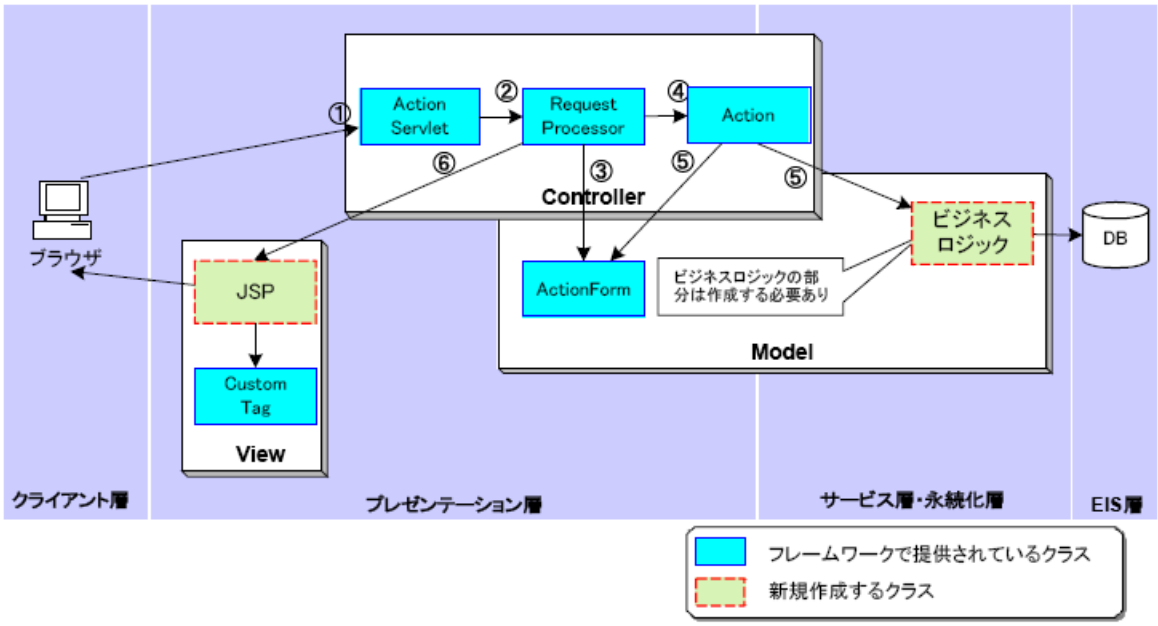
1.5 Strutsフレームワーク概略

Strutsフレームワーク概略

下図の各層は、アプリケーション設計構造の大枠となる論理的なレイヤである。本書では、以下の4層に区分する。

■ Strutsフレームワーク概略



以下の各層はJavaEEのアーキテクチャにもとづいたものである。

クライアント層	ブラウザが配置される。JavaScriptが実行される場合もある。
プレゼンテーション層	動的Webページ表示・入出力処理が配置される。
サービス層	業務処理が配置される。
永続化層	業務データ（主にデータベース）と連携した処理が配置される。
EIS(Enterprise Information System)層	業務データ（主にデータベース）が配置される。

Strutsは、Model-View-Controller（MVC）アーキテクチャを採用しており、仕様のライフサイクルが異なるビ ジネスロジック（Model）と処理制御（Controller）、画面（View）を分離することにより、画面などの仕様変更 に柔軟に対応できる構成となっている。

リクエスト処理フロー

図の番号にそってリクエスト処理フローを解説する。

1. ActionServlet クラスがクライアントからのリクエストを受け付け、現在のリクエストに対応するモジュールを選択する。
2. ActionServlet クラスは、RequestProcessor クラスからリクエストに対応したマッピング情報を持つActionMapping インスタンスを取得する。
3. RequestProcessor クラスは、2.で取得したマッピング情報をもとに、ActionForm インスタンスを取得しリクエストデータを設定する。
    - ActionForm インスタンスが生成されていない場合はインスタンスを生成し、生成済みの場合は再利用する。
    - データ検証は、ActionForm インスタンスに対してリクエストデータを設定した後に行われる。

4. RequestProcessor クラスは手順2で取得したマッピング情報をもとに、リクエストパスに対応したAction インスタンスを取得する。
    - Action インスタンスが生成されていない場合はインスタンスを生成し、生成済みの場合は再利用する。
5. RequestProcessor クラスが手順4で取得したAction クラスを呼び出し、対応するビジネスロジックを実行する。
   ビジネスロジック実行後、Action クラスは遷移先論理名をRequestProcessor クラスに返却する。
    - Action の呼び出しの際、RequestProcessor クラスは手順3で生成したActionFormインスタンス、手順2で取得したActionMapping インスタンスなどを引数としてAction クラスのexecute メソッドを呼び出す。
    - ビジネスロジックは実行結果として遷移先論理名をAction クラスに返却する。
    - Action クラスは実行結果として遷移先論理名を設定したActionForward インスタンスを生成し、RequestProcessor クラスに返却する。
6. RequestProcessor クラスは、ActionForward インスタンスとして取得した実行結果の遷移先論理名と、手順2で取得したマッピング情報をもとに遷移先画面のパスを取得し、画面をフォワードまたはリダイレクトを行い、クライアントに返す画面を生成、表示する。

(参考)フレームワークで規定しているクラス構造

以下に、Strutsで提供されている主なクラスを説明する。

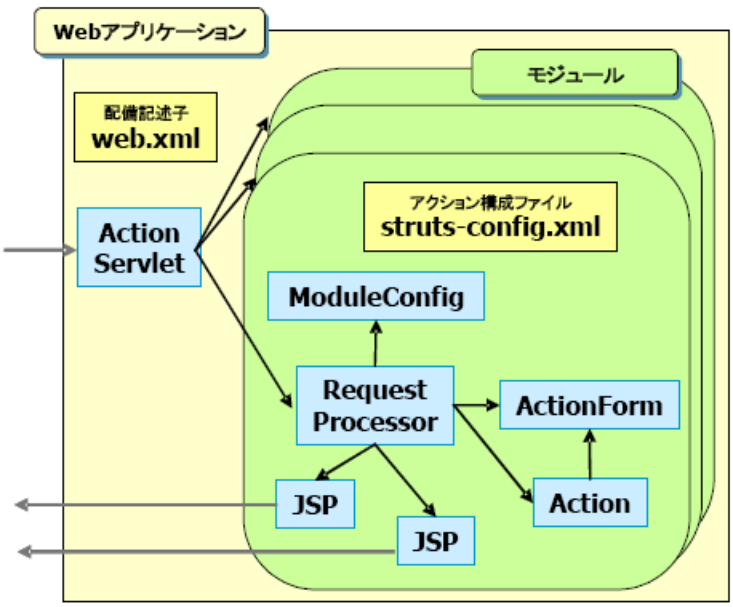
(1) ActionServlet

クライアントから来るリクエストの受付窓口であり、Java Servlet として（詳細にはHttpServlet クラスを継承して） 実装される。
このクラスではRequestProcessorの初期化や、リクエストを適切なRequestProcessorに振り分ける処理、各種リソースの制御などを行っている。

(2) RequestProcessor

RequestProcessorはリクエスト処理の中心となるクラスで、リクエストで指定されたURL とstruts設定ファイル（struts-config.xml）の設定内容に従って、 ActionFormへの入力データの格納、Actionの呼び出し、JSPへの遷移（フォワード、リダイレクト）などを行っている。 また、モジュール化を実現するためのクラスでもある。
モジュール化の概念を以下に示す。

■ モジュール化概念



モジュール化の核になるクラスで、各モジュールがRequestProcessorのインスタンスを1つ保有し、このインスタンスがモジュール内の処理制御を行う。 モジュール化により、分散開発が容易となった。

(3) ModuleConfig

struts設定ファイル（struts-config.xml）の定義情報のうち、モジュールに関する定義情報を保持するためのクラス。
ActionServletの初期化時にstruts設定ファイル（struts-config.xml）を読み込み、このクラスのインスタンスに定義情報を格納する。

(4) ActionMapping

struts設定ファイル（struts-config.xml）の定義情報のうち、Actionに関する定義情報を保持するためのクラス。
RequestProcessorの初期化時（ActionServletの初期化時に実行される）にstruts設定ファイル（struts-config.xml）を読み込み、 このクラスのインスタンスに情報を格納する。
ActionMappingのインスタンスはModuleConfig内のHashMapで管理され、リクエストで指定されたURLの“xxxx.do”の部分をキーに、 HashMapからActionMappingのインスタンスを検索し、そのインスタンスに格納されている情報をもとに処理の振り分けや遷移先の指定が行われる。

(5) ActionForm

HTMLのフォームに入力されたパラメータを格納するクラスであり、JavaBean として実装される。
入力パラメータに対して、入力チェックを行うことができる。Struts1.0系では入力されるデータの種類や組み合わせに 合わせて、あらかじめActionFormを継承してデータを格納するクラスを定義しておく必要があったが、 Struts1.1では入力されるデータの種類と組み合わせをstruts設定ファイル（struts-config.xml）に定義しておくことで、 クラスをあらかじめ定義しなくても入力データを格納できる機能が提供された。

(6) Action

業務ロジックを呼び出すクラスである。
開発者はこのクラスを継承して各Action クラスを設計する必要がある。 Action クラス自身に業務ロジックを持つことも可能であるが、Strutsでは推奨されておらず、通常は業務ロジック用のクラスを定義して、 そのインスタンスを呼び出す。業務ロジックを実行した戻り値を受け取って、遷移先情報としてActionForward クラスのインスタンスを返却する。
Action クラスの実装においては、スレッドセーフとなるよう注意する必要がある。

(7) ActionForward

遷移先と遷移の手順（フォワード、またはリダイレクト）の情報を格納するクラス。
Action クラスのインスタンスを呼び出した際の戻り値として使用され、RequestProcessorはActionForwardに格納されている情報にもとづいて画面遷移を実行する。

(8) JSP

動的にHTMLを生成するための仕組み。
JSPは最終的にはServlet クラスに変換され、コンパイルされることで動的にHTMLを生成する。

(9) Custom Tag

JSP内にHTML とプログラムが混在するとコードが複雑化し可読性も落ちるので、表の表示など共通的に使うものはタグハンドラクラスとして切り出して設計する。
アプリケーション開発者は必要に応じて拡張することができる。Strutsでは様々な種類のタグライブラリを以下の5種類に分けて提供している。

- struts-beanタグライブラリ
  - JavaBeansの定義、取得、内容表示機能を提供するタグライブラリ。
- struts-htmlタグライブラリ
  - HTMLフォームとStrutsのActionForm とを結びつける機能を提供するタグライブラリ。
- struts-logicタグライブラリ
  - 比較、存在チェック、繰り返し制御機能を提供するタグライブラリ。

- struts-nestedタグライブラリ
    - 階層化されたプロパティに効率的にアクセスする機能を提供するタグライブラリ。
- struts-tilesタグライブラリ
    - レイアウトとコンテンツを分離するための枠組みを提供するタグライブラリ。

備考

Strutsでは、データベースアクセスを制御するクラスや業務ロジックを定義するクラスは存在しない。 これらのクラスは、JavaBeansやEJBなどの技術を利用して、独自に定義する必要がある。 Server（Web）版を始めとするStruts拡張フレームワークの多くは、この点を補っているものが多い。

次節：1.6 Springフレームワーク概略