Oracle® WebCenter Sites

サイト・キャプチャ・アプリケーション管理者ガイド

11g リリース 1 (11.1.1)

部品番号 : B69688-01

2012 年 4 月

ORACLE®

Oracle® WebCenter Sites: サイト・キャプチャ・アプリケーション管理者ガイド , 11g リリース 1 (11.1.1)

部品番号 : B69688-01

原本名 : Oracle® WebCenter Sites: Administrator's Guide for the Site Capture Application, 11g Release 1 (11.1.1)

原本主著者 : Tatiana Kolubayev

原本協力者 : Amit Kumar、Kamal Kapur

目次

このガイドについて

このガイドでは、動的にパブリッシュされた Web サイトをダウンロードするために使用される Web アプリケーション、Oracle WebCenter Sites: サイト・キャプチャについて説明します。

このガイドでは、まず、サイト・キャプチャ・アプリケーションの概要を示します。サイト・キャプチャ・アプリケーションとそのファイル・システムをナビゲートし、クロール・セッションとダウンロードした Web サイトを管理する方法を説明します。最後の章では、クロールのサイト・キャプチャ・プロセスを制御するために使用される構成コードを示します。これらの例では、リンク抽出ロジックを実装し、URL をリライトして、クロール・セッションの終わりに電子メール通知を有効にし、それ以外の場合はキャプチャ・プロセスをカスタマイズするための Java メソッドとインタフェースの使用方法について示します。

このガイドで説明しているアプリケーションは、旧 FatWire の製品です。命名規則は次のとおりです。

- *Oracle WebCenter Sites* は、以前は *FatWire Content Server* と呼ばれていたアプリケーションの現在の名前です。このガイドでは、*Oracle WebCenter Sites* を *WebCenter Sites* と呼ぶこともあります。
- *Oracle WebCenter Sites: サイト・キャプチャ*は、以前は *FatWire Site Capture* と呼ばれていたアプリケーションの現在の名前です。このガイドでは、*Oracle WebCenter Sites: サイト・キャプチャ*を *サイト・キャプチャ*と呼ぶこともあります。
- *Oracle WebCenter Sites: Web Experience Management Framework* は、以前は *FatWire Web Experience Management Framework* と呼ばれていた環境の現在の名前です。このガイドでは、この環境を *Web Experience Management Framework* または *WEM Framework* と呼ぶこともあります。

サイト・キャプチャ・アプリケーションは、*Oracle WebCenter Sites 11g リリース 1 (11.1.1.x) の動作保証マトリックス*の仕様に準じて Oracle WebCenter Sites と統合されます。詳細は、WebCenter Sites のリリース・ノートを参照してください。最新の*動作保証マトリックス*およびリリース・ノートについては、WebCenter Sites のドキュメント・サイトを定期的に確認してください。

対象読者

サイト・キャプチャ・アプリケーションおよびこのガイドは、WebCenter Sites の全体管理者および開発者を対象としています。前述のユーザーのいずれも、サイト・キャプチャ・インタフェースをナビゲートすることができ、Web サイトをダウンロードするためにそのクローラを基本的なコードを使用して簡単に構成できます。最後の章で説明するように、高度な構成コードを記述する場合は、Java 開発者の専門知識が必要です。

このガイドで説明するようにサイト・キャプチャ・アプリケーションを使用するには、次の権限と経験が必要です。

- `GeneralAdmin` ロールと、`RestAdmin` セキュリティ・グループにも属する全体管理者のすべての資格証明を持つ WebCenter Sites ユーザーであること。
- サイト・キャプチャ・ホスト・マシンへの管理アクセス権限があること。静的にキャプチャされたサイトおよび関連情報には、サイト・キャプチャ・ファイル・システムを使用してアクセスします。
- 高度な構成コードを記述する場合は、Java の専門知識があること。

関連ドキュメント

詳細は、次のドキュメントを参照してください。

- *『Oracle WebCenter Sites サイト・キャプチャ・アプリケーション・インストレーション・ガイド』*
- *『Oracle WebCenter Sites 管理者ガイド』*

表記規則

このガイドでは、次の表記規則を使用します。

- **太字**は、ユーザーが選択するグラフィカル・ユーザー・インタフェース要素を示します。
- *斜体*は、ドキュメントのタイトル、強調、またはユーザーが特定の値を指定する変数を示します。
- 等幅フォントは、ファイル名、URL、サンプル・コード、または画面に表示されるテキストを示します。
- **等幅太字フォント**は、コマンドを示します。

このガイド内の手順

このガイドに記述している手順の一部は簡易手順として記述されており、様々な操作を実行する際のクイックリファレンスになります。たとえば、アーカイブ済サイトのクローラのリストを表示するには、次に示す簡易手順を参照します。

クローラ→ *crawlerName* **→アーカイブ**

前述の手順は次を意味します。

「クローラ」ホームページに移動してクローラをポイントし、ポップアップ・メニューから**「アーカイブ」**を選択します。

概念、機能および関連操作の説明が必要な場合は、その手順が詳細に記載されています。

サード・パーティのライブラリ

Oracle WebCenter Sites およびそのアプリケーションには、サード・パーティのライブラリが含まれています。詳細は、*Oracle WebCenter Sites 11gR1: サード・パーティのライセンス*を参照してください。

第 1 章

Oracle WebCenter Sites: サイト・キャプチャへようこそ

この章では、Oracle WebCenter Sites: サイト・キャプチャ・アプリケーションについて紹介し、そのインタフェースをナビゲートする方法について説明します。

この章は、次の項で構成されています。

サイト・キャプチャ・モデル

クロールは、サイト・キャプチャ・インタフェースから手動で開始できます。または、WebCenter Sites リアルタイム・パブリッシュ・セッションの完了によりトリガーできます。各シナリオで、クローラをどのように実行するように選択したかに応じて静的モードまたはアーカイブ・モードのいずれかで、クローラはディスクに Web サイトをダウンロードします。

キャプチャ・モード

サイトが静的モードでダウンロードされる場合も、アーカイブ・モードダウンロードされる場合も、同じファイル (html、css など) がディスクに格納されますが、いくつかの相違があります。たとえば、静的にダウンロードされたサイトは、ファイル・システムでのみ使用可能ですが、アーカイブ済サイトは、ファイル・システムとサイト・キャプチャ・インタフェースの両方で使用できます。キャプチャ・モードは、クローラがサイトをどのようにダウンロードし、結果をどのように管理するかを決定します。

静的モード	アーカイブ・モード
静的モードは、迅速なデプロイメント、高可用性シナリオをサポートしています。	アーカイブ・モードは、コンプライアンス目的または類似の理由で定期的に Web サイトのコピーを維持するために使用されます。
静的モードでは、クロールされたサイトは提供される準備ができたファイルとして格納されます。最新のキャプチャのみが保持されます (前に格納されたファイルは上書きされます)。	アーカイブ・モードでは、すべてのクロールされたサイトはタイムスタンプ付きフォルダに zip ファイル (アーカイブ) として保持され、格納されます。zip ファイルへのポインタは、サイト・キャプチャ・データベースで作成されます。
静的クロール・セッションは、アプリケーション・インタフェースから手動で開始することも、パブリッシュ・セッションの終わりに開始することも可能です。ただし、ダウンロードされたサイトはサイト・キャプチャ・ファイル・システムからのみアクセスおよび管理できます。	アーカイブ・クロール・セッションは、静的セッションと同様に、サイト・キャプチャ・インタフェースから手動で開始することも、パブリッシュ・セッションの終わりに開始することも可能です。ただし、zip ファイルは、サイト・キャプチャ・データベースのポインタで参照されるため、サイト・キャプチャ・インタフェースから管理できます。このインタフェースで、ファイルをダウンロードし、アーカイブ済サイトをプレビューして、キャプチャ・スケジュール設定できます。

いずれのキャプチャ・モードの場合でも、ログがクロール・セッションの終わりに生成され、クロール済 URL、HTTP ステータス、ネットワーク状態などの情報を提供します。静的キャプチャでは、ログはファイル・システムから取得される必要があります。アーカイブ・キャプチャでは、サイト・キャプチャ・インタフェースからダウンロードできます。いずれのキャプチャ・モードでも、ログが

生成されるとすぐにレポートを電子メールで送信するようにクローラを構成するオプションがあります。

クローラ

どのタイプのサイト・キャプチャ・プロセスを開始する場合でも、クローラをサイト・キャプチャ・インタフェースで定義する必要があります。すぐに始められるように、サイト・キャプチャには、「Sample」と「FirstSiteII」という 2 つのサンプル・クローラがあります。ここでは、サイト・キャプチャ・インストール・プロセス中にクローラがインストールされていることを前提としています。このガイドでは、主に「Sample」クローラを使用します。

独自のクローラの作成には、クローラに名前を付け (通常、ターゲット・サイトにちなんだ名前を付けます)、`CrawlerConfigurator.groovy` という名前のテキスト・ファイルをアップロードすることが含まれます。これにより、クローラのサイト・キャプチャ・プロセスが制御されます。`groovy` ファイルは、少なくともクローラの開始 URI とリンク抽出ロジックを指定する、`BaseConfigurator` クラスのメソッドを使用してコーディングされる必要があります。`groovy` ファイルはクローラのサイト・キャプチャ・プロセスを制御しますが、クローラのキャプチャ・モードはファイル外で設定されます。

パブリッシュ・トリガー・サイト・キャプチャ用のクローラを使用するには、追加の手順が必要です。*Oracle WebCenter Sites サイト・キャプチャ・アプリケーション・インストレーション・ガイド*で説明されるように、クローラに名前を付け、サイト・キャプチャと統合されている WebCenter Sites ソース・システム上のパブリッシュの宛先定義でキャプチャ・モードを指定します。(すべてのパブリッシュの宛先定義において、1 つまたは複数のクローラを指定できますが、単一のキャプチャ・モードしか指定できないことを念頭においてください。) クローラ起動の成功に関する情報は、サイト・キャプチャ・ファイル・システムと WebCenter Sites ソースおよびターゲット・システムのログ・ファイル (デフォルトで `futuretense.txt`) に格納されます。

この章の演習では、手動およびパブリッシュ・トリガーの両方のタイプのクローラ起動シナリオを取り上げます。

サイト・キャプチャ・アプリケーションへのログイン

サイト・キャプチャ・アプリケーションは、WebCenter Sites 上で実行されます。サイト・キャプチャ・アプリケーションには、WebCenter Sites にログインしてアクセスします。

サイト・キャプチャ・アプリケーションにログインするには :

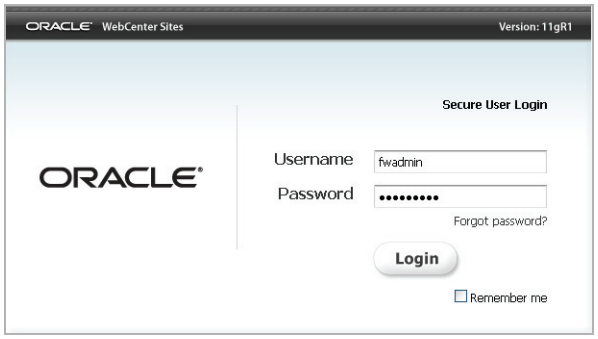
1. 次の URL で WebCenter Sites にアクセスします。

   `http://<server>:<port>/<context>/login`

   ここで `<server>` は WebCenter Sites を実行しているサーバーのホスト名または IP アドレス、`<port>` は WebCenter Sites アプリケーションの番号、`<context>` は、サーバー上にデプロイされた WebCenter Sites Web アプリケーションの名前です。

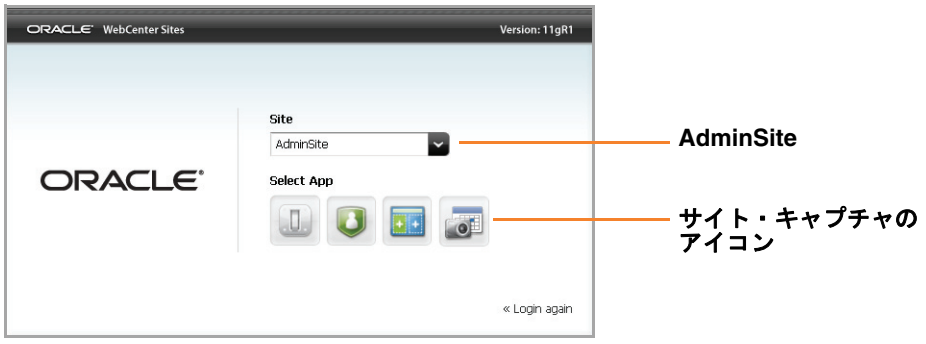
2. 全体管理者としてログインします。**ログイン資格証明では大 / 小文字が区別されます。**このガイドでは、次のようなデフォルトの資格証明を使用します。

   ユーザー名： `fwadmin`
   パスワード： `xceladmin`

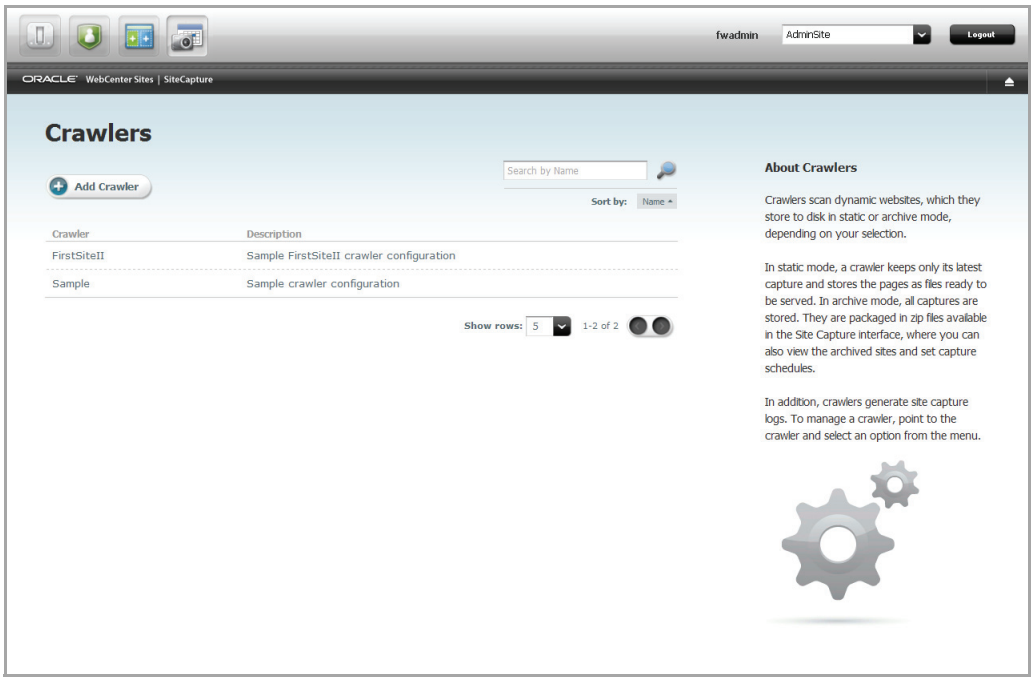


3. 「**ログイン**」をクリックします。
4. 初めてログインする場合、次の画面が開きます。

AdminSite (サイト・キャプチャ・アプリケーションは、デフォルトで AdminSite に割り当てられています) を選択し、サイト・キャプチャのアイコンを選択します。



5. 表示される最初の画面は「クローラ」という名前です。デフォルトのクローラがサイト・キャプチャとともにインストールされている場合は、それらが **Sample** と **FirstSiteII** という名前の下にリストされます。



6. 次の手順は、要件に応じて次のいずれかとなります。
   - デフォルトのクローラに関する詳細を学ぶ場合は、14 ページの「デフォルト・クローラの使用方法」に進みます。
   - 独自のサイト・キャプチャ操作を設定し、その過程で、サイト・キャプチャ・インタフェースのナビゲートについて学ぶ場合は、15 ページの「サイト・キャプチャ操作の設定」にスキップします。
   - クローラ構成コードについて学ぶ場合は、第 3 章「クローラ構成ファイルのコーディング」を参照してください。

デフォルト・クローラの使用方法

このガイドでは、デフォルトのクローラ「Sample」および「FirstSiteII」がサイト・キャプチャ・アプリケーションでインストールされていて、(13 ページの手順 5 で示すように) そのインタフェースで表示されることを前提としています。独自のクローラを定義する場合は、15 ページの「サイト・キャプチャ操作の設定」を参照してください。

Sample クローラ

「Sample」クローラはすべてのサイトのダウンロードに使用できます。「Sample」クローラの目的は、サイトを迅速にダウンロードできるように支援し、必須の構成コード (独自のクローラの作成時にこれを再利用します) を提供することです。「Sample」クローラは、必須メソッドと、クロールするリンク数を制限することによってクロールの期間を制限するオプションのメソッドを持つ最小構成になっています。

- 必須メソッドは `getStartURi` と `createLinkExtractor` (クロール済ページからリンクを抽出するためのロジックを定義します) です。
- オプションのメソッドは `getMaxLinks` で、クロールするリンク数を指定します。

これらのメソッドとクローラのカスタマイズ・メソッドおよびインタフェースの詳細は、第 3 章「クローラ構成ファイルのコーディング」を参照してください。

FirstSiteII クローラ

「FirstSiteII」クローラは、WebCenter Sites の動的 FirstSiteII サンプル Web サイトを静的サイトとしてダウンロードするために使用されます。このクローラの目的は、`LinkExtractor` および `ResourceRewriter` インタフェースを使用して、カスタム・リンク・エクストラクタとリソース・リライタの作成方法を示す、高度な構成コードを提供することです。インタフェースの詳細は、第 3 章「クローラ構成ファイルのコーディング」を参照してください。

デフォルト・クローラの実行

この項では、Sample クローラまたは FirstSiteII クローラを実行します。「FirstSiteII」クローラを使用するには、WebCenter Sites の FirstSiteII サンプル・サイトをパブリッシュする必要があります。

デフォルト・クローラを実行するには :

1. 「クローラ」画面で、デフォルト・クローラのいずれか (**Sample** または **FirstSiteII**) をポイントし、**「構成の編集」**を選択します。

> **注意**
>
> デフォルト・クローラがリストされていない場合は、15 ページの「サイト・キャプチャ操作の設定」にスキップして、独自のクローラを定義してください。

2. クローラの開始 URI は、クローラの構成ファイルを編集して設定します。手順については 20 ページの手順 2 にスキップして、クローラの実行およびそのキャプチャされたデータを管理するための残りの手順を続行してください。

サイト・キャプチャ操作の設定

この項では、サイト・キャプチャ・インタフェースとファイル・システムが編成される方法について理解するため、独自のクローラを作成し、実行するプロセスを段階的に説明します。基本的な手順は次のとおりです。

手順 1: 初期クローラ構成ファイルの作成

クローラを作成する前に、クローラのサイト・キャプチャ・プロセスを制御する構成ファイルが必要です。役立つファイルを作成する最速の方法は、サンプル・コードをコピーして、必要に応じてリコードすることです。

初期クローラ構成ファイルを作成するには :

1. **Sample** クローラの構成ファイルを次のいずれかの方法でローカル・マシンにコピーします。
   - サイト・キャプチャ・アプリケーションにログインします。「クローラ」画面で「Sample」クローラがリストされている場合は、次を実行します (そうでない場合は、直接下の項目にスキップします)。
     a) 「**Sample**」をポイントし、「**構成の編集**」を選択します。
     b) 「構成ファイル」フィールドに移動し、ローカル・マシン上のテキスト・ファイルにそのコードをコピーし、そのファイルを `CrawlerConfigurator.groovy` として保存します。

- サイト・キャプチャ・ホスト・マシンに移動し、CrawlerConfigurator.groovy ファイルを <SC_INSTALL_DIR>/fw-site-capture/crawler/Sample/app/ からローカル・マシンにコピーします。

注意

すべてのクローラは自身の CrawlerConfigurator.groovy ファイルにより制御されます。ファイルは、カスタム・フォルダ構造で格納されます。次に例を示します。

クローラを定義すると、サイト・キャプチャは、クローラの名前を持つフォルダ (このシナリオでは、<crawlerName> または Sample) を作成し、そのフォルダを <SC_INSTALL_DIR>/fw-site-capture/crawler/ というパスに置きます。<crawlerName> フォルダ内で、サイト・キャプチャは、ローカル・マシンから groovy ファイルのアップロード先となる /app サブフォルダを作成します。

クローラが初めて指定のモードで使用される場合、そのモードでキャプチャされたサイトを格納するため、サイト・キャプチャは (/<crawlerName>/ に) 追加のサブフォルダを作成します。サイト・キャプチャ・ファイル・システムの詳細は、34 ページの「静的にキャプチャされたサイトの管理」を参照してください。

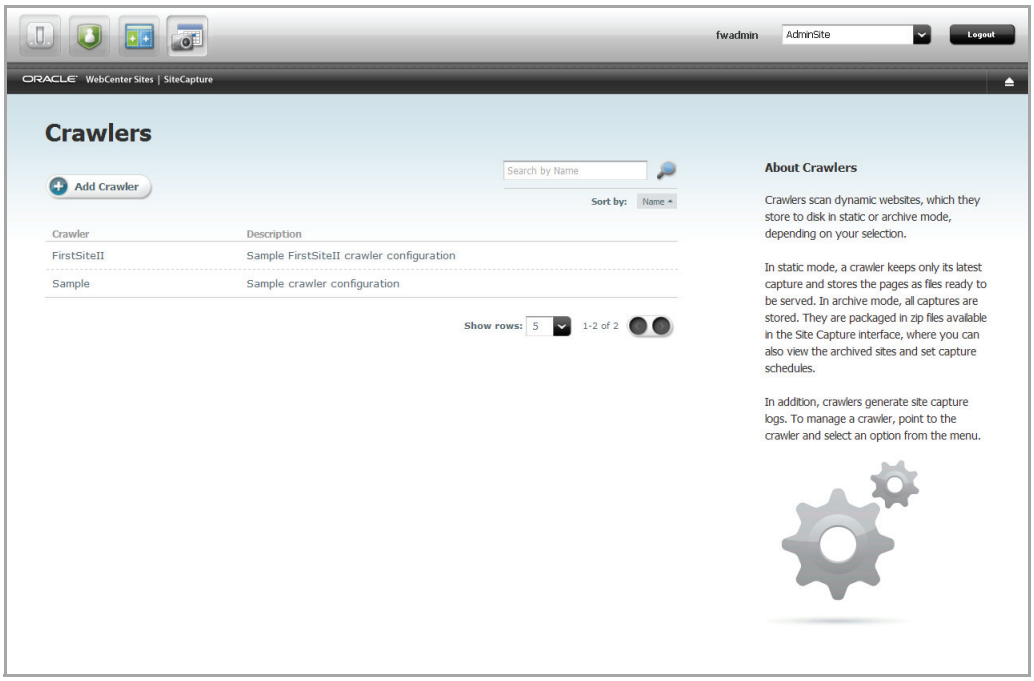
2. サンプル groovy ファイルは、サンプルの開始 URI を指定します。これは、次の手順で作成するクローラ用にリセットします。(開始 URI のほか、クロール深度などのパラメータを設定し、post-crawl コマンドを起動して、ターゲット・サイトに固有のロジックを定義するためのインタフェースを実装します。)

   この時点では、ダウンロードされた groovy ファイルをただちにカスタマイズするか、または最初にクローラを作成してからその groovy ファイル (サイト・キャプチャ・インタフェースで編集可能) をカスタマイズするかのオプションがあります。

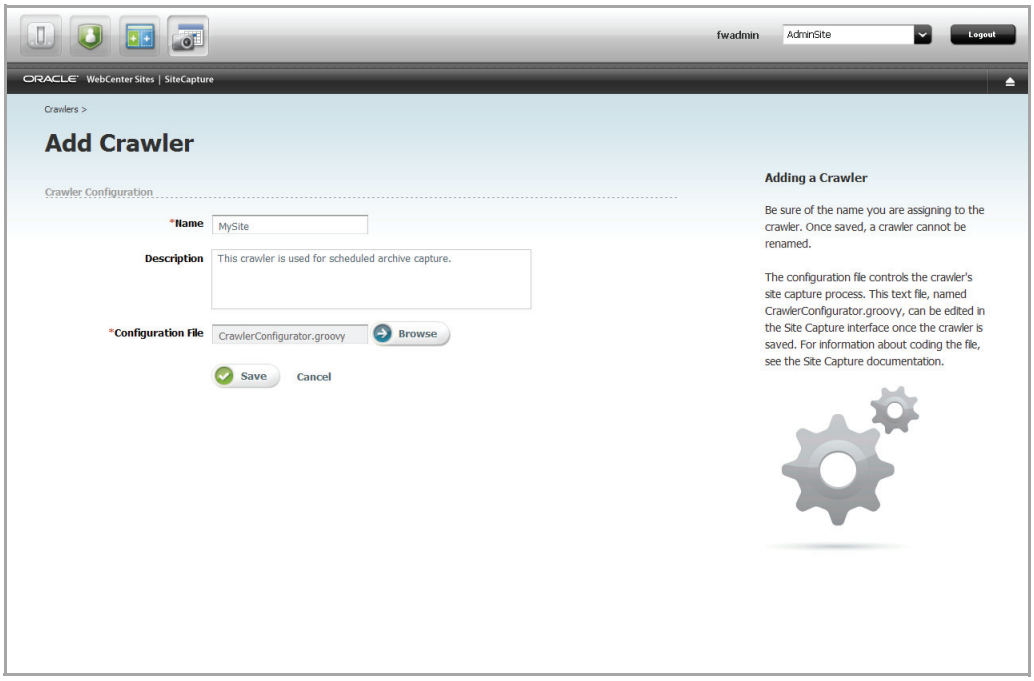
   - この演習を行うには、次の手順「手順 2: クローラの定義」に進んでください。
   - クローラの構成方法およびインタフェースの詳細は、第 3 章「クローラ構成ファイルのコーディング」を参照してください。

手順 2: クローラの定義

1. 「クローラ」画面に移動し、「**クローラの追加**」をクリックします。



2. 「クローラの追加」画面で次の手順を実行します。



a. クロール対象のサイトにちなんでクローラに名前を付けます。

注意

- いったん保存したら、クローラの名前は変更できません。
- このガイドでは、すべてのカスタム・クローラが、ターゲット・サイトにちなんだ名前を付けられ、その他のサイトを取得するために使用されないことを前提としています。

b. 説明を入力します (オプション)。例 : 「このクローラはパブリッシュ・トリガー・サイト・キャプチャ用に予約されています」または「このクローラはスケジュール済キャプチャ用に予約されています」など。

c. 「構成ファイル」フィールドで、15 ページの「手順 1: 初期クローラ構成ファイルの作成」で作成した `groovy` ファイルを参照します。

d. 新しいクローラを保存します。

`CrawlerConfigurator.groovy` ファイルは、サイト・キャプチャ・ホスト・マシンの `<SC_INSTALL_DIR>/fw-site-capture/crawler/<crawlerName>/app` フォルダにアップロードされます。このファイルは、サイト・キャプチャ・インタフェースで直接編集できます。

3. 19 ページの「手順 3: クローラ構成ファイルの編集」に進みます。

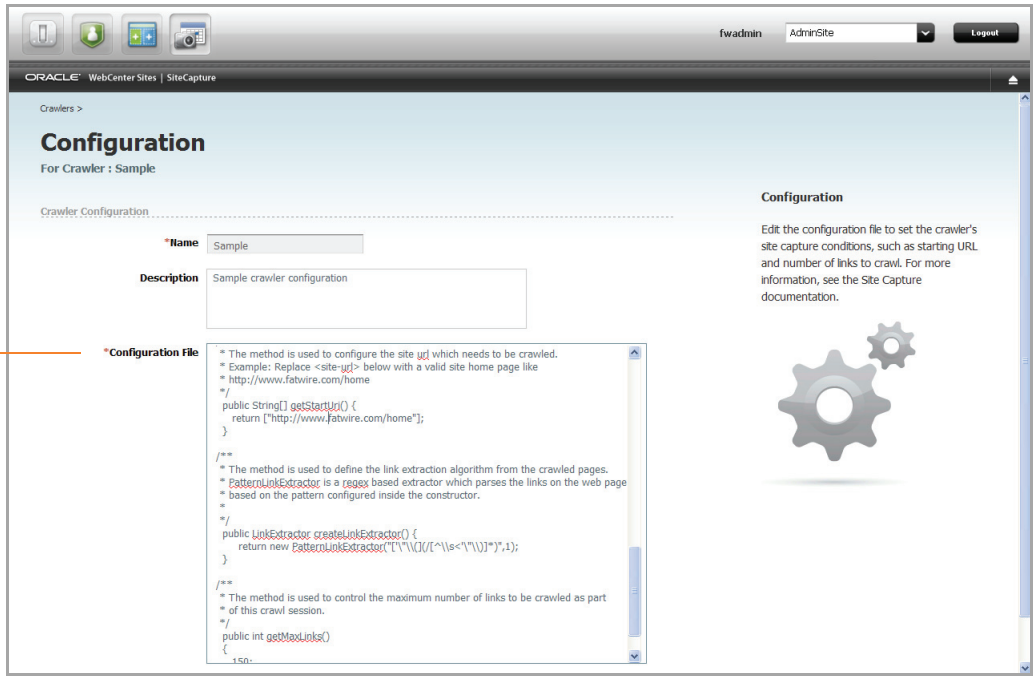
手順 3: クローラ構成ファイルの編集

サイト・キャプチャ・インタフェースから、クローラ構成ファイル全体をリコードできます。この例では、単純にクローラの開始 URI を設定します。

クローラの構成ファイルを編集するには :

1. 「クローラ」画面で、定義したクローラをポイントし、**「構成の編集」**を選択します。

このフィールドには、`<SC_INSTALL_DIR>/fw-site-capture/crawler/<crawlerName>/app` にある、クローラの `CrawlerConfigurator.groovy` ファイルが表示されます。



2. クローラの開始 URI を次のメソッドで設定します。

```
public String[] getStartUri() {
  return ["http://www.mycompany.com/home"]
```

注意

- 複数の開始 URI を設定できます。これらは同じサイトに属している必要があります。下の例に示すように、カンマ区切りの配列を入力します。

```
public String[] getStartUri()
 {
  return ["http://www.fatwire.com/product","http://
   www.fatwire.com/support"];
 }
```

- 構成ファイルには、クロール対象のリンクを抽出するためのロジックをコールする `createLinkExtractor` メソッドが含まれています。リンクは、クロール・セッション中にダウンロードされるマークアップから抽出されます。このメソッドおよび抽出ロジックの詳細は、46 ページの「createLinkExtractor」を参照してください。
- 構成ファイルには、クロールするリンク数を指定する `getMaxLinks()` メソッドも含まれています。デフォルト値は、迅速な実行のために、`150` に設定されます。なんらかの理由で静的キャプチャを停止する必要がある場合は、アプリケーション・サーバーを停止する必要があります。アーカイブ・キャプチャは、サイト・キャプチャ・インタフェースから停止できます。

クローラの構成方法およびインタフェースの詳細は、第 3 章「クローラ構成ファイルのコーディング」を参照してください。

3. 「保存」をクリックします。
4. 20 ページの「手順 4: クロールの開始」に進みます。

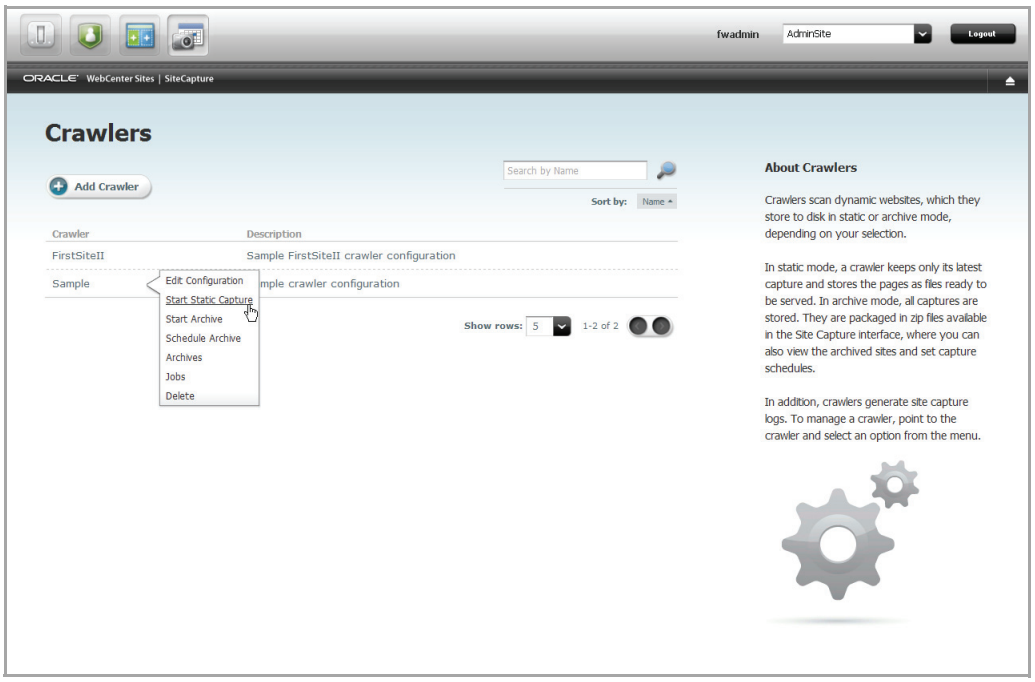
手順 4: クロールの開始

クロールは次のいくつかの方法で開始できます。

- 静的モードでのクローラの手動実行
- アーカイブ・モードでのクローラの手動実行
- クローラのアーカイブ・キャプチャのスケジュール (静的キャプチャはスケジュールできません)
- リアルタイム・モードでのサイトのパブリッシュ

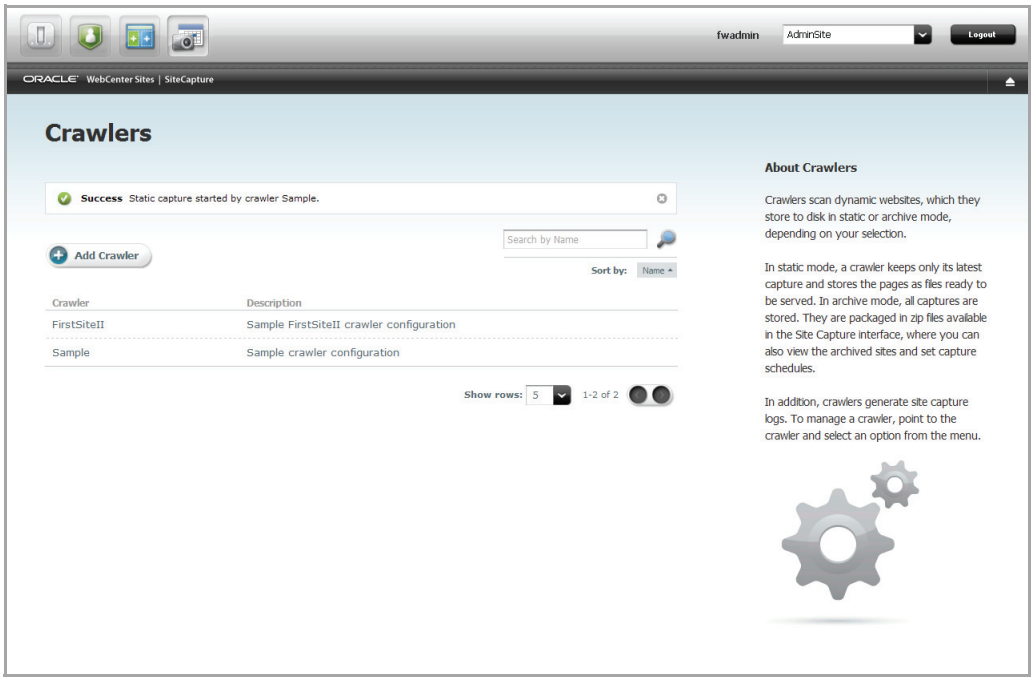
静的モードでのクローラの手動実行

1. 「クローラ」画面で、作成したクローラをポイントし、ポップアップ・メニューから**「静的キャプチャの開始」**を選択します。



「クローラ」画面には、キャプチャが開始されるときに次のメッセージが表示されます。

「**成功。**クローラ <crawlerName> によって静的キャプチャが開始されました。」



2. この時点では、サイト・キャプチャ・インタフェースにはクローラまたはそのプロセスに関する他の情報は表示されず、ダウンロード済サイトを使用することもできません。かわりに、サイト・キャプチャ・ファイル・システムを使用して、ダウンロード済ファイルや様々なログにアクセスします。
   - 静的キャプチャ・プロセスを監視するには、次のファイルを探します。
     - <SC_INSTALL_DIR>/fw-site-capture/<crawlerName>/logs の lock ファイル。lock ファイルは一時的なファイルです。追加の静的キャプチャを開始するためにクローラが起動されないよう、静的キャプチャ・プロセスの開始時に作成されます。lock ファイルはクローラ・セッションが終了すると削除されます。
     - <SC_INSTALL_DIR>/fw-site-capture/logs/ の crawler.log ファイル。(このファイルでは「VirtualHost」という用語が使用されていますが、これは「クローラ」を意味します。)
     - <SC_INSTALL_DIR>/fw-site-capture/<crawlerName> の inventory.db ファイル。このファイルは、クロール済 URL をリストします。**inventory.db ファイルは、サイト・キャプチャ・システムで使用されます。削除したり変更したりしないでください。**
     - audit.log、links.txt ファイルおよび report.txt ファイルは、/fw-site-capture/crawler/<crawlerName>/logs/yyyy/mm/dd にあります。
   - ダウンロード済ファイルにアクセスするには、<SC_INSTALL_DIR>/fw-site-capture/crawler/<crawlerName>/www に移動します。

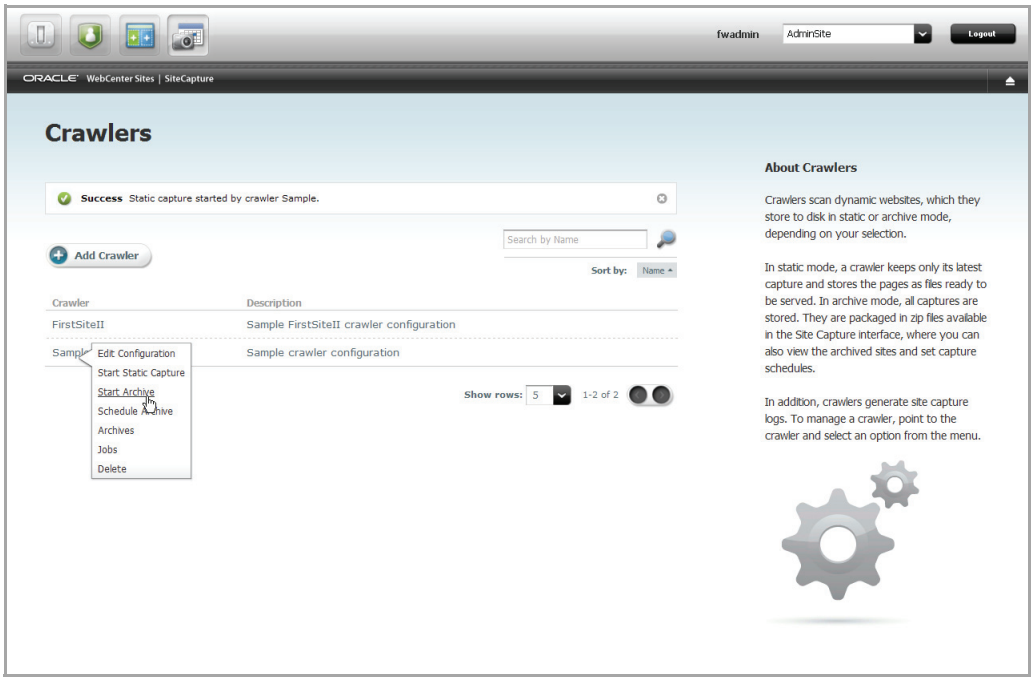
サイト・キャプチャ・ファイル・システムの詳細は、34 ページの「静的にキャプチャされたサイトの管理」を参照してください。

アーカイブ・モードでのクローラの手動実行

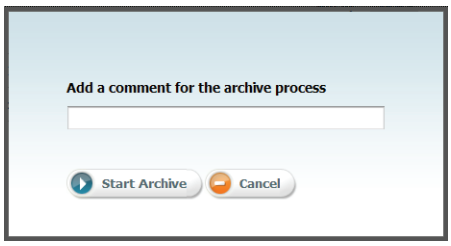
クローラがあるモードで使用されていた場合、それを別のモードで再実行できます。

クローラをアーカイブ・モードで実行するには :

1. 「クローラ」画面で、作成したクローラをポイントし、**「アーカイブの開始」**を選択します。



2. 次のダイアログでは、下記のようなことが可能です。
   - クローラの今後のジョブに関するコメントを追加できます。



注意

クローラの実行が開始されると、コメントを追加できません。

前述のダイアログでコメントを追加することを選択すると、コメントは次の場所に表示されます。

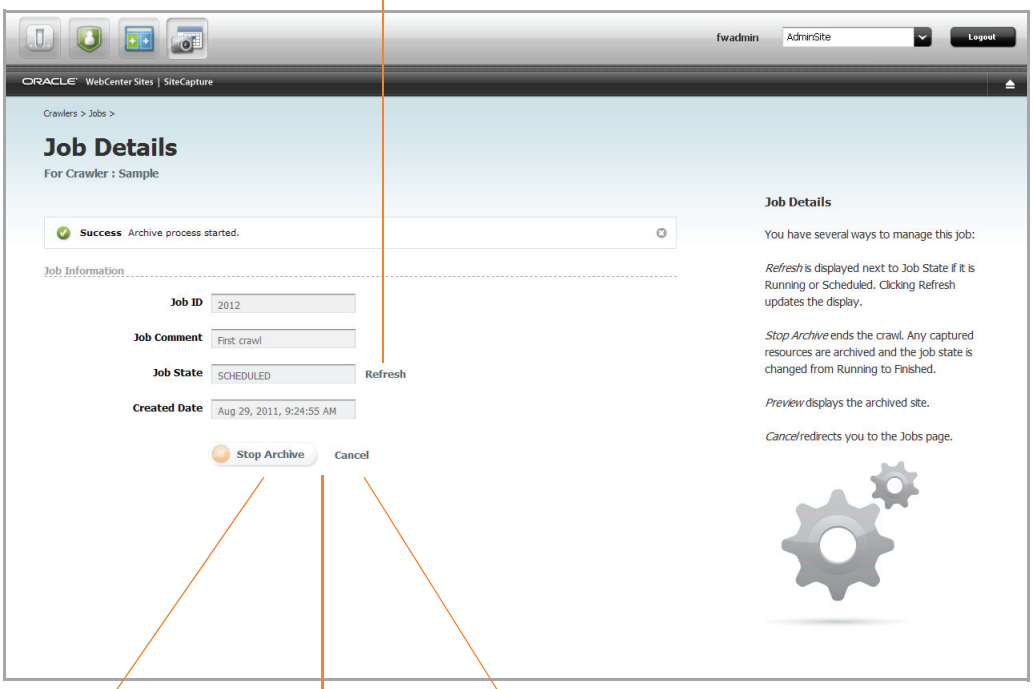
- 「ジョブの詳細」画面の「ジョブのコメント」フィールド (次の手順で説明します)。
- 「ジョブ」画面の「ジョブのコメント」フィールド (**「クローラ」→ *crawlerName* →「ジョブ」**)。
- 「アーカイブ」画面の「コメント」フィールド (**「クローラ」→ *crawlerName* →「アーカイブ」**)。

   - **「アーカイブの開始」**をクリックします。

3. 「ジョブの詳細」画面が表示されます。ここでは、次の図で示すように、アーカイブ・プロセスをいくつかの方法で管理できます。この演習を実行するには、「終了」が表示されるまで**「リフレッシュ」**(「ジョブの状態」の横にあります) をクリックし、次の手順に進みます。

「ジョブの詳細」画面には、アーカイブ・キャプチャを管理するためのいくつかのオプションが表示されます。

「リフレッシュ」は、ジョブの状態が「スケジュール済」または「実行中」であれば表示されます。**「リフレッシュ」**をクリックすると、表示されているジョブの状態が更新されます。可能なジョブの状態は、「スケジュール済」、「実行中」、「停止」および「失敗」です。

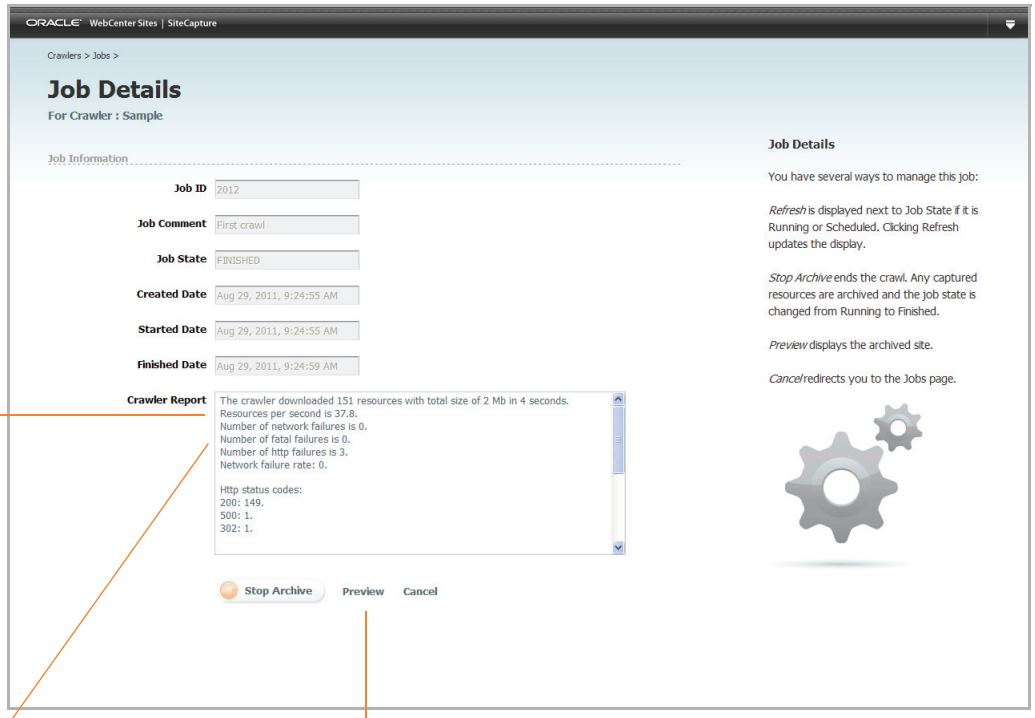


「アーカイブの停止」により、クローラのセッションが終了します。すべてのキャプチャ済リソースはアーカイブされ、ジョブの状態は「実行中」から 「終了」に変更されます (**「リフレッシュ」**をクリックして 変更を反映します)。

「取消」により、「ジョブ」画面にリダイレクトされます。クローラが実行中の場合は、引き続き実行されます。

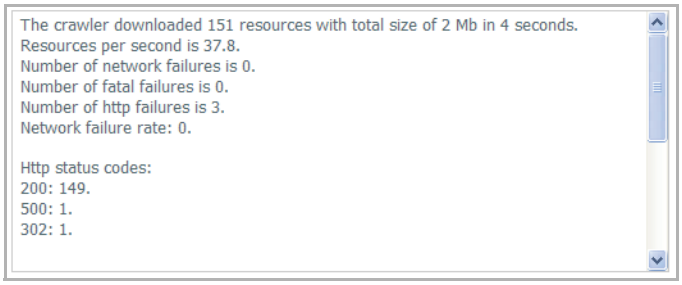
「プレビュー」は、ジョブの状態が「終了」の場合にここに表示されます。**「プレビュー」**をクリックすると、アーカイブ済サイトが表示されます。

4. アーカイブ・クロールが終了すると、結果がサイト・キャプチャ・インタフェースで使用可能になります。次に例を示します。
   - クローラ・レポートが「ジョブの詳細」画面に表示されます。レポートには、ダウンロード済リソース数、それらの合計サイズとダウンロード時間、ネットワーク状態、HTTP ステータス・コード、および必要に応じて追加のメモが記載されます。



クローラ・レポート

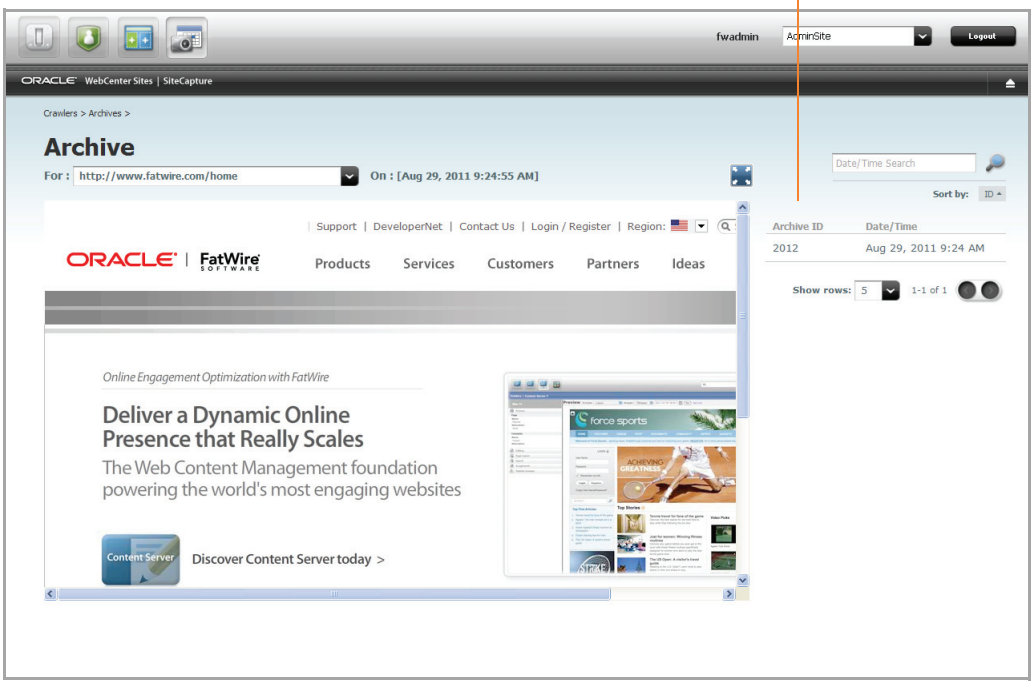
「プレビュー」をクリックすると、アーカイブ済サイトがレンダリングされます。



- 「ジョブの詳細」画面の**「プレビュー」**をクリックすると、アーカイブ済サイトがレンダリングされます (下図を参照)。サイトの横には、アーカイブ管理オプションのあるアーカイブ ID テーブルがあり、これは、アーカイブをポイントすると表示されます。

> **注意**
>
> アーカイブ済サイトに外部ドメインへのリンクが含まれる場合、(`CrawlerConfigurator.groovy` ファイルで) クロール深度とクロールするリンク数が大きな値に設定される場合には特に、そのプレビューには、これらのリンクが含まれることがあります。外部ドメインを参照することはできますが、それらはアーカイブされません。

アーカイブをポイントすると、その管理メニューが開きます (「プレビュー」、「ダウンロード」、「アーカイブ」および「URL の表示」というオプションがあります)。



- 様々なデータへの経路の要約は、37 ページの「アーカイブ済サイトの管理」を参照してください。

クローラのアーカイブ・キャプチャのスケジュール

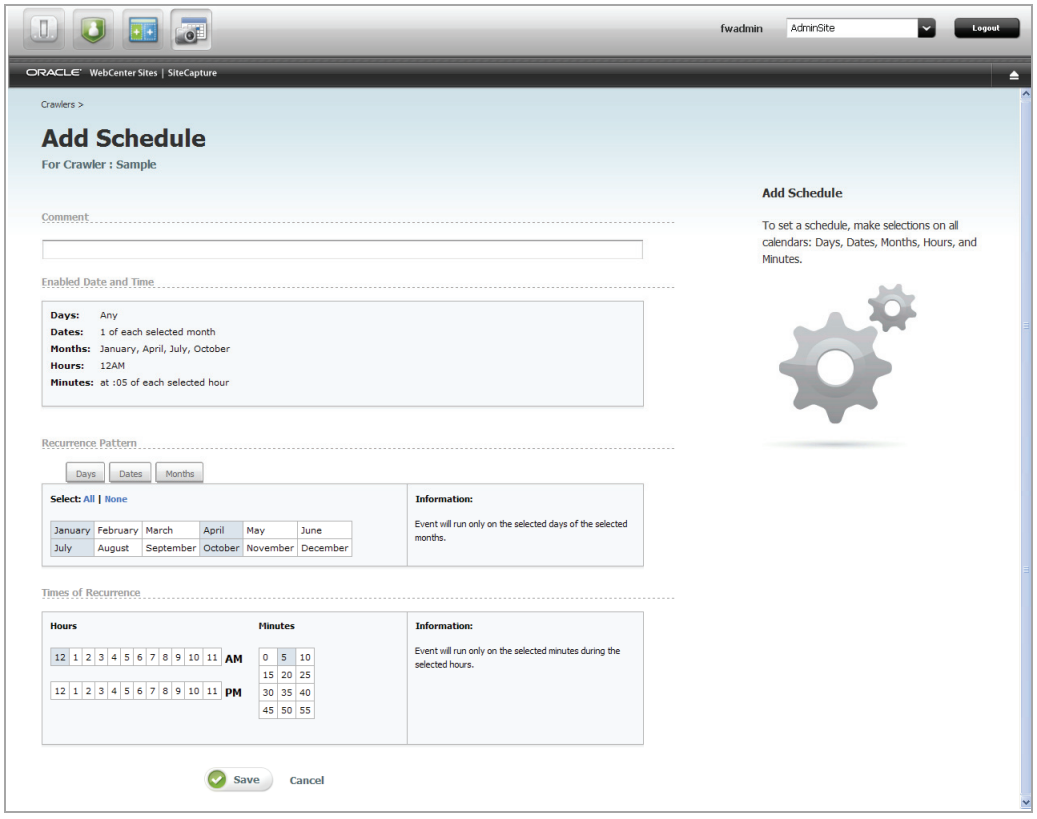
アーカイブ・キャプチャのみをスケジュールできます。あるクローラに対して、複数のスケジュールを作成できます。たとえば、定期的なキャプチャ用と、特定かつ一意の時間のキャプチャ用に別々のスケジュールを作成できます。

注意

複数のスケジュールを設定する場合は、それらが重ならないようにしてください。

クローラのアーカイブ・キャプチャをスケジュールするには :

1. 「クローラ」画面で、作成したクローラをポイントし、**「アーカイブのスケジュール」**を選択します。
2. **「スケジュールの追加」**をクリックし、すべてのカレンダーで「日」、「日付」、「月」、「時間」および「分」を選択します。



3. **「保存」**をクリックし、必要に応じて別のスケジュールを追加します。

リアルタイム・モードでのサイトのパブリッシュ

WebCenter Sites パブリッシュ・システムを構成して、サイト・キャプチャ・アプリケーションと通信する場合は、新たなパブリッシュ済サイトを取得するために 1 つ以上のクローラを起動するようにリアルタイム・パブリッシュ・プロセスを設定できます。手順については、30 ページの「パブリッシュ・トリガー・サイト・キャプチャの有効化」を参照してください。

手順 5: キャプチャ・データの管理

静的キャプチャおよびアーカイブ・キャプチャに関連した様々なデータにアクセスする方法については、第 2 章「ダウンロード・サイトの管理」を参照してください。39 ページの「要約」は、クローラとキャプチャ済データを管理する際に念頭に置くべき注意とヒントを集めたものです。次のトピックが含まれています。

- クローラの作成および編集
- クローラの削除
- クローラのスケジュール
- 静的クロールの監視
- クロールの停止
- アーカイブのダウンロード
- サイトのプレビュー
- パブリッシュの宛先定義の構成
- ログ・ファイルへのアクセス

パブリッシュ・トリガー・サイト・キャプチャの有効化

パブリッシュ・トリガー・サイト・キャプチャを有効にするための主要な手順は次のとおりです。

1. Oracle WebCenter Sites とのサイト・キャプチャ・アプリケーションの統合
2. サイト・キャプチャ用のリアルタイム・パブリッシュの宛先定義の構成
3. クローラの一致

管理ユーザーは、必要なだけサイト・キャプチャ用のパブリッシュの宛先定義を構成し、必要なだけクローラを起動できます。

Oracle WebCenter Sites とのサイト・キャプチャ・アプリケーションの統合

サイト・キャプチャ・アプリケーションが、パブリッシュ・プロセスで使用される WebCenter Sites ソースおよびターゲット・システムと最初に統合される場合のみ、リアルタイム・パブリッシュ・セッションの終わりにサイト・キャプチャを有効にできます。サイト・キャプチャが統合されていない場合は、統合手順について、*Oracle WebCenter Sites サイト・キャプチャ・アプリケーション・インストレーション・ガイド*を参照してから、次の手順に進んでください。

サイト・キャプチャ用のリアルタイム・パブリッシュの宛先定義の構成

パブリッシュの宛先定義を構成する際、パブリッシュ・セッションの終わりに起動されるクローラを指定します。キャプチャ・モードも指定します。

パブリッシュの宛先定義を構成するには :

1. サイト・キャプチャ・アプリケーションと統合される WebCenter Sites ソース・システムに移動します (*WebCenter Sites サイト・キャプチャ・アプリケーション・インストレーション・ガイド*を参照)。
   a. サイト・キャプチャと統合される WebCenter Sites ターゲット・システムにポイントするリアルタイム・パブリッシュの宛先を作成します。(リアルタイム・パブリッシュの宛先定義の作成方法の詳細は、*『Oracle WebCenter Sites 管理者ガイド』* を参照してください。)
   b. パブリッシュの宛先定義の「他の引数」セクションで、パブリッシュ・セッションの終わりに起動するクローラを指定し、クローラ起動を制御するために次のパラメータを使用して、キャプチャ・モードを設定します。
      - `CRAWLERCONFIG`: 各クローラの名前を指定します。複数のクローラを使用する場合は、名前をセミコロン (;) で区切ります。

        例 :
        単一クローラの場合 : `CRAWLERCONFIG=crawler1`

複数のクローラの場合 :
`CRAWLERCONFIG=crawler1;crawler2;crawler3`

注意

ここで指定するクローラは、サイト・キャプチャ・インタフェースで構成され、かつ同じ名前である必要もあります。クローラ名では大 / 小文字が区別されます。

- `CRAWLERMODE`: アーカイブ・キャプチャを実行するには、このパラメータを `dynamic` に設定します。
  デフォルトでは、静的キャプチャが有効になります。

  例 : `CRAWLERMODE=dynamic`

注意

- `CRAWLERMODE` モードが省略されるか、または `dynamic` 以外の値に設定された場合、パブリッシュ・セッションが終了すると、静的キャプチャが開始されます。
- 両方のクローラ・パラメータは次のように単一文で設定できます。
  `CRAWLERCONFIG=crawler1;crawler2&CRAWLERMODE=dynamic`
- 複数のクローラを指定できますが、設定できるモードは 1 つのみです。すべてのクローラがそのモードで実行されます。一部のクローラを別のモードで実行するには、別のパブリッシュの宛先定義を構成します。

2. 次の項に進みます。

クローラの一致

パブリッシュの宛先定義で指定したクローラは、サイト・キャプチャ・インタフェースに存在している必要があります。次を実行します。

- 宛先定義 (30 ページの手順 1b) とサイト・キャプチャ・インタフェースのクローラ名が同じであることを確認します。名前では大 / 小文字が区別されます。
- ターゲット・サイトの有効な開始 URI が各クローラの構成ファイルで設定されていることを確認します。クローラの構成ファイルへの移動の詳細は、19 ページの「手順 3: クローラ構成ファイルの編集」を参照してください。構成コードの記述の詳細は、第 3 章「クローラ構成ファイルのコーディング」を参照してください。

次の手順

1. パブリッシュ・トリガー・サイト・キャプチャが有効になると、ターゲット・サイトをパブリッシュする準備が整います。パブリッシュが終了すると、サイト・キャプチャが開始されます。パブリッシュの宛先定義で CRAWLERMODE パラメータをどのように設定したかに応じて (30 ページの手順 1b)、静的モードまたはアーカイブ・モードのいずれかで、起動されたクローラがページをキャプチャします。
2. サイト・キャプチャ・プロセスを監視するには、次の手順を実行します。
   - 静的キャプチャの場合、サイト・キャプチャ・インタフェースにはクロールに関する情報が表示されず、またキャプチャ済サイトも使用可能になりません。
     - クローラが起動したかどうかを判断するには、ソースまたはターゲットの WebCenter Sites システムで、futuretense.txt ファイルを開きます。

       注意

       WebCenter Sites ソースおよびターゲット・システム上の futuretense.txt ファイルには、静的およびアーカイブのいずれのタイプのクロールに関するクローラ起動ステータスが含まれています。

     - キャプチャ・プロセスを監視するには、サイト・キャプチャ・ファイル・システムに移動し、22 ページにリストされているファイルを確認します。
   - 動的キャプチャの場合、サイト・キャプチャ・インタフェースからクロールのステータスを表示できます。
     a) 「クローラ」画面に移動し、クローラをポイントして、ポップアップ・メニューから**「ジョブ」**を選択します。
     b) 「ジョブの詳細」画面で、「終了」が表示されるまで、「ジョブの状態」の横の**「リフレッシュ」**をクリックします。(「ジョブの状態」で可能な値は、「スケジュール済」、「実行中」、「停止」または「失敗」です。) 「ジョブの詳細」画面の詳細は、25 および 26 ページを参照してください。
3. キャプチャ済データを管理します。

   クロール・セッションが終了したら、キャプチャ済サイトと関連データを次のように管理できます。
   - 静的にキャプチャされたサイトの場合は、サイト・キャプチャ・ファイル・システムに移動します。詳細は、34 ページの「静的にキャプチャされたサイトの管理」を参照してください。
   - アーカイブ済サイトの場合は、サイト・キャプチャ・インタフェースを使用してサイトをプレビューして、zip ファイルとログをダウンロードします。詳細は、37 ページの「アーカイブ済サイトの管理」を参照してください。

第 2 章

ダウンロード・サイトの管理

ダウンロード・サイトは、静的にキャプチャされるか、アーカイブされるかによって、サイト・キャプチャ・ファイル・システムまたはインタフェースのいずれかから管理されます。

この章は、次の項で構成されています。

静的にキャプチャされたサイトの管理

サイト・キャプチャ・インタフェースでユーザーが作成するすべてのクローラについて、サイト・キャプチャでは、そのファイル・システム内に同じ名前を持つフォルダが作成されます。このカスタム・フォルダ <crawlerName> を使用して、図 1 で示すように、クローラの構成ファイルを編成し、キャプチャして、ログを記録します。表 1 では、<crawlerName> フォルダとそのコンテンツを示しています。

注意

静的キャプチャおよびログにアクセスするには、ファイル・システムを使用する必要があります。アーカイブ・キャプチャおよびログは、サイト・キャプチャ・インタフェースから管理されます (ファイル・システム内のこれらの場所については、この項で説明します)。

図 1: サイト・キャプチャのカスタム・フォルダ : <crawlerName>

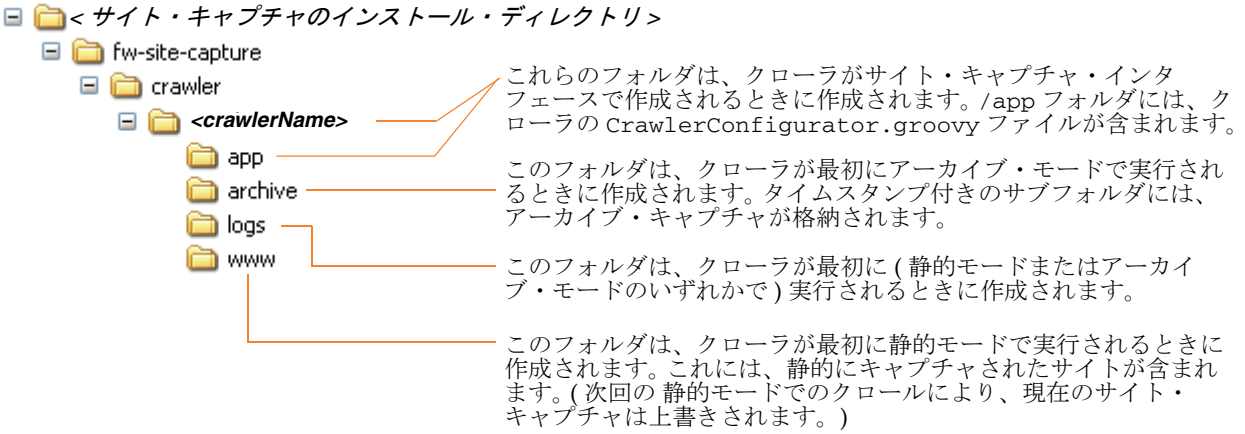
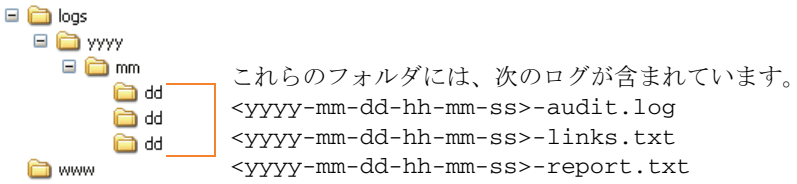


表 1: `<crawlerName>` フォルダおよびそのコンテンツ

フォルダ	説明
`/fw-site-capture/crawler/<crawlerName>`	クローラを表します。サイト・キャプチャ・インタフェースでユーザーが定義するすべてのクローラについて、サイト・キャプチャでは、`/<crawlerName>` フォルダが作成されます。たとえば、サンプル・クローラ「FirstSiteII」および「Sample」をインストールした場合、サイト・キャプチャ・インタフェースには両方のクローラが表示され、サイト・キャプチャ・ファイル・システムに同じ名前を持つフォルダが存在するようになります。 **注意 :** サブフォルダ (下記を参照) のほか、`<crawlerName>` フォルダには、静的にクロールされた URL がリストされている `inventory.db` ファイルが含まれています。このファイルは、クローラがその最初の静的キャプチャを取得したときに作成されます。**`inventory.db` を削除または変更しないでください。**これはサイト・キャプチャ・システムによって使用されます。
`/fw-site-capture/crawler/<crawlerName>/app`	クローラの `CrawlerConfiguration.groovy` ファイルが含まれます。そのコードはクロール・プロセスを制御します。クローラが作成され、保存されると、`/app` フォルダが作成されます。
`/fw-site-capture/crawler/<crawlerName>/archive`	`/archive` フォルダは、完全にアーカイブ・キャプチャ用にのみ使用されます。 このフォルダには、`yyyy/mm/dd` サブフォルダの階層が含まれます。`/dd` サブフォルダには、クローラのアーカイブ・キャプチャのすべてがタイムスタンプ付きの zip ファイルとして格納されます。 `/archive` フォルダは、クローラが最初にアーカイブ・モードで実行されるときに作成されます。zip ファイル (`/dd` にあります) は、データベースで参照されるため、サイト・キャプチャ・インタフェースで Web サイトとしてダウンロードおよび表示できるようになります。 **注意 :** アーカイブ・キャプチャは、サイト・キャプチャ・インタフェースからアクセスできます。各 zip ファイルには、`__inventory.db` という名前の URL ログが含まれます。**`__inventory.db` を削除または変更しないでください。**これはサイト・キャプチャ・システムによって使用されます。

表 1: `<crawlerName>` フォルダおよびそのコンテンツ *(続き)*

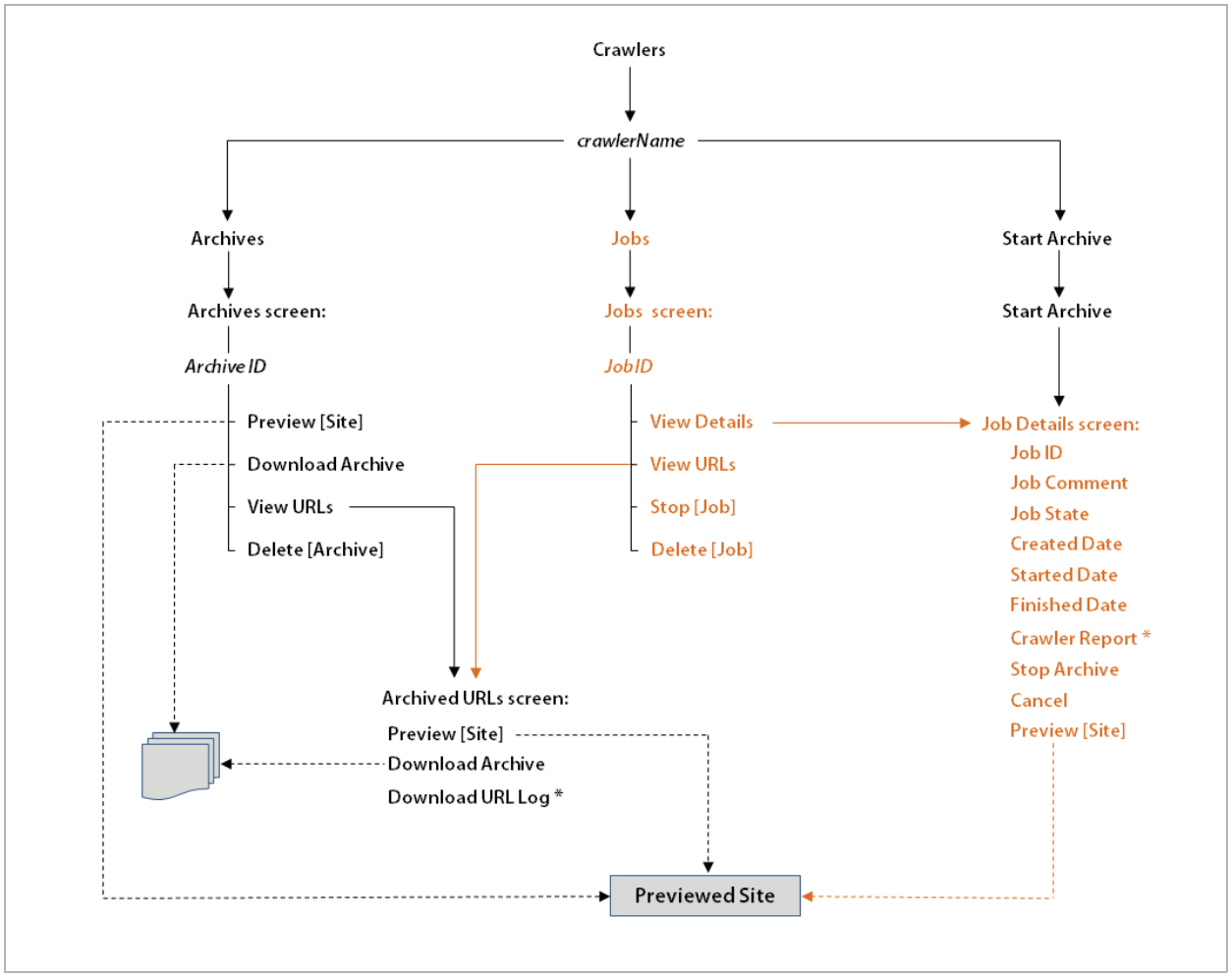
フォルダ	説明
`/fw-site-capture/crawler/<crawlerName>/www`	最新の*静的に*キャプチャされたサイトのみが含まれます (同じクローラが静的モードで再実行されると、前のキャプチャが上書きされます)。サイトは、`html`、`css`、および提供の準備が整っている他のファイルとして格納されます。 `/www` フォルダは、クローラが最初に静的モードで実行されるときに作成されます。 **注意 :** 静的キャプチャは、サイト・キャプチャ・ファイル・システムからのみアクセスできます。
`/fw-site-capture/crawler/<crawlerName>/logs/yyyy/mm/dd`	クロールされた URL に関する情報が記載されたログ・ファイルが含まれます。ログ・ファイルは、`/dd` サブフォルダに格納され、下図に示すように名前が付けられます。  • `audit.log` ファイルは、タイムスタンプ、クロール深度、HTTP ステータス、ダウンロード時間などのデータとともにクロール済 URL をリストします。 • `links.txt` ファイルは、クロール済 URL をリストします。 • `report.txt` ファイルは、ダウンロード済リソース数、合計サイズ、ダウンロード・サイズおよび時間、ネットワーク状態などのクロール統計全体をリストします。アーカイブ・キャプチャでは、このレポートを、クローラ・レポートとしてサイト・キャプチャ・インタフェースで使用できます (「ジョブの詳細」画面。「ジョブの詳細」画面へのパスは、38 ページの図 2 に記載されています)。
`/fw-site-capture/crawler/<crawlerName>/logs` *(続き)*	**注意 :** クローラが静的モードとアーカイブ・モードの両方でキャプチャした場合、`/dd` サブフォルダには、静的キャプチャとアーカイブ・キャプチャのログが含まれます。 `/logs` フォルダは、`lock` という名前の一時ファイルを格納するためにも使用されます。このファイルは、静的キャプチャ・プロセスの開始時に作成され、追加の静的キャプチャに対してクローラが起動されないようにします。`lock` ファイルはクローラ・セッションが終了すると削除されます。

アーカイブ済サイトの管理

アーカイブ済サイトは、サイト・キャプチャ・インタフェースの様々な画面から管理できます。図 2 は、アーカイブ、ジョブ、サイト・プレビュー、クローラ・レポート、URL ログなど様々な情報への経路の一部を示しています。

- たとえば、サイトをプレビューするには、「クローラ」画面で開始し、クローラ (***crawlerName***) をポイントして、ポップアップ・メニュー (「アーカイブ」画面を開くメニュー) から**「アーカイブ」**を選択し、**「アーカイブ ID」**をポイントして、ポップアップ・メニューから**「プレビュー」**を選択します。
- 破線は、同じオプションへの複数のパスを示しています。たとえば、サイトをプレビューするには、クローラの**「アーカイブ」**パス、**「ジョブ」**パス、または**「アーカイブの開始」**パスをたどることができます。アーカイブをダウンロードするには、**「アーカイブ」**パスまたは**「ジョブ」**パスをたどることができます。
- クローラ・レポートおよび URL ログは、アスタリスク (*) でマークされています。

図 2: アーカイブ済情報へのパス



要約

この項では、クローラおよびキャプチャされたデータを管理するための注意点とヒントをまとめています。

クローラの作成および編集

クローラを作成し、その構成コードを編集する際には、次の情報を考慮してください。

- クローラ名では大 / 小文字が区別されます。
- すべてのクローラの構成ファイルには、`CrawlerConfigurator.groovy` という名前が付けられます。このファイルは、依存関係を注入するために使用されます。そのため、この名前は変更しないでください。
- クローラは、指定のサイト上で 1 つまたは複数のシード URI で開始し、1 つまたは複数のパスをクロールするように構成できます。追加の Java メソッドにより、クロール深度などのパラメータを設定したり、post-crawl コマンドを起動したり、セッション・タイムアウトを指定したりできます。リンクを抽出し、URL をリライトして、クロール・セッションの終わりに電子メールを送信するためのロジックを定義するようインタフェースを実装できます。詳細は、第 3 章「クローラ構成ファイルのコーディング」を参照してください。
- クローラが作成されて保存されると、その `CrawlerConfigurator.groovy` ファイルは、サイト・キャプチャ・ファイル・システムにアップロードされ、サイト・キャプチャ・インタフェースで編集可能になります。
- クローラが静的サイト・キャプチャ・プロセスを実行している間は、2 番目の静的キャプチャ・プロセスを起動して実行することはできません。
- クローラがアーカイブ・キャプチャ・プロセスを実行している間は、2 番目のアーカイブ・キャプチャ・プロセスを起動して実行できます。2 番目のプロセスは「スケジュール済」としてマークされ、最初のプロセスが終了した後で開始されます。

クローラの削除

クローラ (これには、キャプチャされた情報のすべてが含まれます) を削除する必要がある場合は、ファイル・システムではなく、サイト・キャプチャ・インタフェースから行ってください。インタフェースから削除すると、リンクが壊れるのを防ぐことができます。たとえば、クローラがアーカイブ・モードで実行された場合、インタフェースからそれを削除すると、次の 2 セットの情報が削除されます : (1) クローラのアーカイブおよびログ、(2) これらのアーカイブおよびログへのデータベース参照。(ファイル・システムからクローラを削除すると、データベース参照がアーカイブに保持されたまま、それらがもう存在していないことがログに記載され、サイト・キャプチャ・インタフェースで壊れたリンクが作成されます。)

クローラのスケジュール

アーカイブ・クロールのみをスケジュールできます。

- クローラのスケジュールを設定する場合、サイトのパブリッシュ・スケジュールを考慮し、2 つが重ならないようにします。
- 単一クローラに対して複数のスケジュールを作成できます。たとえば、クローラを定期的に起動するためのスケジュールを 1 つ作成し、特定の一意の時間にクローラを起動する別のスケジュールを作成できます。
- 複数のスケジュールを作成する場合は、それらが重ならないようにしてください。

静的クロールの監視

静的クロールが進行中であるか、完了しているかを判断するには、`<SC_INSTALL_DIR>/fw-site-capture/<crawlerName>/logs` フォルダでクローラの `lock` ファイルを探します。`lock` ファイルは一時的なファイルです。追加の静的キャプチャを開始するためにクローラが起動されないよう、静的キャプチャ・プロセスの開始時に作成されます。`lock` ファイルはクローラ・セッションが終了すると削除されます。

クロールの停止

クローラを実行する前に、クロールされるリンク数およびクロール深度を考慮してください。クロールのセッションの期間は、その両方で決まります。

- アーカイブ・クロールを終了する必要がある場合は、サイト・キャプチャ・インタフェース (**「ジョブの詳細」**画面→**「アーカイブの停止」**) を使用します。
- 静的クロールを終了する必要がある場合は、アプリケーション・サーバーを停止する必要があります。

アーカイブのダウンロード

サイト・キャプチャ・インタフェースからは (250MB を超える) 大きなアーカイブ・ファイルをダウンロードしないでください。かわりに、`getPostExecutionCommand` を使用して、サイト・キャプチャ・ファイル・システムから希望の場所にファイルをコピーします。

アーカイブ・サイズは、「ジョブの詳細」画面のクローラ・レポートから取得できます。「ジョブの詳細」画面へのパスは、38 ページの図 2 に記載されています。`getPostExecutionCommand` メソッドの詳細は、50 ページを参照してください。

サイトのプレビュー

アーカイブ済サイトに外部ドメインへのリンクが含まれる場合、(クロールの `groovy` ファイルで) クロール深度とクロールするリンク数が大きな値に設定される場合は特に、そのプレビューには、これらのリンクが含まれる場合があります。外部ドメインを参照することはできますが、それらはアーカイブされません。

パブリッシュの宛先定義の構成

- パブリッシュ・トリガー・サイト・キャプチャを実行する場合は、パブリッシュの宛先定義の単一文でクローラ・パラメータを設定できます。

  `CRAWLERCONFIG=crawler1;crawler2&CRAWLERMODE=dynamic`

- パブリッシュの宛先定義で複数のクローラを指定できますが、設定できるキャプチャ・モードは 1 つのみです。すべてのクローラがそのモードで実行されます。一部のクローラを別のモードで実行するには、別のパブリッシュの宛先定義を構成する必要があります。

ログ・ファイルへのアクセス

- 静的にキャプチャされたサイトの場合、ログ・ファイルは、サイト・キャプチャ・ファイル・システムでのみ使用可能です。
  - 静的にクロールされた URL をリストしている `inventory.db` ファイルは、`/fw-site-capture/crawler/<crawlerName>` フォルダにあります。

    注意

    `inventory.db` ファイルはサイト・キャプチャ・システムによって使用されます。削除したり変更したりしないでください。

  - `crawler.log` ファイルは、`<SC_INSTALL_DIR>/fw-site-capture/logs/` フォルダにあります。(`crawler.log` ファイルでは「VirtualHost」という用語が使用されていますが、これは「クローラ」を意味します。)
- 静的にキャプチャされ、アーカイブされたサイトの場合、共通のログ・ファイル・セットがサイト・キャプチャ・ファイル・システムに存在します。
  - `audit.log`。クロール済 URL、タイムスタンプ、クロール深度、HTTP ステータスおよびダウンロード時間をリストします。
  - `links.txt`。クロール済 URL をリストします。
  - `report.txt`。クローラ・レポートです。

  前述の名前のファイルは、次のフォルダにあります。
  `/fw-site-capture/crawler/<crawlerName>/logs/yyyy/mm/dd`

  注意

  アーカイブ済サイトの場合、`report.txt` は、サイト・キャプチャ・インタフェースの「ジョブの詳細」画面で使用できます。ここでは、「クローラのレポート」と呼ばれます (「ジョブの詳細」画面へのパスは、38 ページの図 2 に記載されています)。

- アーカイブ・プロセスでは、すべてのクロールの URL ログも生成されます。ログは次の 2 つの場所で使用できます。

- サイト・キャプチャ・ファイル・システムでは、__inventory.db と呼ばれます。このファイルは、次のフォルダの zip ファイル内にあります。
/fw-site-capture/crawler/<crawlerName>/archive/yyyy/mm/dd

注意

__inventory.db ファイルはサイト・キャプチャ・システムによって使用されます。削除したり変更したりしないでください。

- サイト・キャプチャ・インタフェースの「アーカイブ済 URL」画面 (このパスは 38 ページの図 2 に記載されています)。

第 3 章

クローラ構成ファイルのコーディング

この章には、BaseConfigurator クラス、クローラのサイト・キャプチャ・プロセスを制御するためのメソッドとインタフェースの実装、FirstSiteII クローラのサイト・キャプチャ・インストレーションで使用可能なサンプル・コードに関する情報が含まれています。

この章は、次の項で構成されています。

概要

クローラを制御するには、その CrawlerConfigurator.groovy ファイルを少なくとも開始 URI とリンク抽出ロジックを使用してコーディングする必要があります。開始 URI とリンク抽出ロジックはいずれも、getStartUri() および createLinkExtractor() メソッドを介して提供されます。追加のコードは、たとえば、クロールするリンク数、クロール深度、静的にダウンロードしたファイルを Web サーバーの doc ベースにコピーするといった post-crawl イベントの起動などを指定するため、必要に応じて追加できます。

使用するメソッドおよびインタフェースは、BaseConfigurator クラスで提供されます。デフォルト実装をオーバーライドして、ターゲット・サイトの構造や収集する必要のあるデータに一致する方法で、クロール・プロセスをカスタマイズおよび制御できます。

この章では、BaseConfigurator メソッドと単純な CrawlerConfigurator.groovy ファイルから開始して、必要なメソッドの使用方法を示します。クローラのカスタマイズ・メソッドを説明し、次に、デフォルトおよびカスタム実装を含むサイト・キャプチャの Java インタフェースについて説明します。

BaseConfigurator メソッド

CrawlerConfigurator.groovy ファイルには、CrawlerConfigurator クラスのコードが含まれています。このクラスは、クローラのデフォルト実装を提供する抽象クラスである BaseConfigurator を拡張する必要があります。表 1 は、BaseConfigurator クラスのメソッドとインタフェースを示しています。これらについては、後続の項で説明します。基本的なサンプルの CrawlerConfigurator.groovy ファイルを、47 ページに示します。

表 1: BaseConfigurator **クラスのメソッド**

メソッド・タイプ	メソッド	注意
必須	“getStartUri”	
	“createLinkExtractor”	“LinkExtractor” インタフェースのファクトリ・メソッド。[a,b]
クローラのカスタマイズ	“getMaxLinks”	
	“getMaxCrawlDepth”	
	“getConnectionTimeout”	
	“getSocketTimeout”	
	“getPostExecutionCommand”	
	“getNumWorkers”	
	“getUserAgent”	
	“createResourceRewriter”	“ResourceRewriter” インタフェースのファクトリ・メソッド。[a、b]
	“createMailer”	“Mailer” インタフェースのファクトリ・メソッド。[a]

表 1: `BaseConfigurator` クラスのメソッド

メソッド・タイプ	メソッド	注意
	“getProxyHost” “getProxyCredentials”	

a. ここに示されているインタフェースは、この章で説明するデフォルト実装を持ちます。

b. サイト・キャプチャは、FirstSiteII サンプル・クローラで使用される、サンプル・リンク・エクストラクタとリソース・リライタの両方を提供します。“ カスタム・リンク・エクストラクタの記述とデプロイ ” および “ カスタム ResourceRewriter の記述 ” を参照してください。

必要なメソッド

`BaseConfigurator` の 2 つの abstract メソッドは、`CrawlerConfigurator` でオーバーライドされる必要があります。これらは、`getStartUri()` および `createLinkExtractor()` です。

getStartUri

このメソッドは、クローラの開始 URI を注入するために使用されます。URI が同じサイトに属しているかぎり、クロールに対して 1 つまたは複数の開始 URI を構成できます。複数の開始ポイントを指定すると、クロールが同時に開始されます。

例 : `www.fatwire.com` サイトの開始 URI を提供するには、次のようにします。

```
/**
 * The method is used to configure the site url which needs to
    be crawled.
 */
  public String[] getStartUri()
  {
        return ["http://www.fatwire.com/home"]; //Groovy uses
           brackets for an array.
  }
```

例 : www.fatwire.com サイトに複数の開始 URI を提供するには、カンマ区切りの配列を入力します。

```
/**
 * The method is used to configure the site url which needs to
    be crawled.
 */
  public String[] getStartUri()
  {
        return ["http://www.fatwire.com/product","http://
           www.fatwire.com/support"]; //Groovy uses brackets
           for an array.
  }
```

createLinkExtractor

このメソッドは、クロール済ページからリンクを抽出するロジックを構成するために使用されます。その後、抽出されたリンクは横断されます。createLinkExtractor メソッドは、LinkExtractor インタフェースのファクトリ・メソッドです。

- LinkExtractor インタフェースを実装して、独自のリンク抽出アルゴリズム (たとえば、HTML パーサーを使用して、ページを解析し、クローラが消費するリンクを抽出するなど) を作成できます。
- 正規表現を使用してリンクを抽出するデフォルト実装 PatternLinkExtractor も使用できます。たとえば、PatternLinkExtractor を使用して、46 ページで示すように、Products などの表現からフォーマット /home/products のリンクを抽出できます。

例 : 正規表現を使用して、www.fatwire.com サイト上で Products からリンクを抽出するには、次のようにします。

```
/**
 * The method is used to define the link extraction
    algorithm from the crawled pages.
 * PatternLinkExtractor is a regex based extractor which
    parses the links on the web page
 * based on the pattern configured inside the constructor.
 *
 */
public LinkExtractor createLinkExtractor()
{
        return new PatternLinkExtractor("['¥"¥¥(](/
           [^¥¥s<'¥"¥¥)]*)",1);
}
```

- 正規表現および PatternLinkExtractor の詳細は、56 ページの「LinkExtractor のデフォルト実装の使用方法」を参照してください。

- LinkExtractor インタフェースの実装に関する詳細は、58 ページの「カスタム・リンク・エクストラクタの記述とデプロイ」を参照してください。

基本的な構成ファイル

下記は、単純な CrawlerConfigurator.groovy ファイルの例です。このファイルでは、必要なメソッド getStartUri() および createLinkExtractor() がオーバーライドされます。これらのコードは、行 35 で始まります。

注意

次のサンプルでは、48 ページで説明されている追加のメソッド getMaxLinks() をオーバーライドします。サンプルでは、テストの実行を迅速に完了できるようにするために、150 を戻すように設定されています。

CrawlerConfigurator.groovy という名前のファイルは、依存関係を注入するために使用されます。そのため、この名前は変更しないでください。

```
1  package com.fatwire.crawler.sample
2
3  import java.text.DateFormat;
4  import java.text.SimpleDateFormat;
5
6  import java.util.regex.Pattern;
7
8  import javax.mail.internet.AddressException;
9  import javax.mail.internet.InternetAddress;
10
11 import com.fatwire.crawler.*;
12 import com.fatwire.crawler.remote.*;
13 import com.fatwire.crawler.remote.di.*;
14 import com.fatwire.crawler.impl.*;
15 import com.fatwire.crawler.util.FileBuilder;
16
17 import org.apache.commons.lang.SystemUtils;
18 import org.apache.http.HttpHost;
19 import org.apache.http.auth.*;
20 import org.apache.http.client.*;
21 import org.apache.http.impl.client.*;
22 /**
23 * Configurator for the crawler.
24 * This is used to inject the dependency inside the crawler
   to control the crawling process
25 */
26
27 public class CrawlerConfigurator extends BaseConfigurator {
28
29 public CrawlerConfigurator(GlobalConfigurator delegate){
30 super(delegate);
31 }
```

```
32
33
34 /**
35 * The method is used to configure the site url which needs
   to be crawled.
36 */
37 public String[] getStartUri() {
38 return ["http://www.fatwire.com/home"]; //Groovy uses
   brackets for an array.
39 }
40
41 /**
42 * The method is used to define the link extraction
   algorithm from the crawled pages.
43 * PatternLinkExtractor is a regex based extractor which
   parses the links on the web page
44 * based on the pattern configured inside the constructor.
45 *
46 */
47 public LinkExtractor createLinkExtractor() {
48 return new PatternLinkExtractor("['¥"¥¥(](/
   [^¥¥s<'¥"¥¥)]*)",1);
49 }
50
51 /**
52 * The method is used to control the maximum number of links
   to be crawled as part
53 * of this crawl session.
54 */
55 public int getMaxLinks()
56 {
57 150;
58 }
}
```

クローラ・カスタマイズ・メソッド

必要なメソッドの他に、BaseConfigurator クラスには、デフォルト実装を持つメソッドがあり、これらのメソッドは、ターゲット・サイトの構造と収集する必要のあるデータに一致する方法でクロール・プロセスをカスタマイズするために、オーバーライドする必要がある場合もあります。

getMaxLinks

このメソッドは、クロールするリンク数を制御するために使用されます。リンク数は正の整数である必要があります。そうでない場合、クロールは、開始 URI からアクセス可能な、同じドメイン内のすべてのリンクをスキャンします。

例 : 500 リンクのクロールを指定するには、次のようにします。

```
/**
 * default: -1; crawler will crawl over all the links
    reachable from the start URI
 * @return the maximum number of links to download.
 */
public int getMaxLinks()
{
     return 500;
}
```

getMaxCrawlDepth

このメソッドは、サイトをクロールする最大深度を制御するために使用されます。指定された深度を超えたリンクは無視されます。開始ページの深度は 0 です。

例 : クロール深度 4 を指定するには、次のようにします。

```
/**
 * default: -1. Indicates infinite depth for a site.
 * @return the maximum depth to which we need to crawl the
    links.
 */
 public int getMaxCrawlDepth()
 {
     return 4;
 }
```

getConnectionTimeout

このメソッドは、クローラがターゲット・サイトへの接続を確立するまで待機する時間を決定します。指定された時間内に接続が確立されない場合、クローラはそのリンクを無視して、次のリンクに進みます。

例 : 50,000 ミリ秒の接続タイムアウトを設定するには、次のようにします。

```
/**
 * default:30000 ms
 * @return Connection timeout in milliseconds.
 */
 public int getConnectionTimeout()
 {
     return 50000; // in milliseconds
}
```

getSocketTimeout

このメソッドは、クロール対象のリンクに対してクローラが行うリクエストのソケット・タイムアウトを制御します。

例 : 30,000 ミリ秒のソケット・タイムアウトを指定するには、次のようにします。

```
/**
 * default: 20000 ms
 * @return Socket timeout in milliseconds.
 */
public int getSocketTimeout()
{
    return 30000; // in milliseconds
}
```

getPostExecutionCommand

このメソッドは、カスタム post-crawl ロジックを注入するために使用されます。このメソッドは、クローラがそのクロール・セッションを終了するときに起動します。このメソッドによって、スクリプトまたはコマンドおよびパラメータの絶対パスが返される必要があります (ある場合)。

たとえば、`getPostExecutionCommand()` を使用して、クロール・セッションの終了後に静的にキャプチャされたファイルをコピーするためのバッチ・スクリプトまたはシェル・スクリプトを起動することにより、Web サーバーの doc ベースへのデプロイメントを自動化できます。

注意

- スクリプトまたはコマンドは、サイト・キャプチャをホストするすべてのサーバー上の同じ場所に存在する必要があります。
- サイト・キャプチャ・インタフェースからは (250MB を超える) 大きなアーカイブ・ファイルをダウンロードしないでください。`getPostExecutionCommand` を使用して、サイト・キャプチャ・ファイル・システムから希望の場所にファイルをコピーします。(アーカイブ・サイズは、「ジョブの詳細」画面のクローラ・レポートから取得できます。「ジョブの詳細」画面へのパスは、38 ページの図 2 に記載されています。)

例 : サイト・キャプチャ・サーバー上で copy.bat という名前のバッチ・スクリプトを実行するには、次のようにします。

```
/**
 * default:null.
 * @return the command string for post execution. Null if there
       is no such command.
*/
public String getPostExecutionCommand()
{
     // The file is supposed to be at the path C:¥¥commands
        folder
     // on the machine where the site capture server is running
     return "C:¥¥commands¥¥copy.bat";
}
```

getNumWorkers

このメソッドは、クロール・プロセスに使用されるワーカー・スレッド数を制御します。クロール・セッションに作成される理想的な並列スレッド数は、サイト・キャプチャをホストするマシンのアーキテクチャによって異なります。

例 : クロール・プロセスに対して 10 個のワーカー・スレッドを開始するには、次のようにします。

```
/**
 * default: 4.
 * @return the number of workers to start.
 * Workers will concurrently downloads resources.
 */
public int getNumWorkers()
{
     // Start 10 worker threads which is involved in the crawl
        process.
     return 10;
}
```

getUserAgent

このメソッドは、サイトを横断するときに、クローラで使用されるユーザー・エージェントを構成するために使用されます。ブラウザでレンダリングされる方法とは異なる方法でサイトがレンダリングされる (たとえば、サイトがモバイル・デバイス上でレンダリングされるなど) 場合は、このメソッドを使用する必要があります。

例 : FireFox 3.6.17 ユーザー・エージェントを構成するには、次のようにします。

```
/**
 * default: publish-crawler/1.1 (http://www.fatwire.com)
 * @return the user agent identifier
 */
public String getUserAgent()
{
     return "Mozilla/5.0 (Windows; U; Windows NT 6.1; en-
        US;rv:1.9.2.17) Gecko/20110420 Firefox/3.6.17 ";
}
```

createResourceRewriter

このメソッドは、クロールされる html ページ内の URL をリライトするために使用されます。たとえば、動的な WebCenter Sites Web サイトの静的配信を有効にするために URL をリライトする場合などです。

`createResourceRewriter` メソッドは、`ResourceRewriter` インタフェースのファクトリ・メソッドです。

- `ResourceRewriter` インタフェースを実装して、動的 URL を静的 URL に変換したり、絶対 URL を相対 URL に変換したりできます。
- 次のデフォルト実装も使用できます。
  - `NullResourceRewriter`: いずれの URL もリライトしません。
  - `PatternResourceRewriter`: 正規パターンを検索し、指定されているようにリライトします。

例 : `PatternResourceRewriter` を使用して、たとえば `http://www.site.com/home.html` のような URL を `/home.html` にリライトするには、次のようにします。

```
/**
 * Factory method for a ResourceRewriter.
 * default:new NullResourceRewriter();
 * @return the rewritten resource modifies the html before it
    is saved to disk.
*/
public ResourceRewriter createResourceRewriter()
{
     new PatternResourceRewriter("http://www.site.com/
        ([^¥¥s'¥"]*)", '/$1');
}
```

- デフォルト実装の詳細は、62 ページの「ResourceRewriter のデフォルト実装の使用方法」を参照してください。
- ResourceRewriter インタフェースの実装の詳細は、62 ページの「カスタム ResourceRewriter の記述」を参照してください。

createMailer

このメソッドは、クロールの終わりに電子メールを送信するための実装を提供します。createMailer メソッドは、Mailer インタフェースのファクトリ・メソッドです。

- サイト・キャプチャでは、SMTP over TLS の実装が提供されており、静的またはアーカイブ・キャプチャ・セッションの終了時にクローラ・レポートを電子メールで送信します (クローラ・レポートは、36 ページで説明されている report.txt ファイルです)。
- SMTP-TLS 以外のメール・サーバー (認証なしの SMTP や、POP3 など) を使用している場合は、独自の実装を提供する必要があります。

例 : 電子メールを送信しない場合は、次のようにします。

```
/**
 * Factory method for a Mailer.
 * <p/>
 * default:new NullMailer().
 * @return mailer holding configuration to send an email at
    the end of the crawl.
 * Should not be null.
 */
public Mailer createMailer()
{
     return new NullMailer();
}
```

- デフォルト実装の詳細は、66 ページの「Mailer のデフォルト実装の使用方法」を参照してください。
- ResourceRewriter インタフェースの実装の詳細は、68 ページの「カスタム・メーラーの記述」を参照してください。

getProxyHost

クロールされるサイトがプロキシ・サーバーの内側にある場合は、このメソッドはオーバーライドされる必要があります。このメソッド内でプロキシ・サーバーを構成できます。

注意

getProxyHost を使用する場合は、54 ページで説明されている getProxyCredentials も使用してください。

例 : プロキシ・サーバーを構成するには、次のようにします。

```
/**
 * default:null.
 * @return the host for the proxy,
 * null when there is no proxy needed
 */
public HttpHost getProxyHost()
{
     //using the HTTPClient library return a HTTPHost
     return new HttpHost("www.myproxyserver.com", 883);
}
```

getProxyCredentials

このメソッドは、`getProxyHost` メソッド (53 ページ) で構成されるプロキシ・サーバーの資格証明を注入するために使用されます。

例 : `sampleuser` という名前のプロキシ・サーバー・ユーザーを認証するには、次のようにします。

```
/**
 * default:null.
 * example: new UsernamePasswordCredentials(username,
    password);
 * @return user credentials for the proxy.
 */
public Credentials getProxyCredentials()
{
     return new UsernamePasswordCredentials("sampleuser",
        "samplepassword");
     //using the HTTPClient library return credentials
}
```

インタフェース

サイト・キャプチャには、デフォルト実装を持つ次のインタフェースがあります。

- LinkExtractor
- ResourceRewriter
- Mailer

LinkExtractor

リンク・エクストラクタは、クロール・セッションでサイト・キャプチャによって横断されるリンクを指定するために使用されます。実装は、`CrawlerConfigurator.groovy` ファイルを介して注入されます。この実装は、クロール・セッション時にサイト・キャプチャ・フレームワークによってコールされ、クロール・セッションの一部としてダウンロードされるマークアップからリンクを抽出します。

サイト・キャプチャでは、`LinkExtractor` の 1 つの実装が提供されています。独自のカスタム・リンク抽出ロジックを記述してデプロイすることもできます。詳細は、次の各項を参照してください。

- LinkExtractor インタフェース
- LinkExtractor のデフォルト実装の使用方法
- カスタム・リンク・エクストラクタの記述とデプロイ

LinkExtractor インタフェース

このインタフェースには、ダウンロード済マークアップからリンクを抽出するアルゴリズムを提供するために実装する必要がある唯一のメソッド (extract) があります。

```
package com.fatwire.crawler;

import java.util.List;

import com.fatwire.crawler.url.ResourceURL;


/**
 * Extracts the links out of a WebResource.
 *
 */

public interface LinkExtractor
{
     /**
      * Parses the WebResource and finds a list of links (if
            possible).
      *
      * @param resource the WebResource to inspect.
      * @return a list of links found inside the WebResource.
     */
     List<ResourceURL> extract(final WebResource resource);
     }
```

LinkExtractor のデフォルト実装の使用方法

PatternLinkExtractor は、LinkExtractor インタフェースのデフォルト実装です。PatternLinkExtractor は、正規表現に基づいてリンクを抽出します。正規表現を入力として使用し、その正規表現に一致するリンクのみを返します。

一般的な使用シナリオを下記に示します。それは次のとおりです。
動的 URL を持つサイトには PatternLinkExtractor を使用し、静的 URL を持つサイトには PatternLinkExtractor を使用します。

- 動的 URL を持つサイトに対する PatternLinkExtractor の使用方法 :

たとえば、www.fatwire.com では、リンクは /home、/support および /cs/Satellite/ のパターンを持ちます。このようなリンクの種類を抽出して横断するには、次のように PatternLinkExtractor を使用します。

```
/**
   * The method is used to define the link extraction
     algorithm from the crawled pages.
   * PatternLinkExtractor is a regex based extractor which
     parses the links on the web page
   * based on the pattern configured inside the constructor.
   *
   */
public LinkExtractor createLinkExtractor()
{
     return new PatternLinkExtractor("['¥"¥¥(](/
        [^¥¥s<'¥"¥¥)]*)",1);
}
```

パターン **['¥"¥¥(](/[^¥¥s<'¥"¥¥)]*)** を使用して、次のようなリンクを抽出します。

- 次の文字のいずれかで始まる
  - 一重引用符 (')
  - 二重引用符 (")
  - 左かっこ (
- スラッシュ (/) が続く
- 次の文字のいずれかで終わる
  - スペース (¥s)
  - 小なり記号 (<)
  - 一重引用符 (')
  - 二重引用符 (")
  - 右かっこ)

次のマークアップ内の URL を考えてみましょう。

```
<a href='/home'>Click Me</a>
```

関心があるのは /home リンクの抽出のみです。このリンクは、一重引用符 (') で始まり、一重引用符 (') で終わるため、正規表現パターンに一致します。1 のグループは、/home として結果を返します。

- 静的 URL を持つサイトに対する PatternLinkExtractor の使用方法 :

  たとえば、www.mysite.com のマークアップは、次のようなリンクを持ちます。

```
<a href="http://www.mysite.com/home/index.html">Click Me</a>"
```

このようなリンクの種類を抽出して横断するには、次のように PatternLinkExtractor を使用します。

```
/**
 * The method is used to define the link extraction algorithm
   from the crawled pages.
 * PatternLinkExtractor is a regex based extractor which
   parses the links on the web page
 * based on the pattern configured inside the constructor.
 *
 */
public LinkExtractor createLinkExtractor()
{
     return new PatternLinkExtractor(Pattern.compile("http://
        www.mysite.com/[^¥¥s<'¥"]*"));
}
```

前述の例では、クローラに http://www.mysite.com で始まり、次の文字のいずれかで終わるリンクを抽出するように指示します。
スペース (¥s)、小なり記号 (<)、一重引用符 (') または二重引用符 (")。

注意

グループおよびパターンの詳細は、Javadoc の Pattern および Matcher クラスを参照してください。

カスタム・リンク・エクストラクタの記述とデプロイ

注意

サイト・キャプチャは、WebCenter Sites の FirstSiteII 動的 Web サイトを静的サイトとしてダウンロードするために、FirstSiteII サンプル・クローラで使用されるサンプル・リンク・エクストラクタ (およびリソース・リライタ) を提供します。詳細は、次のフォルダにある FSIILinkExtractor クラスのソース・コードを参照してください。

```
<SC_INSTALL_DIR>/fw-site-capture/crawler/_sample/
  FirstSiteII/src
```

カスタム・リンク・エクストラクタを記述する手順は次のとおりです。

1. 目的の Java IDE にプロジェクトを作成します。
2. <SC_INSTALL_DIR>/fw-site-capture/webapps/ROOT/WEB-INF/lib フォルダからプロジェクトのビルド・パスにファイル fw-crawler-core.jar をコピーします。
3. extract() メソッドの実装を提供するために、LinkExtractor インタフェースを実装します。(LinkExtractor インタフェースは 56 ページに示されています。)

カスタム実装を表す疑似コードを次に示します。

```
package com.custom.crawler;

import java.util.List;

import com.fatwire.crawler.url.ResourceURL;
import com.fatwire.crawler.LinkExtractor;

/**
 * Extracts the links out of a WebResource.
 *
 */
public class CustomLinkExtractor implements LinkExtractor
     {
     /**
      * A sample constructor for CustomLinkExtractor
      */
     public CustomLinkExtractor(String .......)
     {
           // Initialize if there are private members.
           // User's custom logic
     }

     /**
      * Parses the WebResource and finds a list of links (if
          possible).
      *
      * @param resource the WebResource to inspect.
      * @return a list of links found inside the WebResource.
      */
     List<ResourceURL> extract(final WebResource resource)
     {
           // Your custom code for extraction Algorithm.
     }

}
```

4. カスタム実装用の jar ファイルを作成し、次のフォルダにコピーします。
   <SC_INSTALL_DIR>/fw-site-capture/webapps/ROOT/WEB-INF/lib
5. サイト・キャプチャ・アプリケーション・サーバーを再起動します。

6. カスタム・リンク・エクストラクタ・クラス (この例では CustomLinkExtractor) を含めるため、CrawlerConfigurator.groovy ファイルをコーディングすることにより依存関係を注入します。

```
/*
 * User's custom link extractor mechanism to extract the
   links from the
 * web resource downloaded as part of the crawl session.
 *
 * The code below is only a pseudo code for an example.User
   is free to implement
 * their own custom constructor as shown in the next
   example.
 */
public LinkExtractor createLinkExtractor()
{
     return new CustomLinkExtractor("Custom Logic For Your
        Constructor");
}
```

ResourceRewriter

リソース・リライタを使用して、クロール・セッション時にダウンロードされるマークアップ内の URL をリライトします。実装は、`CrawlerConfigurator.groovy` ファイルを介して注入する必要があります。

リソース・リライタを必要とするユースケースは次のとおりです。

- 動的サイトのクロールと静的コピーの作成。たとえば、FirstSiteII サンプル・サイトに動的リンクがあるとします。FirstSiteII を静的サイトに変換したら、ダウンロード済マークアップ内の URL をリライトする必要があります。
- 絶対 URL の相対 URL への変換。たとえば、マークアップに `http://www.mysite.com/abc.html` などの URL がある場合、クローラは URL から `http://www.mysite.com` を削除します。これにより、ダウンロード済ファイルが格納されているホストからリソースを提供することが可能になります。

サイト・キャプチャでは、`ResourceRewriter` の 2 つの実装が提供されています。カスタム実装を作成することもできます。詳細は、次の各項を参照してください。

- ResourceRewriter インタフェース
- ResourceRewriter のデフォルト実装の使用方法
- カスタム ResourceRewriter の記述

ResourceRewriter インタフェース

`rewrite` メソッドを使用して、クロール・セッション時にダウンロードされるマークアップ内の URL をリライトします。

```
package com.fatwire.crawler;

import java.io.IOException;


/**
 * Service for rewriting a resource. The crawler will use the
   implementation for
 * rewrite method to rewrite the resources that are downloaded as
   part of crawl
 * session.
 */
public interface ResourceRewriter
{

     /**
      * @param resource
      * @return the bytes after the rewrite.
      * @throws IOException
      */
     byte[] rewrite(WebResource resource) throws IOException;


}
```

ResourceRewriter のデフォルト実装の使用方法

サイト・キャプチャでは、ResourceRewriter の次の実装が提供されています。

- リンクのリライトをスキップするために、デフォルトで構成される NullResourceRewriter。CrawlerConfigurator.groovy ファイルで ResourceRewriter が構成されていない場合は、NullResourceRewriter がデフォルトで注入されます。
- 正規表現に基づいて URL をリライトするために使用される PatternResourceRewriter。PatternResourceRewriter は、マークアップ内のリンクと一致させるため正規表現を入力として使用し、これらのリンクを、コンストラクタ内で提供される文字列で置き換えます。

  例 : 絶対 URL を相対 URL としてリライトするには、次のようにします。

  元の URL:

  ```
  <a href="http://www.site.com/about/index.html">Click Me</a>
  ```

  リライト後の URL:

  ```
  <a href="/about/index.html">Click Me</a>
  ```

```
/**
 * Factory method for a ResourceRewriter.
 * default:new NullResourceRewriter();
 * @return the rewritten resource modifies the html before it
   is saved to disk.
*/
public ResourceRewriter createResourceRewriter()
{
     new PatternResourceRewriter("http://www.site.com/
        ([^¥¥s'¥"]*)", '/$1');
}
```

  PatternResourceRewriter には、正規表現と文字列の置換えを使用するコンストラクタが 1 つのみあります。

  ```
  PatternResourceRewriter(final String regex, final String
     replacement)
  ```

カスタム ResourceRewriter の記述

注意

サイト・キャプチャは、WebCenter Sites の FirstSiteII 動的 Web サイトを静的サイトとしてダウンロードするために、FirstSiteII サンプル・クローラで使用されるサンプル・リソース・リライタ (およびリンク・エクストラクタ) を提供します。詳細は、次のフォルダにある FSIILinkExtractor クラスのソース・コードを参照してください。

```
<SC_INSTALL_DIR>/fw-site-capture/crawler/_sample/
  FirstSiteII/src
```

カスタム・リソース・リライタを記述する手順は次のとおりです。

1. 目的の IDE 内にプロジェクトを作成します。
2. <SC_INSTALL_DIR>/fw-site-capture/webapps/ROOT/WEB-INF/lib フォルダからプロジェクトのビルド・パスにファイル fw-crawler-core.jar をコピーします。
3. rewrite メソッドの実装を提供するために、ResourceRewriter インタフェースを実装します。(ResourceRewriter インタフェースは 61 ページに示されています。)

   カスタム実装を表す疑似コードを次に示します。

```
package com.custom.crawler;

import com.fatwire.crawler.WebResource;
import com.fatwire.crawler.ResourceRewriter;


/**
 * Rewrite the links inside the markup downloaded as part of
 * crawl session.
 */
public class CustomResourceRewriter implements
     ResourceRewriter
{
     /**
      * A sample constructor for CustomResourceRewriter
      */
     public CustomResourceRewriter(String .......)
     {
           // Initialize if there are private members.
           // User's custom logic
     }

     /**
      * @param resource
      * @return the bytes after the rewrite.
      * @throws IOException
      */
     byte[] rewrite(WebResource resource) throws IOException
     {
           // Your custom code for re-writing Algorithm.
     }

}
```

4. カスタム実装用の jar ファイルを作成し、次のフォルダにコピーします。
   <SC_INSTALL_DIR>/fw-site-capture/webapps/ROOT/WEB-INF
   /lib

5. サイト・キャプチャ・アプリケーション・サーバーを再起動します。
6. カスタム resource rewriter クラス (この例では CustomResourceRewriter) を含めるため、CrawlerConfigurator.groovy ファイルをコーディングすることにより依存関係を注入します。

```
/*
 * User's custom resource rewriting mechanism to rewrite
   the links from the
 * web resource downloaded as part of the crawl session.
 *
 * The code below is only a pseudo code for an example.User
   is free to implement
 * their own custom constructor as shown in the next
   example.
 */
public ResourceRewriter createResourceRewriter()
{
     new CustomResourceRewriter("User's custom logic to
        initialize the things");
}
```

Mailer

メーラーを使用して、クローラが終了した後で電子メールを送信します。実装は、CrawlerConfigurator.groovy ファイルを介して注入する必要があります。

サイト・キャプチャは、SMTP-TLS メール・サーバーからクローラ・レポートを送信するために使用可能な SMTPTlsMailer 実装を提供します。また、Mailer インタフェースを実装して、SMTP-TLS 以外のサーバー (認証なしの SMTP や、POP3 など) から電子メールを送信するためのカスタム・ロジックを提供できます。カスタム・ロジックでは、電子メールがクローラ・レポート以外のオブジェクトであることを指定することもできます。CrawlerConfigurator.groovy ファイルで Mailer が構成されていない場合は、NullMailer がデフォルトで注入されます。詳細は、次のトピックを参照してください。

- Mailer インタフェース
- Mailer のデフォルト実装の使用方法
- カスタム・メーラーの記述

Mailer インタフェース

sendMail メソッドは、Mailer が CrawlerConfigurator.groovy ファイルで構成されている場合は自動的にコールされます。

```
package com.fatwire.crawler;

import java.io.IOException;

import javax.mail.MessagingException;


/**
 * Service to send an email.
 */
public interface Mailer
{
        /**
         * Sends the mail.
         *
         * @param subject
         * @param report
         * @throws MessagingException
         * @throws IOException
         */
        void sendMail(String subject, String report)
                throws MessagingException, IOException;
}
```

Mailer のデフォルト実装の使用方法

サイト・キャプチャでは、静的またはアーカイブ・クロール・セッションの終了時にクローラ・レポートを送信する、SMTP-TLS サーバー・ベースの電子メールの実装が提供されています (クローラ・レポートは、36 ページで説明されて

いる report.txt ファイルです)。デフォルト・メーラーは、次に示すように、CrawlerConfigurator.groovy ファイルを介して注入することで使用できます。

```
/**
 * Factory method for a Mailer.
 * <p/>
 * default:new NullMailer().
 * @return mailer holding configuration to send an email at the
   end of the crawl.
 * Should not be null.
 */
public Mailer createMailer()
{
     try
     {
        // Creating a SmtpTlsMailer Object
        SmtpTlsMailer mailer = new SmtpTlsMailer();

        InternetAddress from;
        // Creating an internet address from whom the mail should
           be sent from = new InternetAddress("from@gmail.com");

        // Setting the mail address inside the mailer object
           mailer.setFrom(from);

        // Setting the email address of the recipient inside
           mailer.mailer.setTo(InternetAddress.parse("xxxxxx@xxx
           xx.com"));

        // Setting the email server host for to be used for
           email.
        // The email server should be SMTP-TLS enabled.
        mailer.setHost("smtp.gmail.com", 587);

        // Setting the credentials of the mail account
           mailer.setCredentials("from@gmail.com",
           "frompassword");

        return mailer;
      }
     catch (AddressException e)
     {
        log.error(e.getMessage());
     }
}
```

カスタム・メーラーの記述

カスタム・メーラーを記述する手順は次のとおりです。

1. 目的の IDE 内にプロジェクトを作成します。
2. <SC_INSTALL_DIR>/fw-site-capture/webapps/ROOT/WEB-INF/lib フォルダからプロジェクトのビルド・パスにファイル fw-crawler-core.jar をコピーします。
3. Mailer インタフェースを sendMail メソッドを使用して実装します。

   カスタム実装を表す疑似コードを次に示します。

```
package com.custom.crawler;

import java.io.IOException;
import javax.mail.MessagingException;
import com.fatwire.crawler.Mailer;


/**
 * Implements an interface to implement the logic for sending
   emails
 * when the crawl session has been completed.
 */
public class CustomMailer implements Mailer
{
        /**
        * A sample constructor for CustomMailer
        */
        public CustomMailer()
        {
           // Initialize if there are private members.
           // User's custom logic
        }

        /**
        * Sends the mail.
        *
        * @param subject
        * @param report
        * @throws MessagingException
        * @throws IOException
        */
        void sendMail(String subject, String report)
              throws MessagingException, IOException
        {
           // User's custom logic to send the emails.
        }

}
```

4. カスタム実装用の jar ファイルを作成し、次のフォルダにコピーします。
<SC_INSTALL_DIR>/fw-site-capture/webapps/ROOT/WEB-INF/lib
5. サイト・キャプチャ・アプリケーション・サーバーを再起動します。
6. カスタム・メーラー・クラス (この例では CustomMailer) を含めるため、CrawlerConfigurator.groovy ファイルをコーディングすることにより依存関係を注入します。

```
/**
 * Factory method for a Mailer.
 * <p/>
 * default:new NullMailer().
 * @return mailer holding configuration to send an email at
   the end of the crawl.
 * Should not be null.
 */
public Mailer createMailer()
{
        CustomMailer mailer = new CustomMailer();
        // Do some of the initilization stuffs
        return mailer;
}
package com.custom.crawler;

import java.io.IOException;
import javax.mail.MessagingException;
import com.fatwire.crawler.Mailer;


/**
 * Implements an interface to implement the logic for sending
   emails
 * when the crawl session has been completed.
 */
public class CustomMailer implements Mailer
{
        /**
         * A sample constructor for CustomMailer
         */
        public CustomMailer()
        {
           // Initialize if there are private members.
           // User's custom logic
        }
/**
```

```
         * Sends the mail.
         *
         * @param subject
         * @param report
         * @throws MessagingException
         * @throws IOException
         */
         void sendMail(String subject, String report)
                     throws MessagingException, IOException
         {
            // User's custom logic to send the emails.
         }

}
```

前述の実装では、sendMail メソッドの String report 引数がクローラ・レポートを指定する場合、デフォルトで、クローラ・レポート (36 ページで説明されている report.txt ファイル) が電子メールで送信されます。ロジックをカスタマイズして、クローラ・レポート以外のオブジェクトを電子メールで送信できます。

要約

この章では、クローラのサイト・キャプチャ・プロセスを制御するための、サイト・キャプチャの BaseConfigurator クラスのメソッドとインタフェースについて説明しました。この項では、メソッドに加えて、インタフェースとそのデフォルト実装についてまとめています。

メソッド

- getStartUri
- createLinkExtractor
- getMaxLinks
- getMaxCrawlDepth
- getConnectionTimeout
- getSocketTimeout
- getPostExecutionCommand
- getNumWorkers
- getUserAgent
- createResourceRewriter
- createMailer
- getProxyHost
- getProxyCredentials

前述のリストでは、ファクトリ・メソッドは次のインタフェースにあります。

- createLinkExtractor は LinkExtractor インタフェースにあります。
- createResourceRewriter は ResourceRewriter インタフェースにあります。
- createMailer は Mailer インタフェースにあります。

インタフェース

- LinkExtractor
  デフォルト実装は `PatternLinkExtractor` です。これは、正規表現に基づいてリンクを抽出します。

  サイト・キャプチャは、WebCenter Sites の FirstSiteII 動的 Web サイトを静的サイトとしてダウンロードするために、FirstSiteII サンプル・クローラで使用されるサンプル・リンク・エクストラクタ (およびサンプル・リソース・リライタ) も提供します。ソース・コードは次のフォルダにあります。
  `<SC_INSTALL_DIR>/fw-site-capture/crawler/_sample/FirstSiteII/src`

  独自のカスタム・リンク抽出ロジックを記述してデプロイすることができます。
- ResourceRewriter
  デフォルト実装は、リンクのリライトをスキップする `NullResourceRewriter` と、正規表現に基づいて URL をリライトする `PatternResourceRewriter` です。

  サイト・キャプチャは、WebCenter Sites の FirstSiteII 動的 Web サイトを静的サイトとしてダウンロードするために、FirstSiteII サンプル・クローラで使用されるサンプル・リソース・リライタ (およびサンプル・リンク・エクストラクタ) を提供します。ソース・コードは次のフォルダにあります。
  `<SC_INSTALL_DIR>/fw-site-capture/crawler/_sample/FirstSiteII/src`

  URL をリライトするための独自のロジックを記述してデプロイすることができます。
- Mailer
  デフォルト実装は `SMTPTlsMailer` です。これは、SMTP-TLS メーラー・サーバーからクローラ・レポートを送信します。ロジックをカスタマイズして、他のタイプのサーバーから他のタイプのオブジェクトを電子メールで送信できます。