

## 本日のお品書き

- Event

- 前々回のおさらい
- Event の interface
- Event の実装
- Event/Plugin 間のやりとり ( Visitor Pattern ) について
- Event Filter の interface
- Event Filter の実装
- Internal Event Flow
- Event Filter の生成
- EventDispatcherImpl
- PluginSocket

- hack #1 - MyBlogSearch(仮)

- デモ
- 実装の説明
- これからの計画

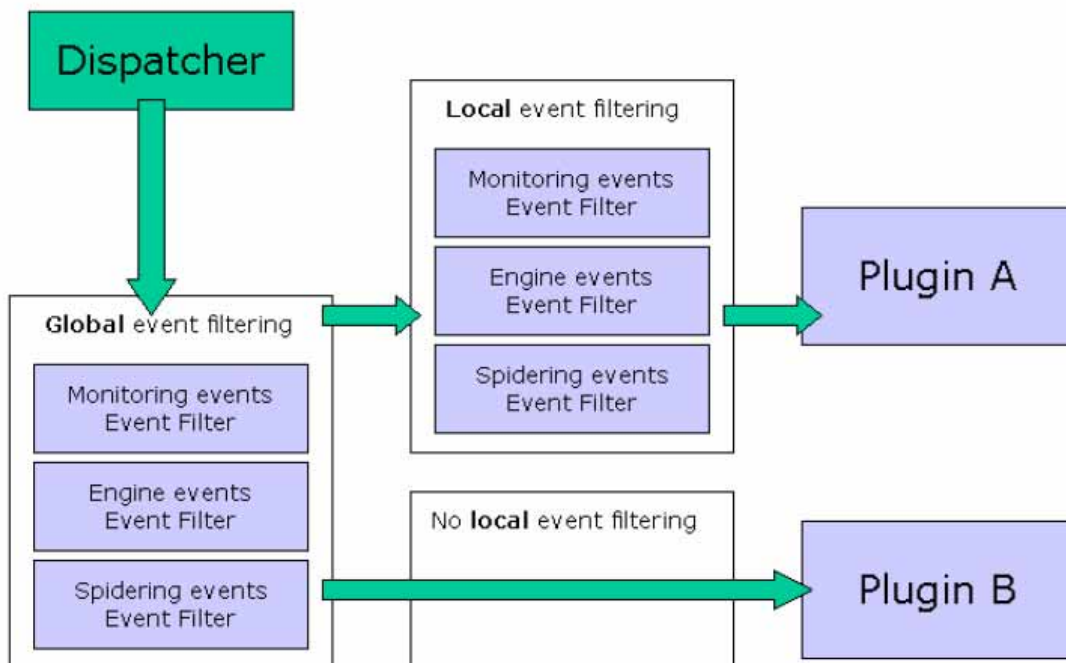
・ Event

前々回のおさらい

JSpider では、ダウンロードしたデータをストレージに書き出したり、コンソールにログを出したりという処理を、Plugin という仕組みによって実装していました。また、イベント・ドリブンなモデルを採用し、各々の Plugin の処理は、プログラム中のさまざまな箇所から発行される Event をきっかけとして実行されるようになっていました。

発行された Event は Event Filter というクラスによってフィルタリングされ、適切な Event だけが各 Plugin に渡されます。Event Filter にはプログラム全体でたった一つだけ作成される Global なものと、Plugin ごとに作成される Local なものがあります。また、Event には"engine event", "spidering event", "monitoring event"の3つの分類があり、この分類ごとに Event Filter を設定するようになっていました。

Event dispatching



## Event の interface

先ず、EventVisitable という interface を見てみます。所在は¥net¥javacoding¥jspider¥api¥event¥EventVisitable.java です。

```
/**
 * Visitable part of the Visitor pattern implementation that's used for the
 * handling of JSpider API events. By implementing this interface, the Event
 * becomes visitable for an EventVisitor
 *
 * $Id: EventVisitable.java,v 1.1.1.1 2002/11/20 17:02:31 vanrogu Exp $
 *
 * @author G?nther Van Roey
 */
public interface EventVisitable {

    /**
     * Notifies the visitable object that a visitor wants to visit it.
     * @param visitor the visitor object that wants to visit this visitable
     */
    public void accept(EventVisitor visitor);
}
```

この EventVisitable interface が、Event のもっとも基底となるものです。visitor を引数にする accept 関数のみが宣言されています。ここで使われている「Visitor パターン」については後で扱います。

また、これを継承したものとして、JSpiderEvent クラスがあります。所在は¥net¥java¥coding¥jspider¥api¥event¥JSpiderEvent.java です。少々長めのクラスですが、引用します。

```
/**
 * Base class of all JSpider API events. All events that will be dispatched
 * towards JSpider modules will extends directly or indirectly from this
 * class.
 *
 * $Id: JSpiderEvent.java,v 1.5 2003/03/23 15:44:48 vanrogu Exp $
 *
 * @author G?nther Van Roey
 */
public abstract class JSpiderEvent implements EventVisitable {

    public static final String EVENT_PACKAGE = "net.javacoding.jspider.api.event.";

    /** Event type used for events related to the JSpider engine. */
    public static final int EVENT_TYPE_ENGINE = 1;

    /** Event type used for events related to monitoring. */
    public static final int EVENT_TYPE_MONITORING = 2;

    /** Event type used for real spider events. */
}
```

```
public static final int EVENT_TYPE_SPIDER = 3;

/** Timestamp of the event. */
protected Date date;

/**
 * Public constructor.
 */
public JSpiderEvent() {
    date = new Date();
}

/**
 * Returns the name of the event
 * @return name of the event
 */
public String getName() {
    return getClass().getName().substring(EVENT_PACKAGE.length());
}

/**
 * Returns a Date object containing the event's timestamp
 * @return Date object containing the event's timestamp
 */
public Date getRaisedDate() {
    return date;
}

/**
 * Returns optional comments on the event.
 * @return comment on the event.
 */
public abstract String getComment();

/**
 * Returns whether this event describes some sort of error situation.
 * @return boolean that is true if this event describes an error
 */
public boolean isError ( ) {
    return false;
}

/**
 * Returns whether this event can be filtered out. Only the most critical
 * events will return FALSE.
 * @return whether this event can be filtered out
 */
public boolean isFilterable ( ) {
    return true;
}

/**
 * Returns the type of the event.
 * @return the type of the Event - Engine / Monitoring / Spider
 */
public int getType ( ) {
    // By default, it will be a normal spider event.
    return JSpiderEvent.EVENT_TYPE_SPIDER;
}

/**
 * Accept method for the Visitor-pattern that's used for handling events.
 * @param visitor the visitor instance that wants to visit the event
 */
```

```
public void accept(EventVisitor visitor) {
    visitor.visit(this);
}

/**
 * Object's overridden toString() method.
 * @return String representation of the event
 */
public String toString() {
    return this.getClass().getName();
}
}
```

基本的には、Event の種別やコメント、発行された時刻など、Event に関連する情報を提供するためのインターフェースを宣言したものとなっています。また、抽象クラスですので、これ自体のインスタンスは出来ないということに注意してください。

isFilterable という属性があります。これに true を設定しておくと、その Event はフィルタリングすることができないものとされます。具体的には、isFilterable が true となっている Event は、Event Filter による判断を回避されます。

また、`net.javacoding.jspider.core.event.CoreEvent.java` に CoreEvent という interface がありますが、これは Core の中でだけ使われる Event であり、今回扱う内容には含まれません。

### Event の実装

JSpiderEvent クラスを継承して、各々の Event が実装されます。所在は以下です。

```
net.javacoding.jspider.api.event.engine*.java
net.javacoding.jspider.api.event.folder*.java
net.javacoding.jspider.api.event.monitor*.java
net.javacoding.jspider.api.event.resource*.java
net.javacoding.jspider.api.event.site*.java
```

ここで重要なこととして、Event は、単にその Event の種類を表す「タグ」としての機能のみを持つものではありません。各々の Event には、その Event に関連づけられた情報を取得するためのインターフェースが実装されます。その例として、ここでは EMailAddressDiscoveredEvent クラスを取り上げます。HTML をパースして、メールアドレスが見つ

かったときに発行される Event です。所在は¥net¥javacoding¥jspider¥api¥event¥resource¥EmailAddressDiscoveredEvent.java です。

```
/**
 * $Id: EMailAddressDiscoveredEvent.java,v 1.1 2003/04/08 15:50:25 vanrogu Exp $
 */
public class EMailAddressDiscoveredEvent extends ResourceRelatedEvent {

    protected String email;

    public EMailAddressDiscoveredEvent ( FetchedResource resource, String email ) {
        super ( resource );
        this.email = email;
    }

    public FetchedResource getResource ( ) {
        return (FetchedResource)resource;
    }

    public String getEmailAddress ( ) {
        return email;
    }

    public String getComment() {
        return "e-mail address '" + email + "' discovered";
    }

    public void accept(EventVisitor visitor) {
        visitor.visit(this);
    }

    public String toString ( ) {
        return getComment();
    }
}
```

このクラスでは、getEmailAddress で見つけたメールアドレスを文字列型で返すようになっています。また、FetchedResource という、JSpider の内部的なオブジェクトを取得することもできます。他の Event クラスでも、それぞれが情報を取得するための独自のインターフェースを持っています。

#### Event/Plugin 間のやりとり(Visitor Pattern)について

Plugin は Visitor パターンを使って実装することも可能です。また、Plugin の中には、実際にそうされているものもあります。

Plugin が Visitor、Event が Acceptor として振る舞います。Visitor としての interface は EventVisitor(¥net¥javacoding¥jspider¥api¥event¥EventVisitor.java)です。

```
/**
 * Visitor interface that must be implemented upon each class that will act as
 * a visitor in the applied visitor pattern for event handling.
 * This interface contains a visit method for each JSpider API event type.
 *
 * $Id: EventVisitor.java,v 1.6 2003/04/08 15:50:24 vanrogu Exp $
 *
 * @author G?nther Van Roey
 */
public interface EventVisitor {

    /**
     * Visit method.
     * @param event that occurred
     */
    public void visit(JSpiderEvent event);

    /**
     * Visit method.
     * @param event that occurred
     */
    public void visit(EngineRelatedEvent event);

    /**
     * Visit method.
     * @param event that occurred
     */
    public void visit(SpideringStartedEvent event);

    /**
     * Visit method.
     * @param event that occurred
     */
    public void visit(SpideringStoppedEvent event);

    (中略)

    /**
     * Visit method.
     * @param event that occurred
     */
    public void visit(UserAgentObeyedEvent event);
}
```

EventVisitor interface を実装したものとして、FlatOutputPlugin(¥net¥javacoding¥j spider¥mod¥plugin¥FlatOutputPlugin.java)があります。

```
/**
 * $Id: FlatOutputPlugin.java,v 1.12 2003/04/08 15:50:38 vanrogu Exp $
 */
public abstract class FlatOutputPlugin implements Plugin, EventVisitor {

    public final void initialize() {
        setUp ( );
    }

    public void shutdown() {
        tearDown ( );
    }
}
```

```
public void visit(JSpiderEvent event) {
    println (event);
}

public void visit(EngineRelatedEvent event) {
    println (event);
}

public void visit(SpideringStartedEvent event) {
    println("Module : " + getName ( ) );
    println("Version: " + getVersion( ) );
    println("Vendor : " + getVendor ( ) );
    println("Spidering Started, baseUrl = " + event.getBaseUrl());
}

public void visit(SpideringStoppedEvent event) {
    println ("Spidering Stopped");
}

( 中略 )

public void visit(UserAgentObeyedEvent event) {
    println ( event.getComment());
}

public void notify(JSpiderEvent event) {
    event.accept(this);
}

protected abstract void println ( Object object );

protected void setUp ( ) {
}

protected void tearDown ( ) {
}
}
```

この JSpider のケースでは、Event の実装ごとにデータ構造が異なっており、FlatOutputPlugin のような多くの Event を取り扱う Plugin を記述しようとしたときには、Event の実装ごとに異なるコードを書く必要があります。前回紹介した DiskWriterPlugin クラス (`¥net¥javacoding¥jspider¥mod¥plugin¥diskwriter¥DiskWriterPlugin.java`) のような、ある特定の Event を相手にすれば良い Plugin の場合は直接それに対応したコードを notify 関数の中に書いてしまっても良いのですが、そうでないなら、Visitor パターンを使うことが望ましいとおもいます。

また、FlatOutputPlugin は抽象クラスです。これをさらに継承したものとして、ConsolePlugin クラス (`¥net¥javacoding¥jspider¥mod¥plugin¥console¥ConsolePlugin.java`) が実装されています。Plugin までを含めた UML は以下のようになります。



## Event Filter の interface

Event Filter の interface です。所在は`net.javacoding.jspider.spi.EventFilter.java`です。

```
/**
 * Interface that will be implemented upon a component in a JSpider module that
 * allows to filter JSpider event coming from the engine.
 *
 * <p>
 * This can be handy, for instance, if you're writing a module that checks all
 * error conditions on a server. In that case, you would not be interested in
 * the HTTP statuses 2xx, but you would definitely want to receive
 * notifications about 4xx and 5xx states.
 * </p>
 *
 * $Id: EventFilter.java,v 1.2 2003/04/25 21:29:06 vanrogu Exp $
 *
 * @author Gunther Van Roey (gunther@javacoding.net)
 * @version $Revision: 1.2 $ $Date: 2003/04/25 21:29:06 $ $Name: $
 */
public interface EventFilter {

    /**
     * Method that will filter an events and tells the dispatcher whether we're
     * interested in it or not. Returns true if the event must be dispatched,
     * false if it can be ignored.
     * @param event - the JSpider event to be filtered
     * @return boolean value telling whether to dispatch the event or not
     */
    public boolean filterEvent(JSpiderEvent event);
}
```

JSpiderEvent クラスのインスタンスを受け取って、判断の結果を boolean で返すようになっています。false ならその event はフィルタされます。

## Event Filter の実装

EventFilter interface を継承した実装は以下にあります。

`net.javacoding.jspider.mod.eventfilter.*.java`

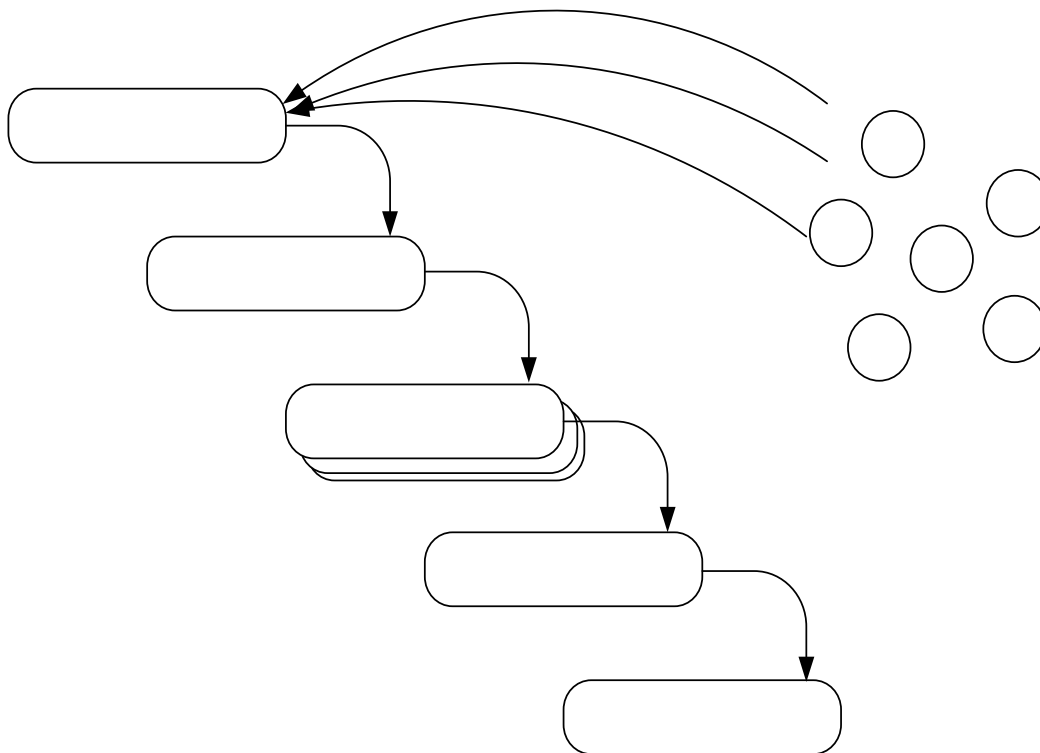
また、一例として ErrorsOnlyEventFilter クラス(`net.javacoding.jspider.mod.eventfilter.ErrorsOnlyEventFilter.java`)を紹介します。

```
/**
 * Event filter implementation that filters out all events that are not error
 * notifications. This way it is possible to get an overview of all errors that
 * are encountered during the spidering process.
 *
 * $Id: ErrorsOnlyEventFilter.java,v 1.3 2003/04/03 16:25:12 vanrogu Exp $
 *
 * @author G?nther Van Roey
 */
public class ErrorsOnlyEventFilter implements EventFilter {

    /**
     * Method that filters the events. Returns true for error events, false for
     * other events.
     * @param event the event to be evaluated.
     * @return true if the event is to be let through the filter
     */
    public boolean filterEvent(JSpiderEvent event) {
        return event.isError ( );
    }
}
```

### Internal Event Flow

発行された Event は、下図のような Flow を経て、各 Plugin に到達します。ここでは、各クラスの実装の詳細はとりあえず置いといて、Event がどのように流れていくかを把握しておいてください。



また、図に示された各クラスについて、簡単な説明を付けます。

- SpiderContextImpl(¥net¥javacoding¥jspider¥core¥impl¥SpiderContextImpl.java)

クローリングに必要なオブジェクトを集約し、また、Spider自身のステータスを管理するクラス。プログラム全体で一つだけインスタンスがされます。レイヤで言うと Core に属し、プログラムの中で中心的な役割を担います。

## SpiderContextImpl

- EventDispatcherImpl(¥net¥javacoding¥jspider¥core¥dispatch¥impl¥EventDispatcherImpl.java)

Event の伝達を担当するクラス。また、EventFilter のインスタンスを管理するのもこのクラスで行われます。後に詳述します。

## EventDispat

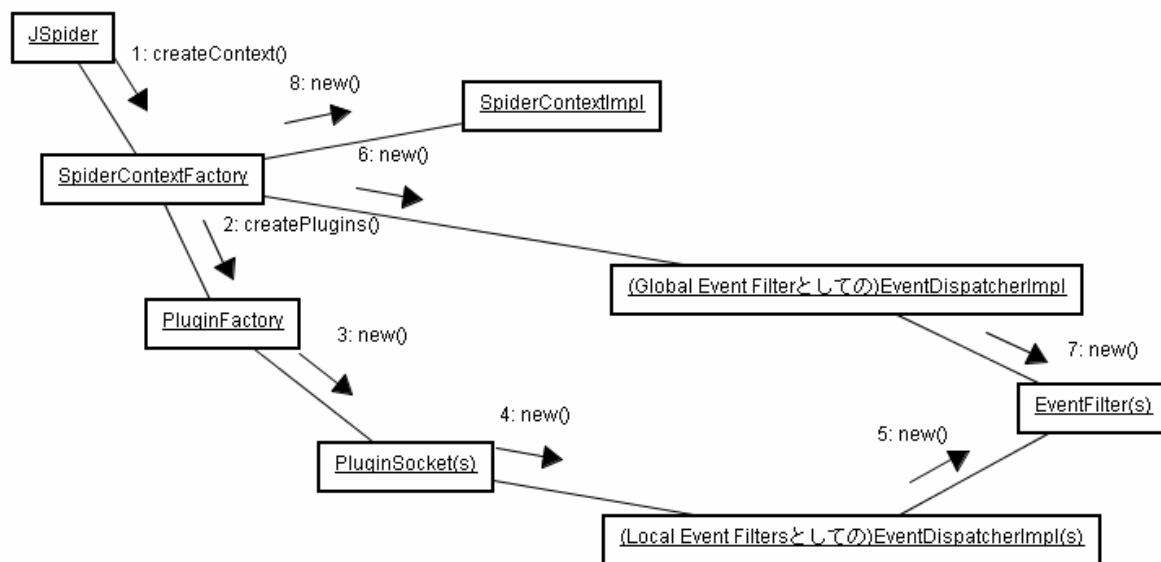
- PluginSocket(¥net¥javacoding¥jspider¥core¥dispatch¥impl¥PluginSocket.java)

Plugin と Event Filter の対応付けを管理するクラス。これも後に詳述します。

- a Plugin Instance

それぞれの Plugin のインスタンスです。

### Event Filter の生成



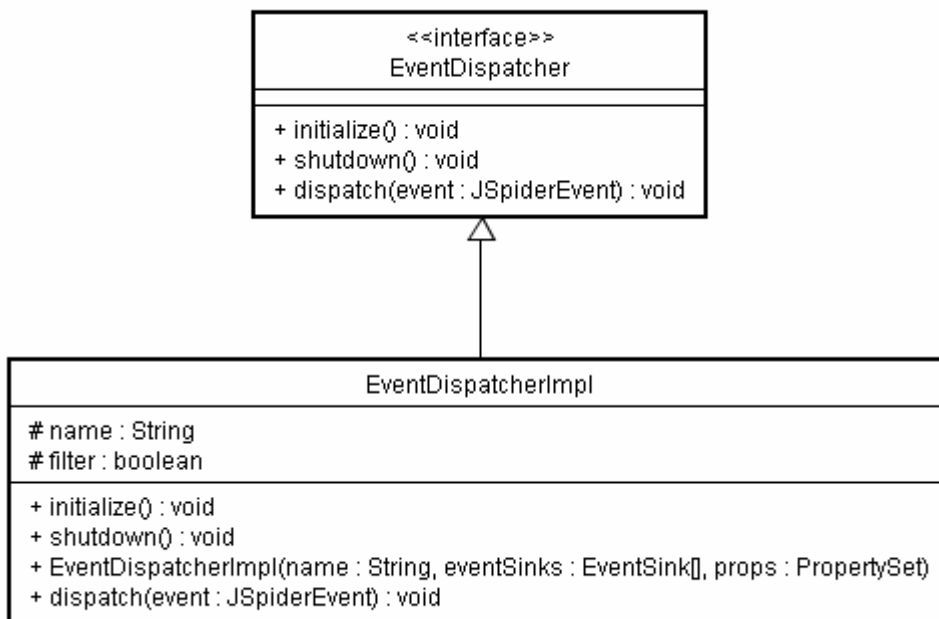
## EventDispatcherImpl

EventDispatcherImpl クラス(¥net¥javacoding¥jspider¥core¥dispatch¥impl¥EventDispatcherImpl.java)は、Event をフィルタリングすべきかを判断し、チェックを通過した Event をさらに次のオブジェクトに伝える役割を担います。属性として、

- ・ Event の分類に対応した 3 つの Event Filter
- ・ 通過した Event を伝える EventSink の配列

を持ちます。また、dispatch 関数で Event を処理します。

EventDispatcherImpl クラスの UML とソースコードを下に示します。



```
/**
 *
 * $Id: EventDispatcherImpl.java,v 1.11 2003/04/22 16:43:33 vanrogu Exp $
 *
 * @author G?nther Van Roey
 */
public class EventDispatcherImpl implements EventDispatcher {

    protected String name;
    protected boolean filter;

    protected EventSink[] eventSinks;
    protected EventFilter engineEventFilter;
    protected EventFilter monitorEventFilter;
    protected EventFilter spiderEventFilter;
```

```
protected Log log;

public void initialize() {
    log.debug(name + " intializing...");
    for (int i = 0; i < eventSinks.length; i++) {
        EventSink eventSink = eventSinks[i];
        eventSink.initialize();
    }
    log.debug(name + " intialized.");
}

public void shutdown() {
    log.debug(name + " shutting down.");
    for (int i = 0; i < eventSinks.length; i++) {
        EventSink eventSink = eventSinks[i];
        eventSink.shutdown();
    }
    log.debug(name + " shutdown.");
}

public EventDispatcherImpl(String name, EventSink[] eventSinks, PropertySet props) {
    log = LogFactory.getLog(EventDispatcher.class);
    log.debug(name + " configuring...");
    this.name = name;
    this.filter = props.getBoolean(ConfigConstants.FILTER_ENABLED, true);
    this.eventSinks = eventSinks;

    if (filter) {
        Class engineEventFilterClass = props.getClass(ConfigConstants.FILTER_ENGINE, AllowAllEventFilter.class);
        Class monitorEventFilterClass = props.getClass(ConfigConstants.FILTER_MONITORING, AllowAllEventFilter.class);
        Class spiderEventFilterClass = props.getClass(ConfigConstants.FILTER_SPIDER, AllowAllEventFilter.class);
        log.debug("EventFilter for engine events = " + engineEventFilterClass.getName());
        log.debug("EventFilter for monitor events = " + monitorEventFilterClass.getName());
        log.debug("EventFilter for spider events = " + spiderEventFilterClass.getName());
        try {
            engineEventFilter = (EventFilter) engineEventFilterClass.newInstance();
            monitorEventFilter = (EventFilter) monitorEventFilterClass.newInstance();
            spiderEventFilter = (EventFilter) spiderEventFilterClass.newInstance();
        } catch (InstantiationException e) {
            log.error("InstantiationException on EventFilter", e);
        } catch (IllegalAccessException e) {
            log.error("IllegalAccessException on instantiation of EventFilter", e);
        }
    } else {
        log.info("Global event filtering is DISABLED");
    }
    log.debug("EventDispatcher " + name + " configured.");
}

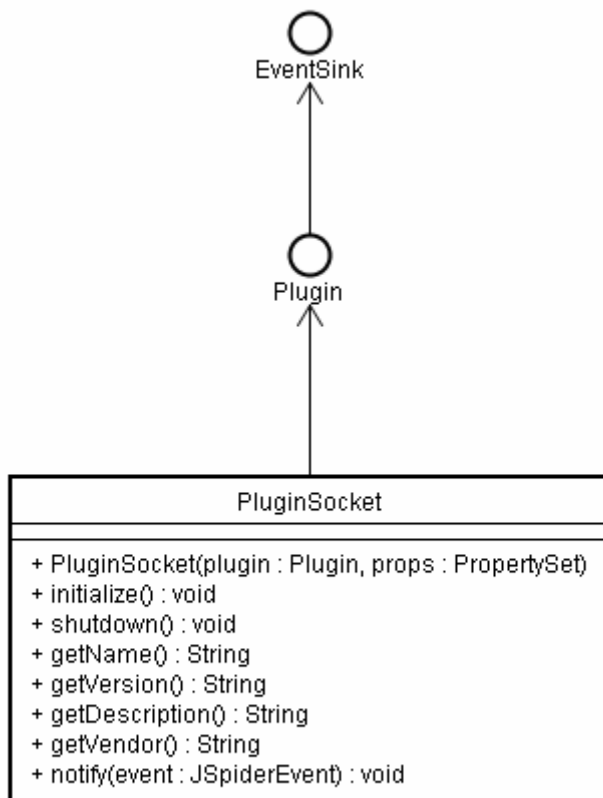
public void dispatch(JSpiderEvent event) {
    boolean mustDispatch = false;
    if (filter) {
        EventFilter eventFilter = spiderEventFilter;
        if (event.isFilterable()) {
            switch (event.getType()) {
                case JSpiderEvent.EVENT_TYPE_ENGINE:
                    eventFilter = engineEventFilter;
            }
        }
    }
}
```

```
        break;
        case JSpiderEvent.EVENT_TYPE_MONITORING:
            eventFilter = monitorEventFilter;
            break;
        case JSpiderEvent.EVENT_TYPE_SPIDER:
            eventFilter = spiderEventFilter;
            break;
        default:
            eventFilter = spiderEventFilter;
    }
    if (eventFilter.filterEvent(event)) {
        mustDispatch = true;
    }
    } else {
        mustDispatch = true;
    }
    } else {
        mustDispatch = true;
    }
    }
    if (mustDispatch) {
        for (int i = 0; i < eventSinks.length; i++) {
            EventSink sink = eventSinks[i];
            sink.notify(event);
        }
    }
}
```

## PluginSocket

PluginSocket(`net.javacoding.jspider.core.dispatch.impl.PluginSocket.java`)は、対応する Plugin と EventDispatcher を結びつけて管理するためのクラスです。プログラムの起動時に SocketFactory クラスによってインスタンスングされます。

PluginSocket の UML とソースコードを下に示します。



```
/**
 * Proxy class that enables event filtering for a plugin. If the plugin is
 * configured for local event filtering, an instance of this class will wrap
 * the real plugin instance and filter the incoming dispatched events.
 *
 * $Id: PluginSocket.java,v 1.9 2003/04/03 16:24:51 vanrogu Exp $
 *
 * @author G?nther Van Roey
 */
public class PluginSocket implements Plugin {

    protected Plugin plugin;
    protected EventDispatcher dispatcher;

    public PluginSocket ( Plugin plugin, PropertySet props) {
        this.plugin = plugin;
        EventSink[] sinks = new EventSink[1];
        sinks[0] = plugin;
        dispatcher = new EventDispatcherImpl("EventDispatcher for Plugin '" + plugin.getName
() + "'", sinks, props);
    }

    public void initialize() {
        dispatcher.initialize ( );
    }
}
```

```
public void shutdown() {
    dispatcher.shutdown();
}

public String getName() {
    return plugin.getName ( );
}

public String getVersion() {
    return plugin.getVersion ( );
}

public String getDescription() {
    return plugin.getDescription ( );
}

public String getVendor() {
    return plugin.getVendor ( );
}

public synchronized void notify(JSpiderEvent event) {
    dispatcher.dispatch(event);
}
}
```

- ・ hack #1 - MyBlogSearch(仮)

## デモ

## 実装の説明

口頭で説明。

ソースコードを研究室 PC に持ってきてあるので、ご希望なら実際にソースコードをお見せすることもできます。

## これからの計画

- ・ スケジューラを作る。MyBlogSearch(仮に登録された blog を定期的にクローリングし、常に新鮮な情報を検索結果として提供できるようにする。
- ・ 日付を抽出する機能を実装する。現在、記事の日付として、作成日時ではなく蒐集した日時が使われてしまっており、これでは日付でソートする機能が意味を為さない。これをなんとかするために、蒐集した記事データを解析して日付を抽出し、記事の日時として使うようにする。
- ・ RSS から更新をチェックする機能を実装する。初回はサイト全体をクローリングするとしても、二回目以降は RSS をチェックして、新着記事だけを取りに行くようにしたい。毎回サイト全体をクローリングするのでは、実装として下さいし、クローリング先のサーバ & 回線に負荷を掛けてしまう。
- ・ ceekz 氏から教えてもらったやつを namazu に組み込む。
- ・ 公式サイトを作ってユーザ登録を受け付け、みなさんにサービスを提供する（ここまでやるのか？）
- ・ エトセトラ、エトセトラ