

## 本日のお品書き

### 1. 導入

- ・ クローラってなに？
  - クローラとは
  - 世の中にはどんなクローラがあるか
- ・ なにが難しいの？
  - もっとも単純なクローラ
  - もっとも単純なクローラではどこがだめか
- ・ さらにその先にある問題
  - google のクローラ
  - ハードウェアと OS
  - Web の広大さ、そしてその「構造」について
- ・ クローラに求められる振る舞いとは？
  - 大前提 ~ クローラは慎み深くなければならないということ
  - robots.txt を尊重する
  - 必要なものを適切な頻度で取得する
- ・ クローラを作ったら？
  - 名前を付けよう
  - User-Agent を設定しよう
  - Web 上にページを作ろう
  - クローラを登録しよう
  - なぜこんなことをするの？
- ・ クローラについてより深く学ぶには

### 2. JSpider

- JSpider について
  - 概要
  - 今回読むソースコードについて
  - JSpider をビルドする
  
- JSpider の構造
  - 概要図
  - Rule の仕組み
  - Event の仕組み
  - Plugin の仕組み
  - micro-tasks
  - JSpider の 2 つの使い方
  
- 今回のネタ元

## 1. 導入

- ・クローラってなに？

### クローラとは

クローラとは、WWW 上の文書や画像などを自動的に蒐集するプログラムのことです。一般に、Web サーバから HTML 文書を取得し、HTML 文書中に含まれるリンクを取り出し、リンクを辿って別の HTML 文書を収集していくという機能を持ちます。場合によっては、蒐集したデータを格納するデータベースや、データを何らかの方法で構造化するインデクサまでを含めてクローラと呼ぶこともあります。

この勉強会ではクローラを「クローラ」と呼びますが、クローラが常に「クローラ」と呼ばれるわけではありません。クローラは以下のようにも呼ばれることがあります。

- ・ロボット(robot, bot)
- ・スパイダ(spider)
- ・ワーム(worm)
- ・アント(ant)

昔はロボットと呼ばれることが多かったのですが、最近ではクローラという呼び方が主流になりつつあります（よって、この勉強会でもクローラという呼称を用います）。また、クローラのスペルは crawler ですが、日本語で表記するときには「クロウラー」などとされることもあります。

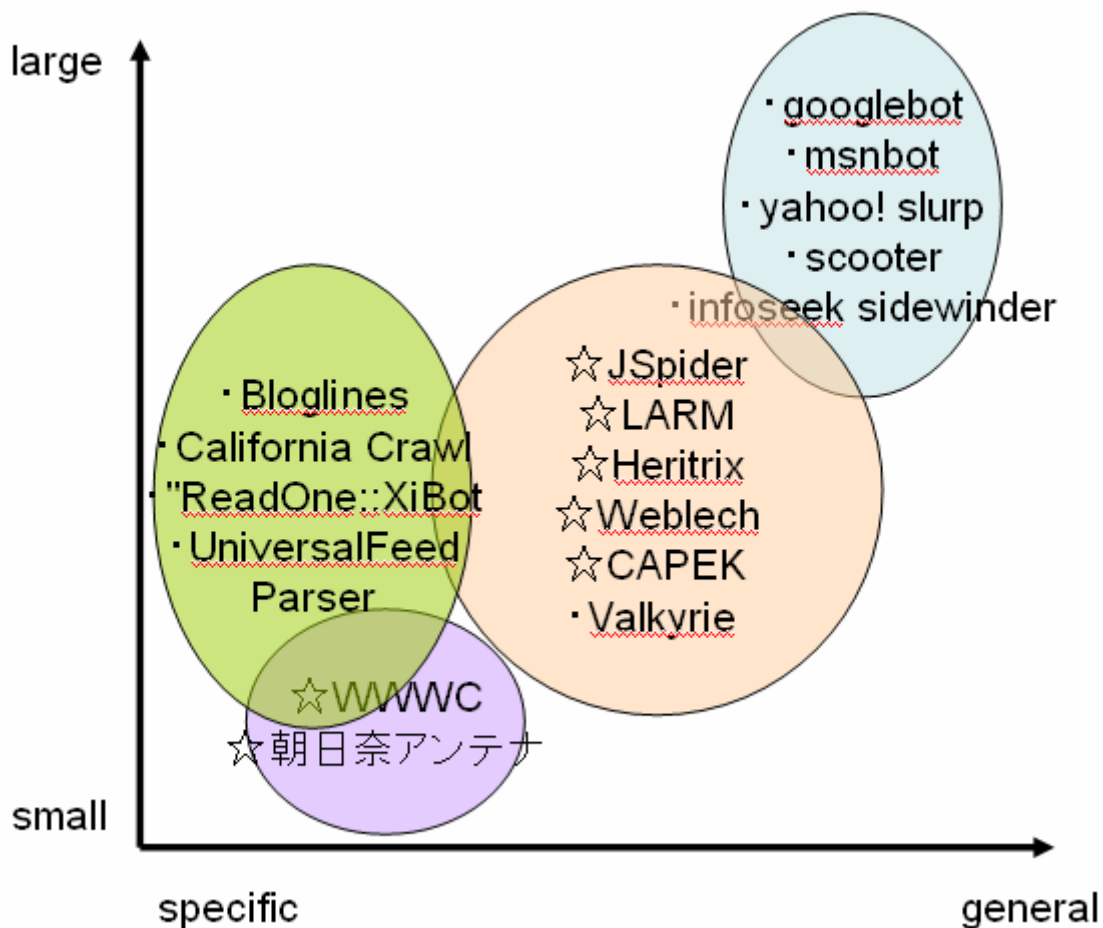
クローラがそれ単体で完結したシステムとして使われることはあまりありません。個人的な用途、たとえばエロサイトからエロ画像をまとめてダウンロードするような場合や、登録したサイトの更新を自動的にチェックする「アンテナ」と呼ばれるようなプログラムでは、クローラそれ単体で機能が完結します。しかし、多くの場合ではクローラはもっと大きなシステムの一部として使われます。もっとも典型的なものとしては、google や yahoo! に代表されるロボット型の検索エンジンがあります。検索エンジンというシステムの中で、クローラは広大な WWW からひたすらリソースを取得してくる役割を負います。また、特定のサイトに特化した検索エンジンを提供する 『オークション統計ページ(仮)』のようなサイトや、blog 検索エンジンのような、特定の領域を対象とした検索エンジンでも、クローラは活躍してします。

ところでブラウザをクローラとは呼びませんが、何故かというとブラウザは人間がモニタの前に座って操作するからです。一般に、「クローラかそうでないか」の区別は、人間が

操作をするかどうかで為されます。たとえば、ブラウザを使って自動的にどこかのサイトにアクセスしてリソースを拾ってくるようなプログラムを書いたら（IEのコンポーネントを使えば簡単に作れます）、それはクローラに分類されるでしょう。

世の中にはどんなクローラがあるか

一口にクローラと言っても、その目的や規模は千差万別です。そこで、おおまかな見通しを掴むため、主要なクローラをざっと二次元座標にプロットしてみました。



表の見方を説明します。横軸がそのクローラがどれだけ特定の目的に特化しているかを、縦軸がそのクローラの扱えるリソースの規模を表しています。名前の前に"・"が付いているのは実装が公開されていないもの、"☆"がついているのは公開されているものです。

また、この二次元座標の上で、クローラを大きく4つのグループに分類しました。

・青い円.....でかいクローラ。Google や yahoo!に代表される大規模な検索エンジンで使われているクローラ。WWW に存在するすべてのリソースを蒐集することを究極の目標として何十億のオーダーのリソースをかき集める。

・オレンジ.....小さいクローラ。指定した一つのサイト、あるいはそこから繋がるいくつかのサイトをダウンロードするようなクローラ。Heritrix や CAPEK あたりはかなり大きな規模のリソースを落として来られるような気がします。たぶんだけど。

・紫.....いわゆるアンテナ。ユーザが列挙した URL に定期的にアクセスし、更新があったらお知らせする。

・緑色.....特定の目的に強く特化したクローラ。Blog 検索エンジンや、画像検索エンジンでは、限定されたリソースのみを取得してきたいという要求があります。

・なにが難しいの？

クローラなんて、いかにも簡単に作れそうな気がしませんか。「ただリンクを辿っていけばいいんだろ、超絶簡単だよ」、「サルでも書けるぜ（もちろんおれさまなら楽勝で書けるぜ）」なんて思ったりしてませんか？ .....しかしながら、クローラは実際にはそれほど簡単なものではありません。それでは、なにが難しいのか。クローラを作るときに問題になることは何なのか。この節ではそれを明らかにしていきます。

ここでは、まず、もっとも単純に実装されたクローラを想定し、それではなにがまずいのかを考えることで、クローラのはなにが難しいのか説明していこうとおもいます。

### もっとも単純なクローラ

リンクを辿ってリソースを蒐集する機能を実装した、もっとも単純なクローラを考えてみましょう。処理の流れとしては次のようなものになるでしょう。

- 1 . どこか起点となる URL を設定する
- 2 . HTTP プロトコルを用いてページをゲットする。

3. ページに含まれるリンクを、正規表現か何かを使って取り出す。取り出したリンクはキューにでも貯めておく

4. 以下、2~3を繰り返して、どんどんリンクを辿っていく

これだけの処理を実装するのは簡単です。実際に書いてみたわけではありませんが、perlを使えば100行も使わずに書けるでしょう。

.....ですが、こんな単純な実装では、たくさんの問題が出てきます。次の節で、それらを具体的に列挙してみます。

#### もっとも単純なクローラではどこがだめか

- ・リソースの取得に失敗したときのことを考慮していない

リソースの取得に常に成功するわけではありません。相手先のサーバは何かの原因でダウンしているかもしれませんから、タイムアウトの処理を入れる必要があるでしょう。ひょっとしたら既にそのリソースはサーバから削除されているかも知れません。もしそうなら、サーバは404エラーを返すでしょうから、それを適切に扱ってやらなくてはなりません。エラーコードは404だけではありません。それ以外のエラーを返したときにはどうすれば良いでしょうか。

- ・アクセスの頻度が管理されていない

単純に、取り出したリンクに次々とアクセスを仕掛けるという実装をすると、時として、クローラの挙動は相手のWebサーバへのattack(DoS)になってしまいます。これは人様に迷惑をかけるという点で非常にまずい振る舞いです。

よって、あるサーバに対するアクセスに一定の制限(たとえば1秒に1アクセスまで、とか)を掛けて、それ以上の頻度でアクセスすることのないように、何らかの方法で管理してやらなくてはなりません。

- ・すでに蒐集済みのページも集めてきちゃう

あるHTMLをゲットしてきて、そこからリンクを取り出したとして、ひょっとしたら、そのリンクはすでに蒐集済みであるかもしれませんよね。すでに蒐集されているリソースを再度取りに逝ってしまうのは非効率でアホなおまぬけな実装です。なので、あるリンクが、すでに蒐集されているかどうかをチェックする仕組みが必要になります。

単純な実装としては、蒐集済みのURLをリストや配列として保持しておいて、新しい

URL へアクセスするたびに、すでに蒐集済みでないかをチェックしてやるというやりかたがあります。もうちょっとしっかりした方法としては、ハッシュを使う、データベースを用いるなどが考えられます。

- ・同じページを集めて来ちゃう

たとえば、

```
www.seman.cs.uec.ac.jp/~shin/blog/  
www.seman.cs.uec.ac.jp/~shin/blog/index.html  
www.seman.cs.uec.ac.jp/~shin/blog//index.html  
www.seman.cs.uec.ac.jp/%7eshin/blog/  
ace.seman.cs.uec.ac.jp/~shin/blog/  
130.153.150.4/~shin/blog/
```

これらの URL はすべて同じリソースを指します。URL は違うのに、どこかの Web サーバにあるまったく同じファイルを指します。ですから、同じリソースを集めてしまったということを検出して、重複した分を除外することができないと、まったく同じリソースをいくつも蒐集してしまうケースが発生することになります。単純に蒐集済みの URL と比較するだけでは、この問題を解決することはできません。

そのため、URL を解釈し、このような「違う URL が同じリソースを指す」問題を解決するコードを加えてやるべきです。一般に、URL を一定の規則に従って変換し、内部的に多重性のない表現形式にして扱ってやるという実装をするようです。また、この「URL を一定の規則に従って変換する処理」を指して"normalize"と呼ぶようです。他には、ある程度処理コストを掛けることができるなら、取得したリソースについてハッシュ値を計算し、それ以前に蒐集されたリソースのハッシュ値と比較することによって、重複を検出することができるかもしれません。

いずれにせよ、重複の検出は非常に難しい問題です。Google などは非常に上手くこれをこなしているという印象を受けますが、どんな方法でやっているのか、是非知りたいものです。

- ・ネットワークの帯域を有効に利用できない

もっとも単純なクローラはシングルスレッドで実装されています。.....ですが、これではネットワークの帯域を有効に利用することができません。たった一つのスレッドがレスポンスの遅い Web サーバにアクセスしているとき、ネットワークの帯域はその大部分が遊んでいます。クローラの性能の指標の一つに、スループットの高さ、つまり「ある時間内に、どれだけ多くのリソースを蒐集してこれるか」ということがあります。ネットワーク帯域を有効に利用することはとても重要です。

これには、ふつう、マルチスレッド化、つまり、リソースを取得するスレッドを複数走

らせるという解決策が取られます。JSpider は割合綺麗にこの辺りの処理を実装しているので、参考になる部分があるかと思います。

- ・ さらにその先にある問題

- google のクローラ

googlebot や yahoo! slurp のような、広大な WWW 全体を集めてこようとするクローラの場合には、また違ったスケールの問題が出てきます。それについて考えてみましょう。

Googlebot は、次の 2 点を目標として作られています (.....と言って良いでしょう)。

- ・ Web に存在するすべてのリソースを蒐集すること
- ・ それらのリソースを常に新鮮な状態に保つこと

1 点目については説明の余地もないでしょう。Web に存在する”すべての”リソースを集めるのは、実際には極めて難しいことですが、googlebot の究極の目標はそこにあると言えると思います。

2 点目についてです。Web 上のリソースは常に更新され続けているため、今日蒐集したリソースは明日になればもう古くなっている可能性があります。よって、蒐集済みの URL であっても定期的に巡回し、手元のデータを新鮮な状態に保つ必要があります。これもやはり容易ではない問題です。

この 2 つの目標に挑もうとしたときに、どういった事柄が問題になってくるのでしょうか。

- ハードウェアと OS

現在、google は約 80 億、yahoo は約 190 億もの Web ページをインデクシングしているといえます。これだけの量のデータを扱うには、膨大な CPU パワーとストレージが必要になってきます。Google は、巨大な検索システムを、耐障害性の高い OS・ミドルウェアと、何千台もの安価なマシンとで実現しています。

Google の持つ OS (分散ファイルシステム) としては、Google File System(GFS)がよく知られています。数百 TB クラスの広大なストレージを、沢山の安価なマシンで構築しています。GFS については以下の URL や論文を参考にしてください。

<http://labs.google.com/papers/gfs.html>

<http://www.moodindigo.org/blog/archives/000200.html>

<http://www.radiumsoftware.com/0404.html#040406>

他、google については分散プログラミングモデル MapReduce 等が知られています。Yahoo や MSN でも、おそらく同じようなソフトウェアや PC クラスタを開発していると思います。

- Web の広大さ、そしてその「構造」について

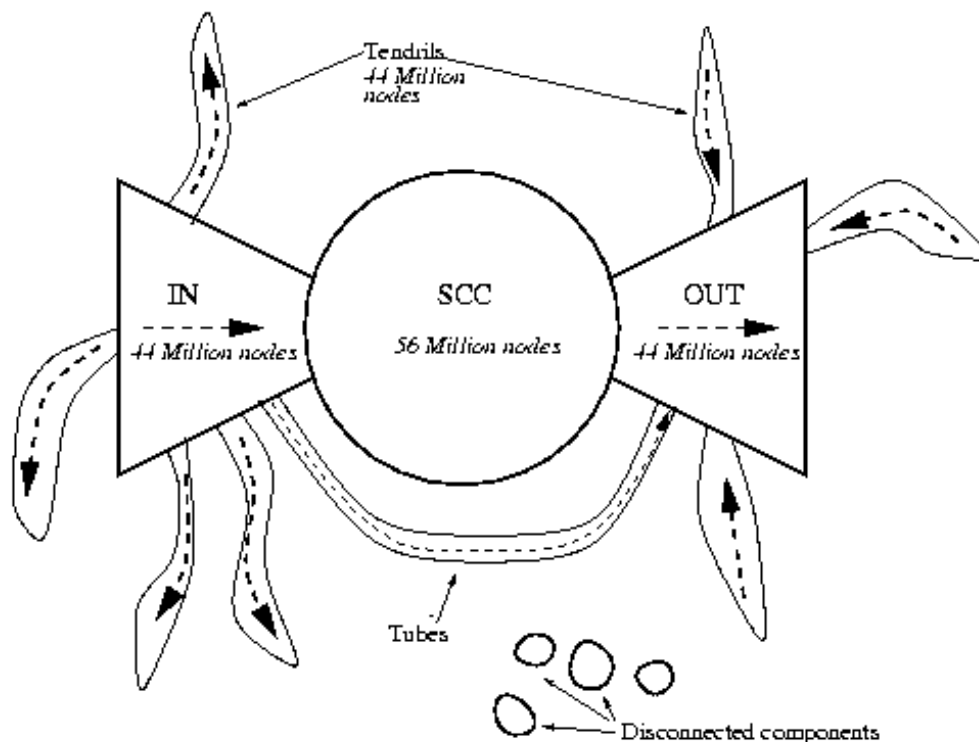
yahoo が 190 億もの Web ページをインデックスしていると聞くと、それが Web の全てであるかのようにも思えてしまいますが、そんなことはありません。Web はそれよりもはるかに広大で、奥深いものです。

クローラを用いる検索エンジンの検索範囲を推定してきた S. Lawrence らは、彼らの論文の中で以下の調査結果を報告しています。

- ・ 個々の検索エンジンについて見たとき、その検索範囲は Web の 5% から 30% である
  - ・ 11 の主要なサーチエンジンを合わせても、その検索範囲は Web の 50% に満たない
- 11 の主要なサーチエンジンをあわせても、Web のたった半分にもならないというのは、意外な感じがするのではないのでしょうか。それは何故でしょう。

先ず、Web の構造について触れましょう。Web には構造があります。一つ一つの Web ページはただ無造作にリンクを貼っているだけでも、Web 全体を調べると、それらのリンクがとても興味深い構造を作っていることが分かってきます。

AltaVista の技術者（当時）であった A. Broder らは、彼らの論文の中で、およそ 5 億の Web ページのリンク構造を解析した結果を報告しています。下の図を見て下さい。



中央の、“SCC”とある丸い部分、ここが Web の中核を為す領域です。多くのリンクを貼り、また自らも多くのリンクを集めるような、典型的にはポータルサイトのような Web ページによって構成される集合です。その両側には、“IN”と“OUT”があります。IN は、SCC に対してリンクを貼っているが、SCC からはリンクされていない Web ページ、OUT は逆に、SCC からリンクされているが、SCC へのリンクは貼っていない Web ページの集合です。また、IN や OUT にくっついたような“Tendril”や、SCC を迂回して IN と OUT を繋ぐ“Tubes”があります。最後に、右下のようにある離れ島のような小さな丸“Disconnected components”、これは SCC を中心とした大きな集合に加わらない、いわば分離された集合です。

では、クローラの立場から考えてみましょう。実際、上の図を見ると、どこか大きな Web サイト(たとえば Yahoo! Japan としましょう)から出発して延々とリソースを集めていっても、Web 全体を蒐集することができるわけではないということが分かるでしょう。Yahoo! Japan のようなサイトは SCC に含まれる筈です。そこで、SCC に含まれるページを出発点としてクローリングをします。すると、蒐集できるのは SCC と OUT に含まれるページだけで、それ以外の IN, Tendril, Tubes, Disconnected components には至ることができません。Web ページにおけるリンクというのは片方向にしか行けないので、こういうことになってしまいます。

では、どうしたら良いでしょう。もっとも典型的な解決策は、ユーザに教えてもらうことです。たとえば google の提供しているツールバーがあります。あれをインストールするときに、「あなたの見ている Web ページの情報を google に提供しても良いですか？」みたいな選択肢が出てきます。あれに yes と答えると、それ以降、google ツールバーは、あなたがどこかの Web ページにアクセスするたびに、その URL を google のサーバに送信します。これによって、google は IN や Tendril, Tubes に含まれるような、さらには Disconnected components に含まれる URL をも知ることができるわけです。そうやって、ツールバーのユーザが教えてくれた URL を巡回対象に入れることで、クローリングの網羅性を向上させています。

次に、Hidden Web(Deep Web, Invisible Web)などと呼ばれることもあります)について触れましょう。Web には、クローラでは取得できない Web ページが数多くあります。たとえば、中にはいるのにユーザ認証が必要なページや、問い合わせインタフェースを介して情報を取得するようなページです。そのようなページの中にはコンテンツとして価値が高いものが少なからずあるんですが、クローラはそれらを取得することが出来ません。これも「Web 全体を取得する」という目的の妨げとなります。

.....ところでこの章で書くべきことは他にもあるんですが、面倒になってきたのでこのあたりで止めにして、次に行きますかね。

#### ・クローラに求められる振る舞いとは？

大前提 ~クローラは慎み深くなければならないということ

まず、前提として理解しておいてほしいことがあります。クローラってのは、つねに慎み深くなければならないということです。

何故でしょう。クローラは必然的に、相手の Web サーバに負荷を与えますし、相手のネットワークの帯域を消費します。サーバにしる回線にしるタダではありません。クローラを使う側のモラルの問題もあります。残念ながら、クローラの使用者のなかにはモラルを欠いた人たちが少なくありません。近年では電子メールアドレス収集業者などもクローラを利用し、問題となっています。また、クローラによって蒐集されたコンテンツが、倫理的、あるいは法的に、正当に扱われるという保証はどこにもありません。よって、あな

たのクローラが、相手先のサイトに大歓迎で迎えられるようなことはまずありません。

ところで、サイト内に、クローラによるコンテンツの蒐集を拒否する文言を載せているサイトも少なくありません。たとえば、『Wikipedia』のサイト内には次のような記述があります。

記事を大量にダウンロードするためにクローラを使わないで下さい。強引なクローリングは、ウィキペディアが劇的に遅くなる原因となります。

また、利用規約のなかで、クローラの仕様を禁止しているサイトもあります。オンラインオークションサイト『eBay』では、規約の中に以下の文言を含めています。

ロボット、スパイダ、スクレーバ、その他の自動化手段による本サイトへのアクセスは、書面による許諾なしには、どのような目的であっても許可されないことに合意したものとします。

クローラを走らせるときには、とりわけコンテンツでメシを食っている相手へのクローリングには注意をしたほうが良いでしょう。たとえば新聞社などがこれに該当します。とはいえ、クローラを避けたいサイトはほとんどのケースで robots.txt を設置しているため、これを遵守する限りは、問題が起こることはまずないとおもいますが。

### robots.txt を尊重する

robots.txt とは、クローラに対して、その Web サイトのリソースを取得して良いかどうかを指定するものです。通常、Web サイトの/robots.txt (ルート直下。尾内研なら <http://www.seman.cs.uec.ac.jp/robots.txt>)に置かれます。

robots.txt の書式は非常に単純です。"User-agent:"にクローラの名称を指定し、"Disallow:"に許可しない部分を指定します。たとえば、

```
User-Agent: Googlebot
Disallow: /cgi-bin/
```

とすると、googlebot に対して、cgi-bin/¥以下へのアクセスを禁止する、という意味になります。また、User-Agent と Disallow の組は、ひとつの robots.txt のなかにいくつで

も書くことができます。

すべてのクローラに対し、あらゆるアクセスを拒否する場合には、以下のように書きま  
す。

```
User-agent: *  
Disallow:/
```

robots.txt の指示に従うことは、クローラにとって最低限のマナーとされています。

### 必要なものを適切な頻度で取得する

あなたのしたいことに何が必要なかを考慮しましょう。HTML だけがあればいいなら HTML だけを取得し、画像やその他のリソースは無視するようにしましょう。

また、せいぜい一ヶ月に一度しか更新されない Web サイトに対して、一日に何回ものペースでクローリングを掛けるのはよくありません。特定の Web サイトに定期的にクローリングを掛ける場合、相手の更新頻度を考えるようにしましょう。

### ・クローラを作ったら？

さて。あなたが最高にナイスなクローラを作ったとしましょう。おめでとうございます。今すぐそのナイスで最高なクローラを 24 時間体制で走らせまくりたいところですが、ちょっと待ってください。あなたにはその前にすべきことがあります。「何を？」。この節ではそれを御説明致します:-)

### 名前を付けよう

当たり前ですが、まず、あなたが作ったクローラに名前を付けましょう。すでに公開されているクローラを hack して作ったものであったとしても、元となったクローラの名前を名乗るのではなく、ちゃんと新しい名前を付けてやるべきです。

名前はクローラの概要と目的がわかるようなものとナイスです。あなたのクローラにステキな名前を付けたら、次に進みましょう。

### User-Agent を設定しよう

User-Agent とは、クライアントが Web サーバに対して送信する環境変数の一つです。クライアントは、User-Agent に文字列を設定することで、「自分が何なのか」ということを Web サーバに伝えます。クローラも例外ではありません。

試しに、主要なクローラの User-Agent を、尾内研の Web サーバのログから抜き出して、紹介しましょう。

googlebot:

```
Googlebot/2.1 (+http://www.googlebot.com/bot.html)
```

msnbot:

```
msnbot/0.11 (+http://search.msn.com/msnbot.htm)
```

yahoo! slurp:

```
Mozilla/5.0 (compatible; Yahoo! Slurp; http://help.yahoo.com/help/us/ysearch/slurp)
```

いずれのクローラも、このような情報を Web サーバに通知することで、自分が何者であるのかを明らかにしています。User-Agent の値は、クライアントが任意に設定するものです。あなたが作ったクローラにも、User-Agent として適切な文字列を設定してやる必要があります。

上に挙げた3つの User-Agent を見てください。似通ったフォーマットがあることに気づくでしょう。まず自分自身のプログラムの名称、次にスラッシュを入れて、そのバージョン番号。加えて自分自身に関する情報を参照できる URL。yahoo! slurp のものだけはちょっと毛色が変わっています。最初に何故か"Mozilla"とブラウザの名前が来て、肝心の"Yahoo! Slurp"は括弧のなかにあります。どうして検索エンジンのクローラが Mozilla を名乗るのかわけわめです。このフォーマットはむかし Netscape に居た Jamie Zawinski というひとが"user-agent string"というドキュメントのなかで提唱したものなのですが、なん

で yahoo がこのフォーマットに従って User-Agent を書いてるのはよくわかりません。mozilla.org の成果を何かしら使ってるんでしょうか。

で。あなたのクローラにも、このような User-Agent を設定してやりましょう。ヘンに奇をてらうことなく、これらのフォーマットに乗っ取って記述するのが良いでしょう。最低でも必要な情報は以下の通りです。

- ・クローラの名称
- ・そのクローラについて知るための URL

また、これらも付け加えておくとより親切です。

- ・バージョン番号
- ・制作者の連絡先 (メールアドレス)
- ・(既存のクローラを hack したなら、)元となったクローラの名称

### Web 上にページを作ろう

上に3つのクローラの User-Agent を挙げましたが、そこに書かれた URL にアクセスしてみてください。クローラ自身の説明と、robots.txt の書き方まで含めた丁寧な FAQ が掲載されています。このように、Web 上にクローラの利用者や、その目的などを説明したページを解説し、その URL を User-Agent に含めるといいことがよく為されます。できれば、このようなサイトを作成し、その場所を明らかにしておくことが望ましいです。作成するページには次のような情報を含めると良いでしょう。

- ・クローラの User-Agent
- ・クローラの目的
- ・クローラの作成者への連絡先
- ・クローラをブロックする方法

### クローラを登録しよう

Web 上には、クローラの情報を集めて公開しているサイトがあります。いくつかありますが、もっとも大きなものは『Web Robots Database(<http://www.robotstxt.org/wc/active.html>)』です。素性の知れないクローラについて調べるには、まずここを当たるひとが多いです。できれば、オンラインデータベースという客観的な情報源に身元を登録しておきましょう。

ここにクローラを登録するには、テンプレートに記入し、登録用のアドレスにメールで送信します。詳細な手順は上の URL に書いてあります。

### なぜこんなことをするの？

んで。この節で扱った内容を振り返ってみましょう。名前を付けるのは誰もが当然やることだからまあいいとして、残りの作業はなぜ必要なのでしょう。User-Agent を設定するくらいならまだしも、Web 上にページを作ったり、いちいちデータベースに登録したりという作業は非常に手間で、面倒でうざったくて鬱陶しいものに思えます。

理由を端的に言うと、クローラされる側の、つまり Web サーバの管理者やコンテンツの所有者の不安を解くためです。正体の知れないクローラがアクセスすることに対して、彼らの多くは不審に感じます。なぜこのクローラ（とおぼしきもの）は自分の Web サーバにアクセスしてくるのだろう。取得したりソースはなにに使われるのだろう。SPAM 業者がメールアドレスを蒐集するためにクローラを使っていることを前に述べましたが、身元が明らかでないクローラは、たとえその目的が正当なものであったとしても、そういった不埒なクローラと同一に扱われても仕方ありません。というか実際に同一に扱われます。よって、クローラの作成者には自分の身元とその目的を明らかにすることが求められます。言い換えれば、クローラの作成者は、Web サーバの管理者やコンテンツの所有者に対して、一定の説明責任を負っていると考えて良いと思います。

「どうせわからないだろう」、「自分のクローラにいちいち注目するような奴がいるはずがない」。このような考え方はまあそれはそれで意味正しいんですが、あと IE を偽装しちゃうとかすることもできなくはないんですが、ばれたら非常に格好悪いしまずいことになるので、控えておきましょう。

また、自分の Web サーバにアクセスしてくるクローラに注目するひとが、まったくいないというわけではありません。『WebmasterWorld(<http://www.webmasterworld.com/>)』と

いうサイトには、スパイダの識別と議論に特化したフォーラムが設置されています。  
『Search Engine IP Addresses(<http://www.iplist.com/>)』というサイトでは、主要なサーチエンジンのクローラの IP アドレスを蒐集し、公開しています。ですから、もしあなたが Googlebot を偽装したクローラを走らせていたら、そのうちばれて非難を浴びることになります。

まああれです。要するに、クローラの制作者は誠実であるべきなのです。

### クローラについてより深く学ぶには

この勉強会の後に、クローラについてさらに深く学びたいという人には、以下のような書籍があります。

#### 『Internet Agents: Spiders, Wanderers, Brokers, and 'Bots』

Fah-Chun Cheong, New Riders, ¥3,126 (税込)

HTTP プロトコルや HTML の知識までを含めて、実践的なクローラの作り方を解説した本です。

サンプルとして、WebWalker というクローラ、WebShopper という個人用途なショッピングサイト比較プログラム (詳細がどんなものかは不明) のコードが付属してます。

和訳は出てないっぽいですが、日本の amazon で手に入ります。

#### 『Bots & Other Internet Beasties』

Joseph Williams, Sam's, ¥4,883 (税込)

クローラの作りかたを解説した書籍。

上のものに比べると評価は低いようで、買うとしたら上のものにしたほうが良いと思われます。

和訳は出てないです。日本の amazon で手に入ります。

#### 『SPIDERING HACKS』

村上雅章, オライリー・ジャパン, ¥3,500

Web から特定のリソースをゲットするための、ちょっとした小技を100個集めた本です。amazon や yahoo!の各種サービスなどの特定サイトを対象に、所望のリソースをお手軽にゲットすることを目的としていて、ここで言うクローラを作るというようなものではありませんが、内容としては面白いです。

ソースコードの多くは perl を使って、CPAN で公開されているライブラリを使って簡潔に実装されています。

研究室のぼくの本棚にありますので、読みたい人は勝手に借りてってください。

### 『Web マイニング』

George Chang 他著, 武田善行他訳, ¥2,800

題名の通り、Web マイニング(データマイニングの手法を Web に対して適用する研究領域)についてそのあらましを説明した本です。一章を割いて、クローラについて説明がされています。参考文献が豊富で、サーベイとして質が高いです。

他に、クローラについて扱った学术论文も数多くあります。CiteSeer などを探してみてください。

## 2.JSpider

- ・ JSpider について

### 概要

クローラ勉強会では、JSpider というオープンソースなクローラのソースコードを読んでいます。

JSpider には以下のような特徴があります(このあいだのガイダンスでも述べましたが)。

- ・ JAVA
- ・ LGPL
- ・ 419kb, 251 クラス
- ・ 柔軟性が高い
- ・ ドキュメントがナイス
- ・ Windows でも動くっぽい

JSpider のオフィシャルサイトはここです。

<http://j-spider.sourceforge.net/>

また、JSpider に関するリソースは、以下の URL から取得可能です。

- ・ ソースコード

<http://prdownloads.sourceforge.net/j-spider/jspider-src-0.5.0-dev.zip>

- ・ ドキュメント

<http://prdownloads.sourceforge.net/j-spider/jspider-0-5-0-doc-user.pdf>

### 今回読むソースコードについて

今回の勉強会で読む JSpider のソースコードですが、JSpider のオフィシャルサイトで zip アーカイブとして提供されているもの (jspider-src-0.5.0-dev.zip) を使いたいと思います。JSpider の CVS には最新のソースコードがあるのですが、それではなく、アーカイブとして提供されているほうを使うということです。

また、jspider-src-0.5.0-dev.zip ですが、captain の下の場所にあります(もちろん、JSpider のオフィシャルサイトから直接落としてもかまいません)。

¥¥Captain¥public¥クローラ勉強会

本来ならこういうことはもっと早くに決定して周知しておくべきなのですが、仕事が遅れがちで今になってしまっておめんなさい。反省してます。

### JSpider をビルドする

ドキュメントで説明するのがめんどいので、実際にやってみましょう。

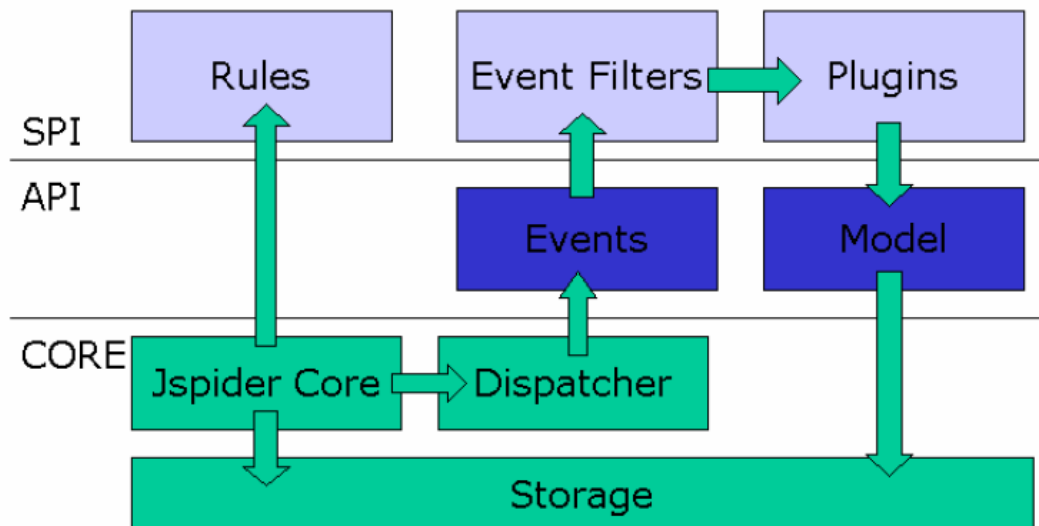
あと、Eclipse の使い方をぜんぜん分かってないので、変な操作をしているところがあったら指摘してください。誇張とかじゃなくてほんとに分かってないです。

#### ・ JSpider の構造

##### 概要図

JSpider の概要図を示します。ドキュメントの p14 から引用してきたものです。図中の矢印はデータの流れを表しています。

## JSpider overview



JSpider は全部で3つのレイヤに分かれています。

- ・ CORE

JSpider のもっとも下層に位置するレイヤです。Web サーバからのリソースの取得、HTML のパースなど、低レベルな処理全般を担当します。ディスクアクセスやスレッド周りの処理もここに含まれます。

- ・ API(Application Programmable Interface)

JSpider のオブジェクトモデルがここに属します。クローリングにおける色んな実体を抽象化したクラス群 (Site クラス、Cookie クラス、Resource クラス、etc...) のことです。また、イベント機構や micro-task (後述) 周りの処理もここに含まれます。

- ・ SPI(Service Provider Interface)

もっとも上のレイヤです。クローラがどんな動作をするかを規定する部分で、ユーザが指定することができます。

おおざっぱな処理の流れを書き出してみましょ。詳細については後述しますので、ここでは全体の流れをおおまかに掴んでください。

まず、"JSpider Core"が Web サーバからリソースをダウンロードします.....が、その前に、ダウンロードしようとしている URL を"Rules"に照会します。Rules は Rule クラスの

インスタンスの集まりであり、URL やリソースの種類によるフィルタリングの指定を行うことができます。また、robots.txt によるクローラへの指示も、内部的には Rule のひとつとして表現されます。

Rules によって許可されると、JSpider Core は Web サーバから実際にリソースをダウンロードします。また、リソースの種類が HTML であれば、パースとリンクの抽出が行われます。この際、"Dispatcher"によって Event が発行されます。発行された Event は"Events"を通して"Event Filters"に送られます。ここで Events とは、単に Event を集約するオブジェクトととらえて良いと思います。

Event Filters では、発生した Event を"Plugins"に通知するかどうかの判断が行われます。ここを通過した Event は、対応する Plugin に渡されます。Event を渡された Plugin は、その内容に応じた処理を行います。ここで、JSpider のオブジェクトモデルを通じて、ストレージ（ハードディスク）への保存や、ロギングが行われます。

## Rule の仕組み

JSpider がダウンロード、あるいは処理すべきリソースを決定します。Rule は全体に対して、あるいは個々のサイトに対して設定されます。ユーザは JSpider の振る舞いを Rule によって指定することができます。

Core が Rules に対して問い合わせをすると、Rules は問い合わせられた URL について判断し、以下のいずれかの指示を返します。

- don't care  
リソースを完全に無視する。
- accept  
状態のチェックをし、リソースをダウンロードし、HTML ならパースしてリンクを抽出する。他によって上書きされる。
- ignore  
状態のチェックのみをし、ダウンロードやパースは行わない。
- forbidden  
状態のチェックのみをし、ダウンロードやパースは行わない。ignore の強いもの。

Rule は複数個設定できます。また、その場合、URL はすべての Rule に対してチェックされます。Rule の返す指示には強弱の関係があり、選択された指示のなかでもっとも強い

ものを返します。

### Event の仕組み

スパイダリングの過程で、しかるべきイベントが発生すると関連づけられたプラグインが呼ばれる。

また、イベントタイプには3種類あります。

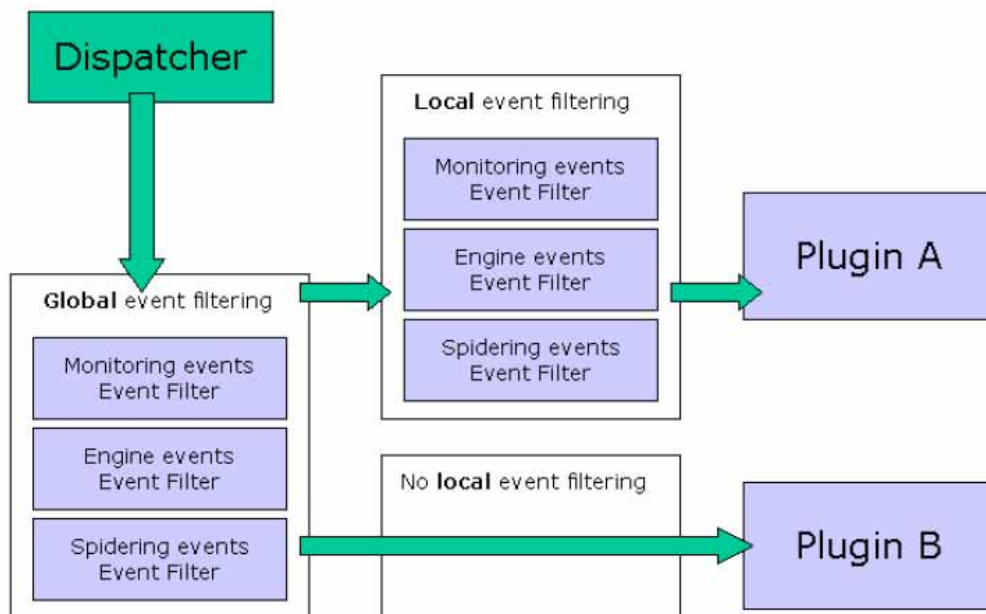
- engine event  
クローリングの開始、クローリングの中止、etc
- spidering event  
サイト発見、リソースをダウンロードした、ダウンロードに失敗した、etc
- monitoring event  
処理状況に関する情報の提示（進行状況や各スレッドの状態など）

具体的には、イベントはこのようなときに発行されます。

- 新しいサイトが見つかった
- robots.txt を解釈した
- リソースをダウンロードした
- リソースが見つからなかった（404 エラー）
- Rule のためにスキップした

発行されたイベントは、Dispatcher によって Event Filters に送られます。Event は対応する Filter によってチェックされ、それを通過したものは関連づけられた Plugin に渡されます。この流れを説明した図を示します（p21）。

## Event dispatching



Dispatcher によって発行されたイベントは、まず Global Event Filters に送られます。Global Event Filter はプログラム全体で一つだけ作成され、発行されるすべての Event はまずここに送られます。Event に3つの種類があることを前述しましたが、Event は対応する種類の Filter でチェックされます。Global Event Filter を通過した Event は、次にその Event に関連づけられた Plugin に対応する Local Event Filter に送られます。また、Local Event Filter はすべての Plugin に対して作成されるわけではありません。対応する Local Event Filter が存在しなければ、Event はそのまま Plugin に渡されます。

## Plugin の仕組み

Plugin は Event に関連づけられます。関連づけられた Event が発行され、それが Filter を通過して Plugin に渡されると、Plugin は各々に応じた処理を行います。次のような Plugin があります。

- ・ レポートを作成する
- ・ コンソールにメッセージを出力する

- ・ リソースをストレージに書き出す
- ・ メールを送信する
- ・ etc...

もちろん、Plugin を自分で作ることも出来ます。ドキュメントの中では、Plugin を上手く使ったシステムとして、次のようなものが挙げられています。

- ・ デッドリンクチェッカ  
指定された Web サイトをクローリングし、404 エラーが見つかるたびに Webmaster に報告のメールを送る
- ・ Web サイトバックアッププログラム  
指定された Web サイトの丸ごとコピーをハードディスクに作成する

#### micro-tasks

JSpider の設計は、"micro-tasks" という設計思想に基づいています。JSpider によって実行される処理は、すべて小さな処理の単位 "task" として生成されます。  
(具体的には、すべての処理が Task という interface を継承したクラスによって表現されます。詳細は次回以降で扱うつもりです)

task の例としては、次のようなものがあります。

- ・ リソースをダウンロードする task
- ・ HTML をパースする task
- ・ あるリソースがダウンロードされるべきかを Rule に基づいて決定する task
- ・ robots.txt を解釈する task
- ・ etc...

task は "Thinker task" と "spider task" の 2 種類に分けられます。

- ・ Spider task:  
ネットワーク(インターネット)経由で外に出て行き、リソースをダウンロードする task
- ・ Thinker task  
ローカル PC 内で完結する task。HTML をパースする、あるリソースをダウンロードするかを決定する、etc...

これら2種類の task は、それぞれ異なった thread pools によって実行されます。

### JSpider の2つの使い方

ドキュメント中では、JSpider の使い方には、"JSpider application"と"JSpider-tool"の2つがあると述べられています。

"JSpider application"とは、JSpider をそれ自体で完成したアプリケーションとして使用するというもの。"JSpider-tool"とは、自分の作りたいものを作るためのツールとして使うというものです。

JSpider の Core はアプリケーション本体からに対して独立性の高い設計になっており、それ自体を切り離して使えるような仕様になっているとのこと。

- ・ 今回のネタ元

- ・ The Web Robots FAQ...

<http://www.robotstxt.org/wc/faq.html>

robotstxt.org 内のコンテンツ。初学者に向けて書かれたナイスなドキュメント。

クローラとは何か、という話から始まって、基礎的な知識を一通り得られるようになっています。

- ・ SPIDERING HACKS

村上雅章, オライリー・ジャパン, ¥3,500

- ・ user-agent string

<http://www.mozilla.org/build/revised-user-agent-strings.html>

Jamie Zawinski 氏が User-Agent のフォーマットを提案したドキュメント。

興味のある人はどうぞ。

- ・ Web サーバに robots.txt を設置する

[http://shattered04.myftp.org/pc\\_32.html](http://shattered04.myftp.org/pc_32.html)

robots.txt の書き方を解説したページ。

・ [クローラー]ロボット対策スレ[robots.txt]

<http://pc5.2ch.net/test/read.cgi/mysv/1047386459/>

不埒なクローラの情報を交換したりしている 2ch のスレッド。

・ The LARM Web Crawler - Technical Overview

<http://jakarta.apache.org/lucene/docs/lucene-sandbox/larm/overview.html>

apache.org の作ってる LARM というクローラの解説ページ。クローラ全般に関する技術的な話題が載っている。