MovableTypeDefaultSettingValidator (デフォルト設定の MovableType)
- ・ MovableTypeYearMonthSettingValidator (年月を URL に fix するように設定された MovableType。TypeKey もこれも)
- ・ HatenaDiaryValidator (はてなダイアリー)
- ・ TDiaryValidator (tDiary)

の4つが実装されています。これらの実装はいちいち取り上げませんので、見てみたいひとはソースコードに当たってください。

BlogArchiveOnlyEventFilter クラスのインストラクタで、これら4つのクラスをそれぞれインスタンス化し、ArrayList に突っ込んでいます。filterEvent 関数の中では、ArrayList から iterator を取得し、これを經由して各インスタンスにアクセスし、URL の判断を行っています。

こちらはわざわざお見せするようなものでもありませんが、BaseDirectoryOnlyRule クラスの実装は以下です。

```
public class BaseDirectoryOnlyRule extends BaseRuleImpl {
```

```
/* (非 Javadoc)
 * @see net.javacoding.jspider.spi.Rule#apply(net.javacoding.jspider.core.SpiderContext, net.javacoding.jspider.api.model.Site, java.net.URL)
 */
public Decision apply(SpiderContext context, Site currentSite, URL url) {
    String baseURL = context.getBaseURL().toString();
    String currentURL = url.toString();

    boolean equals = false;

    try
    {
        equals = baseURL.equals(currentURL.substring(0, baseURL.length
()));
    }
    catch(Exception e)
    {
        // do nothing here.
    }

    if (equals) {
        return new DecisionInternal(Decision.RULE_ACCEPT, "url accepted
");
    } else {
        return new DecisionInternal(Decision.RULE_IGNORE, "url ignored b
ecause it is not the base directory");
    }
}
}
```

RSS に対応する - 前置き -

RSS ってなかに

RSS っていうのは、Web サイトの見出しや要約などのメタデータを構造化して記述する XML ベースのフォーマットです。1999 年に Netscape 社によって作成されました。各種の blog ツール (CMS といったほうが的確だけど、ここでは分かりやすく blog ツールと表記します) が自動的な RSS の生成機能を持つようになったことを切っ掛けに広まり、現在では asahi.com をはじめとして、多くのニュースサイトや検索サイトが更新情報を RSS で提供するようになっています。

RSS を使用する側のアプリケーションとしては、「RSS リーダ」と呼ばれるものが使われるようになってきています。ユーザが更新をチェックしておきたいサイトの RSS を登録すると、RSS リーダはバックグラウンドで定期的に RSS にアクセスし、もし更新があればそれを知らせてくれます。また、ブラウザのコンポーネントを組み込んでおり、実際に記事にアクセスすることもできます。RSS の更新チェックに特化したものは「ティッカー」と

呼ばれます。

現在では、バラエディ豊かな多くの RSS リーダ&ティッカーがリリースされています。揺籃期のアプリケーションということもあってか、さまざまな試みがなされています。様々なユーザ・インターフェースのものがありますし、ローカルアプリケーションなものだけでなく、Web サービスとして提供されているものも多いです。また、基本的な機能は新着記事を alert するというものなのですが、それに加えて、検索機能やクリッピング機能を乗せていたり、blog への記事の投稿をサポートしたりと、さまざまな機能を取り入れているものがあります。面白いのは personalize 機能で、嗜好を学習させて、自分の好みの(?) 記事を推薦してくれるような仕組みを取り入れているものもあります。Web サービスとして提供されているものの中には、ユーザ同士の学習データの連携を意識したものもあり、興味深いところですよ。いくつか URL を紹介します。

- GoodRelease! (<http://dev.goodrelease.com/>)
- ReadOne (<http://www.readone.net/web/>)

面白いのは GoodRelease で、ユーザ同士の学習データを交換(?) させる仕組みがあります。学習データを「Shell」、交換する相手を「Shell パートナー」と呼ぶそうです。詳しくは公式サイトをご覧ください。とはいってもあんまし詳しいことは書いてありませんが。

RSS にはいくつかのバージョンがあり、統一されていません。このあたりの歴史は、ここでは扱いませんが、興味のある人は調べてみると面白いかも。また、RSS を発展させた規格を採用しようという動きもあります。いくつかの規格があるようですが、もっとも知られたものとしては Atom があります。RSS は一般に blog ツールが自動的に生成するもの

ですが、厄介なことに、ツールによって出力する RSS の形式が異なります。ですから、RSS を相手にするプログラムを書こうとすると、処理対象があらかじめ決まっている場合を除いて、現在つかわれている複数の形式に対応することが求められます。

で、この RSS ですが、この調子で普及していき、中長期的にかなり重要な地位を占めるようになるとおもわれます。理由としては、

- ・その利点が一般ユーザにとって分かりやすい
 - ・多くの RSS リーダが 3 カラム形式のユーザ・インタフェースを採用しており、ユーザにとって抵抗 & 習得の労力が少ない
 - ・RSS を提供するサイトが順調に増加している
 - ・広告のチャネルとして注目されている
- などが挙げられます。

RSS の実例を付録として載せておきます。また、RSS の詳しい仕様は以下の URL で見ることが出来ます。

<http://blogs.law.harvard.edu/tech/rss> (rss2.0, 英語)

<http://web.resource.org/rss/1.0/spec> (rss1.0, 英語)

<http://www.kanzaki.com/docs/sw/rss.html> (日本語)

Informa

RSS は XML をベースにしたフォーマットで、構造も単純であるため、手作業でパースしてもそれほどの手間ではありません。ですが、

- ・規格が統一されておらず、複数の形式に対応したコードを書かなくてはならない
 - ・統一されていない上に混乱しており、将来的に規格の修正や、新しい規格の登場があるかもしれない
- ことから、外部のライブラリを使うのが良い方法だと思われま

で、RSS をパースするライブラリですが、オープンソースなものがいくつかリリースされています。

FeedParser(<http://www.peerfear.org/rss/permalink/2003/11/10/FeedParserAnRSSParser>)

APIForJava/)

- ・ RSS4j(<http://www.churchillobjects.com/c/13005.html>)
- ・ Informa(<http://informa.sourceforge.net/>)
- ・ RSSLibJ(<http://enigmastation.com/rsslibj/>)

今回は、この中から Informa を使いました。これを使うことにした理由としては、対応するフォーマットの多さと、簡潔で使いやすいインターフェースになっていたことがあります。Informa の使い方についてはこれから説明します。

RSS に対応する - 設計 -

ここで実装した処理の流れは次のようなものです。

1. 指定された RSS をダウンロード
2. ダウンロードした RSS をパース
3. URL の一覧をゲット
4. 新着記事かどうか (まだダウンロードされていないか) をチェック
5. 新着記事をダウンロード

また、RSS への対応として、次の 2 つの方針を考えました。

- ・ API への拡張。必要なクラス (RSS パーサクラスとか) を JSpider のオブジェクトモデルに組み込む。
- ・ SPI としての拡張。RSS のパース部分を plugin として実装し、あとは SPI に属するオブジェクト間の通信だけで所望の処理を実現する

RSS 対応に限らず、これから何らかの目的で JSpider を拡張しようとしたときに、この 2 つの選択肢のどちらを選択するかで悩むことは多いと思います。API (=JSpider のオブジェクトモデル) に手を加えていくのは、追加する機能をすでにある構造のなかの適切な箇所に配置できる反面、将来 JSpider のバージョンが上がったときに、新しい JSpider と手元にある改造版 JSpider をマージするという手間が発生します。JSpider くらいの規模のプログラムになると、この作業は相当面倒なのでやりたくないですし、バージョンアップによって大きく内部構造が変更されていたような場合に、ちゃんとマージできるとも限りません。そうなる、新しいバージョンのものをを使うことを放棄するか、新しいバージョンのものに改めて修正を加えることとなります。あまり宜しくありません。

オープンソース・ソフトウェアを hack するときの定石ですが、ソースコードをいじるよりも Adapter を噛ませるべきですし、いじるにしても、なるべくソフトウェアの構造に手を入れないようにすることが望ましいです。今回の RSS 対応の修正の場合、RSS をパースするクラスを作って JSpider のオブジェクトモデルの中に組み込んでいくほうが実装としては自然なのですが、それをすると前述のような問題があります。翻って、static 変数を使うなどして強引に通信経路をつくってやれば、Rule、Plugin、EventFilter の SPI オブジェクトの追加だけで実装することもできます。で、どっちにしたほうが良いのかなあ、ってというのがここでの懸案なわけです。

結果としては、SPI オブジェクトとして追加する方法で実装しました。実装の詳細はこれから説明していきます。

RSS に対応する - 設定 -

設定ファイルは以下のようになっています。同様に、デフォルトで付属している "download" に手を加えた場所のみを順番に抜き出していきます。

jspider.properties

```
# -----  
# Rules Configuration  
# -----  
  
jspider.rules.spider.count=1  
jspider.rules.spider.1.class=net.javacoding.jspider.mod.rule.RenewalBlogArchiveOnlyRule  
  
jspider.rules.parser.count=0
```

ダウンロード時の rule として、"RenewalBlogArchiveOnlyRule"を指定しています。自分で実装したクラスです。更新された blog 記事だけをダウンロードします。また、"download"のほうでは、パース時の Rule として TextHtmlMimeTypeOnlyRule が設定されているのですが、そのままにしておくと RSS まで弾いちゃうので、除外してあります。

plugins.properties

```
# -----  
# Plugin Configuration  
# -----  
  
jspider.plugin.count=3  
jspider.plugin.1.config=console  
jspider.plugin.2.config=diskwriter
```

```
jspider.plugin.3.config=rssparser
```

3つ目のプラグインとして"rssparser"を追加しています。

plugins/rssparser.properties

```
plugin.class=net.javacoding.jspider.mod.plugin.rssparser.RSSParserPlugin  
  
plugin.config.output.absolute=false  
plugin.config.output.folder=./  
  
plugin.filter.enabled=true  
  
plugin.filter.engine=net.javacoding.jspider.mod.eventfilter.AllowNoneEventFilter  
plugin.filter.monitoring=net.javacoding.jspider.mod.eventfilter.AllowNoneEventFilter  
plugin.filter.spider=net.javacoding.jspider.mod.eventfilter.AllowAllEventFilter
```

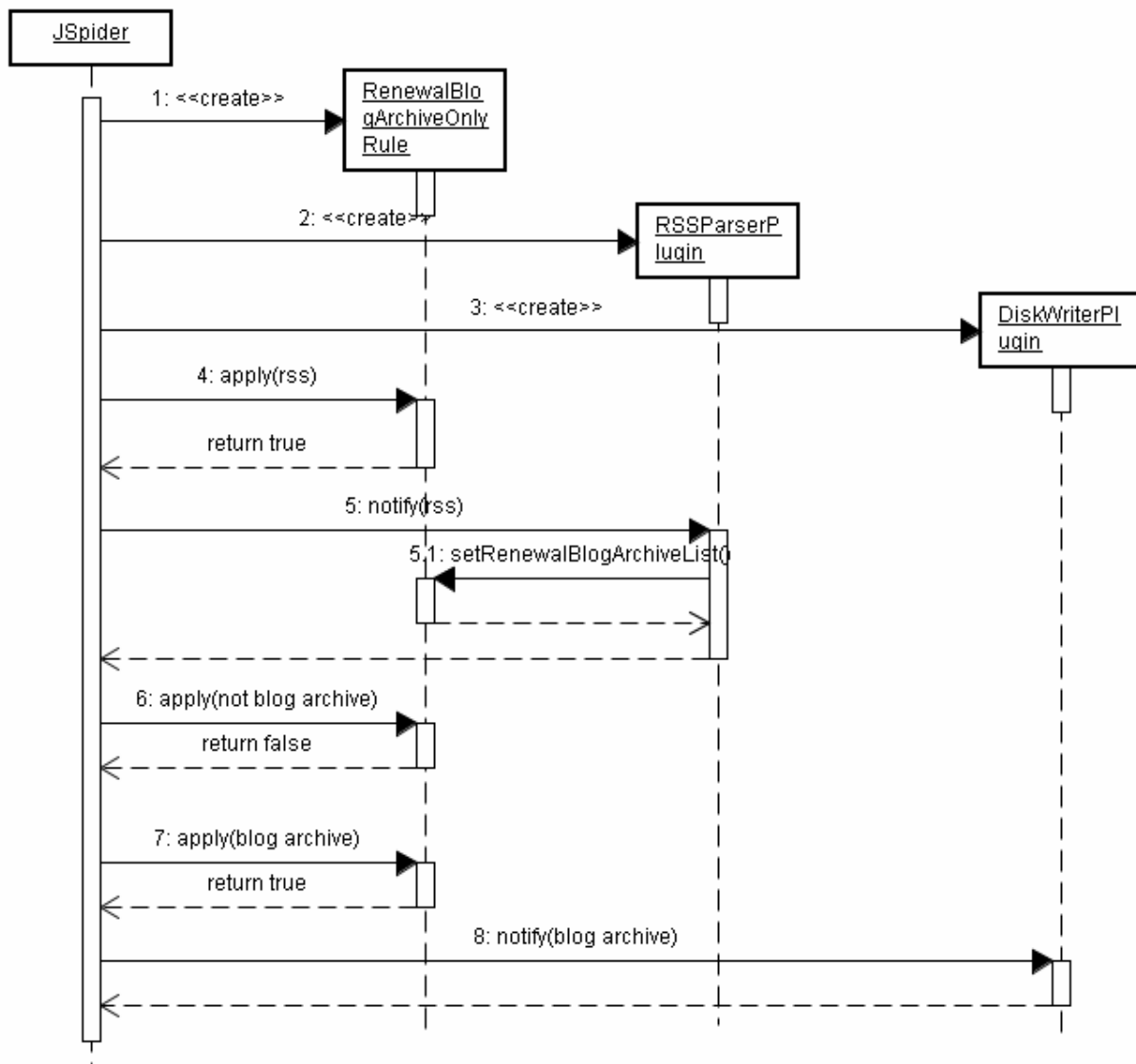
RSSParserPlugin クラスってのが RSS をパースする処理を担当します。

plugins/diskwriter.properties

```
plugin.filter.spider=net.javacoding.jspider.mod.eventfilter.BlogArchiveOnlyEventFilter
```

RSS に対応する - 実装 -

まず、UML のシーケンス図を示します。



起点には RSS の URL を指定します。いい加減な実装をしているので、それ以外のものを指定するとうまく動きません。RenewalBlogArchiveOnlyRule はいちばん最初に渡された URL を無条件で accept するようになっています。よって RSS はダウンロードされ、RSSParserPlugin に渡されます。RSSParserPlugin は RSS をパースし、blog 記事の URL の一覧を取得します。また、それぞれについてすでにダウンロード済みであるかをチェックし、新着記事のリストを作成します。作成されたリストは static 変数を通じて RenewalBlogArchiveOnlyRule に渡されます。以後は、このリストにある URL は accept し、それ以外のものはすべて reject するという動作をするようになります。また、accept された URL はダウンロードされ、DiskWritePlugin によってストレージに書き込まれます。

RenewalBlogArchiveOnlyRule クラスの実装です。これがダウンロード時の Rule として設定されます。static 変数として ArrayList を宣言しており、これを使って RSS パーサから blog の新着記事の URL を受け取ります。

```
public class RenewalBlogArchiveOnlyRule implements Rule {

    public static ArrayList RenewalBlogArchiveList = null;
    private static boolean isFirstApply = true;

    public RenewalBlogArchiveOnlyRule() {
    }

    /* (非 Javadoc)
     * @see net.javacoding.jspider.spi.Rule#getName()
     */
    public String getName() {
        // TODO 自動生成されたメソッド・スタブ
        return null;
    }

    public ArrayList getRenewalBlogArchiveList(){
        return RenewalBlogArchiveList;
    }

    public void setRenewalBlogArchiveList(ArrayList value){
        RenewalBlogArchiveList = value;
    }

    /* (非 Javadoc)
     * @see net.javacoding.jspider.spi.Rule#apply(net.javacoding.jspider.core.SpiderContext, net.javacoding.jspider.api.model.Site, java.net.URL)
     */
    public Decision apply(SpiderContext context, Site currentSite, URL url) {
        // TODO 自動生成されたメソッド・スタブ

        //最初に渡される URL は RSS のものであるとする
        if ( isFirstApply ){
            isFirstApply = false;
            return new DecisionInternal(Decision.RULE_ACCEPT, "maybe rss - accepted" );
        }

        if (null == RenewalBlogArchiveList){
            //どーしたもんだらう
        }

        Object obj = null;
        ListIterator iterator = RenewalBlogArchiveList.listIterator();
        while ( iterator.hasNext() ){
            obj = iterator.next();

            if ( obj instanceof URL ){
                URL currentURL = (URL)obj;
                if ( currentURL.equals(url) )
                    return new DecisionInternal(Decision.RULE_ACCEPT, "renewal blog archive - accepted" );
            }
        }

        return new DecisionInternal(Decision.RULE_IGNORE, "ignored" );
    }
}
```

```
}  
}
```

RSSParserPlugin クラスの実装です。前述の通り、RSS のパースには Informa を使っています。

```
public class RSSParserPlugin implements Plugin {  
  
    public static final String OUTPUT_ABSOLUTE = "output.absolute";  
    public static final String OUTPUT_FOLDER = "output.folder";  
  
    public static final String DEFAULT_OUTPUT_FOLDER = ".";  
  
    protected File outputFolder;  
    protected File baseFolder;  
  
    public RSSParserPlugin(PropertySet config) {  
        JSpiderConfiguration jspiderConfig = ConfigurationFactory.getConfiguration  
(  
            );  
        File defaultOutputFolder = jspiderConfig.getDefaultOutputFolder();  
        if (config.getBoolean(OUTPUT_ABSOLUTE, false)) {  
            outputFolder = new File(config.getString(OUTPUT_FOLDER, DEFAULT_OUTP  
UT_FOLDER));  
            baseFolder = new File("/");  
        } else {  
            outputFolder = new File(defaultOutputFolder, config.getString(OUTPUT  
_FOLDER, DEFAULT_OUTPUT_FOLDER));  
            baseFolder = new File(".");  
        }  
    }  
  
    /* (非 Javadoc)  
    * @see net.javacoding.jspider.mod.api.Plugin#getName()  
    */  
    public String getName() {  
        // TODO 自動生成されたメソッド・スタブ  
        return null;  
    }  
  
    /* (非 Javadoc)  
    * @see net.javacoding.jspider.mod.api.Plugin#getVersion()  
    */  
    public String getVersion() {  
        // TODO 自動生成されたメソッド・スタブ  
        return null;  
    }  
  
    /* (非 Javadoc)  
    * @see net.javacoding.jspider.mod.api.Plugin#getDescription()  
    */  
    public String getDescription() {  
        // TODO 自動生成されたメソッド・スタブ  
        return null;  
    }  
  
    /* (非 Javadoc)  
    * @see net.javacoding.jspider.mod.api.Plugin#getVendor()  
    */  
    public String getVendor() {  
        // TODO 自動生成されたメソッド・スタブ  
        return null;  
    }  
}
```

```
/* (非 Javadoc)
 * @see net.javacoding.jspider.api.event.EventSink#initialize()
 */
public void initialize() {
    // TODO 自動生成されたメソッド・スタブ
}

/* (非 Javadoc)
 * @see net.javacoding.jspider.api.event.EventSink#shutdown()
 */
public void shutdown() {
    // TODO 自動生成されたメソッド・スタブ
}

/* (非 Javadoc)
 * @see net.javacoding.jspider.api.event.EventSink#notify(net.javacoding.jspider.api.
event.JSpiderEvent)
 */

static boolean isExecuted = false;
public void notify(JSpiderEvent event) {
    //
    // CAUTION!
    // this section is executable only once!
    //

    ChannelIF channel = null;
    URL url = null;

    if (event instanceof ResourceFetchedEvent && !isExecuted) {
        ResourceFetchedEvent rfe = (ResourceFetchedEvent) event;

        //parse the rss resource.
        try{
            channel = FeedParser.parse(new ChannelBuilder(), rfe.getRes
source().getInputStream());
        }
        catch (Exception e){
            //fatal error!
        }

        //instancing
        if ( null == RenewalBlogArchiveOnlyRule.RenewalBlogArchiveList ){
            RenewalBlogArchiveOnlyRule.RenewalBlogArchiveList = new Arr
ayList();
        }

        //extract the URL from each rss item.
        ItemIF[] items = (ItemIF[])channel.getItems().toArray(new ItemIF
[0]);

        for (int i=0; i<items.length; i++){
            url = items[i].getLink();

            //convert: url -> file path
            String path = url.getHost() + url.getPath();
            if (includesFile(url)) {
                path = url.getHost() + url.getFile();
            } else {
                path = url.getHost() + url.getPath() + "/index.htm
l";
            }
        }
    }
}
```

```
er all.
//if the url(=blog archive) isn't already downloaded,
//add to the arraylist in order to download the resource aft

File outputFile = new File(outputFolder, path);
if ( !outputFile.exists() ){
    RenewalBlogArchiveOnlyRule.RenewalBlogArchiveLis
t.add(url);
}
}
isExecuted = true;
}
}

protected boolean includesFile(URL url) {
    String urlString = url.getPath();

    //はてなダイアリーなら、常に true を返す
    if (url.getHost().indexOf("d.hatena.ne.jp") != -1) return true;

    return (urlString.lastIndexOf(".") > urlString.lastIndexOf('/'));
}
}
```

- ・ これからの予定

クローラ勉強会の今後の予定です。あと3回でまとめようと思っています。

- ・ 第5回 「API」

JSpider のオブジェクトモデルをざっと眺めていきます。

また、micro-tasks の仕組みと、それに関連してスレッドの取り扱いを見ます。

- ・ 第6回 「Core」

JSpider の低レベルな実装について、要所要所を抜き出して見ていきます。

- ・ 第7回 「hack #2 ~RSS リーダをつくろう!~」

みんな(ただしやりたいひとだけ)で JSpider を弄くって RSS リーダを作りましょう:-)

RSS リーダというのはガイドランスのときに挙げた案に過ぎないので、他に作りたいものがあればそれを作りましょう。

で、今岡くんが作成した RSS リーダ(または他のなにか) について発表します。

```
<?xml version="1.0" encoding="EUC-JP"?>

<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:sy="http://purl.org/rss/1.0/modules/syndication/"
  xmlns:admin="http://webns.net/mvcb/"
  xmlns:cc="http://web.resource.org/cc/"
  xmlns="http://purl.org/rss/1.0/">

<channel rdf:about="http://www.seman.cs.uec.ac.jp/~onai/blog/">
<title>金八先生の blog</title>
<link>http://www.seman.cs.uec.ac.jp/~onai/blog/</link>
<description></description>
<dc:language>en-us</dc:language>
<dc:creator></dc:creator>
<dc:date>2004-07-30T14:10:01+09:00</dc:date>
<admin:generatorAgent rdf:resource="http://www.movabletype.org/?v=2.64" />

<items>
<rdf:Seq><rdf:li rdf:resource="http://www.seman.cs.uec.ac.jp/~onai/blog/archives/000840.html" />
<rdf:li rdf:resource="http://www.seman.cs.uec.ac.jp/~onai/blog/archives/000833.html" />
<rdf:li rdf:resource="http://www.seman.cs.uec.ac.jp/~onai/blog/archives/000768.html" />
<rdf:li rdf:resource="http://www.seman.cs.uec.ac.jp/~onai/blog/archives/000767.html" />
<rdf:li rdf:resource="http://www.seman.cs.uec.ac.jp/~onai/blog/archives/000721.html" />
</rdf:Seq>
</items>

</channel>

<item rdf:about="http://www.seman.cs.uec.ac.jp/~onai/blog/archives/000840.html">
<title>円や三鷹荘</title>
<link>http://www.seman.cs.uec.ac.jp/~onai/blog/archives/000840.html</link>
<description>久しぶりに三鷹に出た。知人の送別会である。円や三鷹荘だったかな、店の名は。創作料理だそう
だ。照明がかなり暗くて何を食べているか、よくわからない。料理は色とりも大切なのに。刺身なんか活きがいいのか
どうかわからない。冷酒が竹筒みたいなものに入ってくる。中はちゃんと洗っているのかなあ。冷酒の入れ物はガラス
が好きだ。あと、トイレが一つしかない。ビールを呑んでいる時は辛い。...</description>
<dc:subject></dc:subject>
<dc:creator>kinpachi</dc:creator>
<dc:date>2004-07-30T14:10:01+09:00</dc:date>
</item>

<item rdf:about="http://www.seman.cs.uec.ac.jp/~onai/blog/archives/000833.html">
<title>泡盛</title>
<link>http://www.seman.cs.uec.ac.jp/~onai/blog/archives/000833.html</link>
<description>幻の泡盛と言われる波照間島で作られている泡波を呑んだ。お呼ばれしたご家庭で出された。波照
間島に行ってきたのだそうだ。オンザロックで呑んだ。すっきりした呑み口だった。つまみは、プロシュート入り生
春巻き、かつおのたたき、ビーフステーキ、チキンネックの炭火焼、海ぶどう。うまかった。...</description>
<dc:subject></dc:subject>
<dc:creator>kinpachi</dc:creator>
<dc:date>2004-07-29T12:04:16+09:00</dc:date>
</item>

<item rdf:about="http://www.seman.cs.uec.ac.jp/~onai/blog/archives/000768.html">
<title>そば</title>
<link>http://www.seman.cs.uec.ac.jp/~onai/blog/archives/000768.html</link>
<description>僕はそばが好きだ。深大寺の深水庵の冷やしたぬきが好きだ。一年中、冬でも食べる。最近、伊豆
長岡の橋本という蕎麦屋で食べた。もうすぐ夕食だったので、評判のかき揚げそばはあきらめた。合いもりという、
そばとうどんの両方がのった、もりを食べた。そばの方がおいしかった。...</description>
<dc:subject></dc:subject>
<dc:creator>kinpachi</dc:creator>
<dc:date>2004-07-22T13:26:51+09:00</dc:date>
</item>
```

```
<item rdf:about="http://www.seman.cs.uec.ac.jp/~onai/blog/archives/000767.html">
<title>日帰り温泉</title>
<link>http://www.seman.cs.uec.ac.jp/~onai/blog/archives/000767.html</link>
<description>僕は日帰り温泉が好きだ。最近は、山歩きのと、寄る。でも、寄る年波で、温泉の名を覚えてい
られない。ごめん。...</description>
<dc:subject></dc:subject>
<dc:creator>kinpachi</dc:creator>
<dc:date>2004-07-22T13:21:55+09:00</dc:date>
</item>
<item rdf:about="http://www.seman.cs.uec.ac.jp/~onai/blog/archives/000721.html">
<title>はじめに</title>
<link>http://www.seman.cs.uec.ac.jp/~onai/blog/archives/000721.html</link>
<description>久しぶりに日記を書くことになった。この内容は口外しないこと。あはは。今日は、今学期の講
義が全部終わって、ほっとした。早く帰って、ビールを呑もう。...</description>
<dc:subject></dc:subject>
<dc:creator>kinpachi</dc:creator>
<dc:date>2004-07-16T15:51:56+09:00</dc:date>
</item>

</rdf:RDF>
```