TERASOLUNA Server Framework for Java（Rich 版）

1. TERASOLUNA Server Framework for Java（Rich 版）とは

TERASOLUNA Server Framework for Java（Rich 版）（以降 TERASOLUNA（Rich 版））とは Spring Framework をベースにしたリッチクライアント・アプリケーションフレームワークです。

TERASOLUNA（Rich 版）には、以下の特徴があります。

- POJO（Plain Old Java Object）で開発を行うことによりテスト容易性が高まり、高品質を実現できます。

- 業務アプリケーションに必要な機能が詰まったフレームワークのため、ビジネスロジックの開発に集中できます。

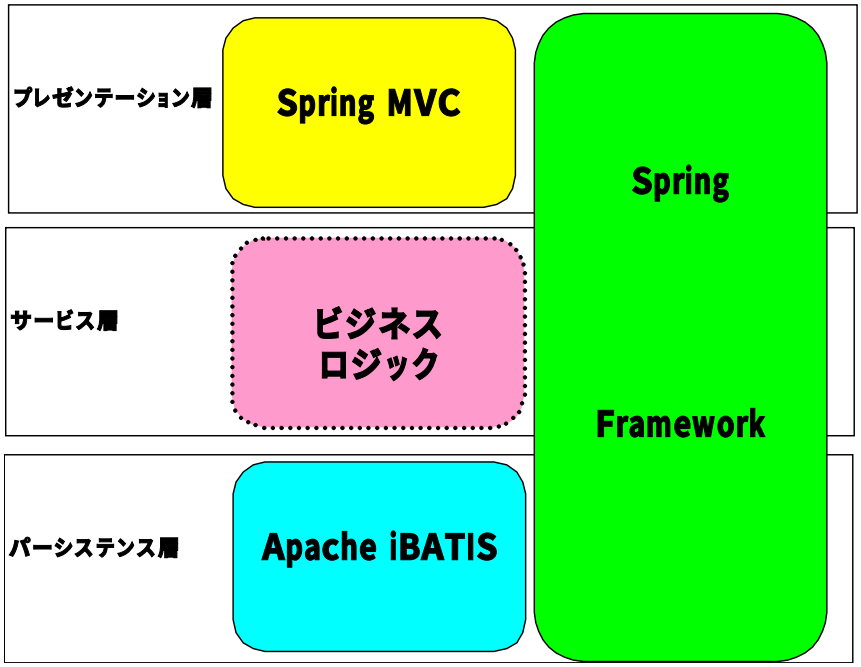
- XML 電文通信機能により、リッチクライアント・アプリケーション開発が容易になっています。

2. TERASOLUNA（Rich 版）の仕組み

TERASOLUNA（Rich 版）について、まずフレームワーク全体を説明し、その後際立った特徴について掘り下げていきます。

2.1. ベースとするフレームワークの選択

図 1　Rich 版　フレームワーク概略



TERASOLUNA（Rich 版）では 1.2 で挙げた特徴を実現するために、プレゼンテーション層は Spring MVC フレームワーク、パーシステンス層は Apache iBATIS を採用し、それらを統括的に管理する DI コンテナおよび横断的な処理を行うための AOP 機能として Spring Framework を選択しています。Spring Framework を利用することにより各コンポーネント間の依存性をなくすことができます。また、Spring AOP と iBATIS を使うことにより、簡潔な記述で DB にアクセスすることができ、トランザクション管理も宣言的に行え、オブジェクトの永続化処理を安全に行えます。TERASOLUNA Server Framework for Java（Web 版）（以降 TERASOLUNA（Web 版））と異なり、Spring MVC を利用することにより、コントローラのテストが可能になったり、アクションフォームに POJO を使用できたり、ビュー技術への依存度が低いため多様な View を選択できるなどの利点があります。これらのフレームワークを用いることにより、テスト容易性が高まり、その結果高品質を実現できます。

2.2. TERASOLUNA（Rich 版）のフレームワーク全体図

図 2　　フレームワーク全体図

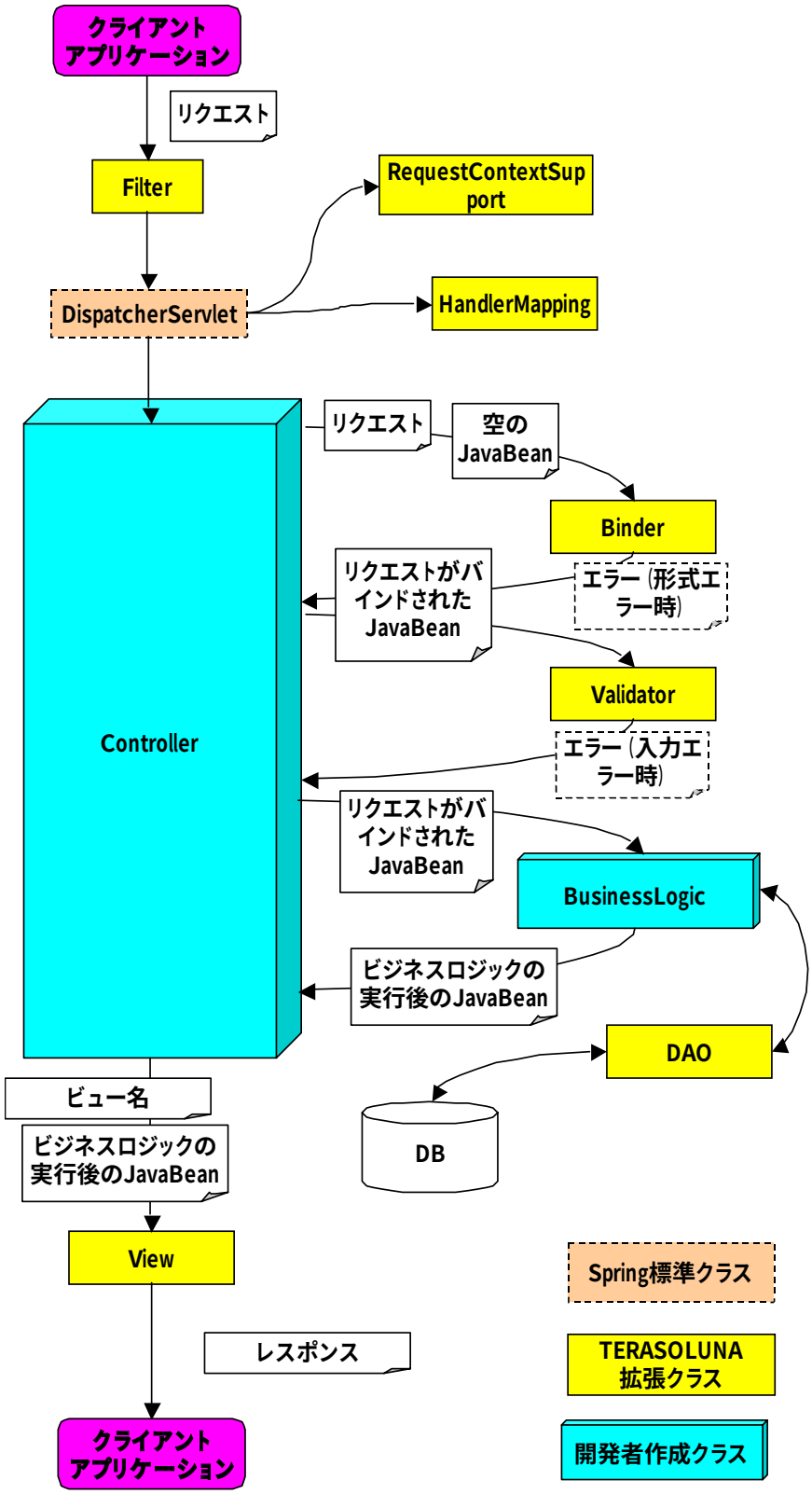


表 1　　各機能の説明

名称	実クラス	説明
Filter	jp.terasoluna.fw.web.rich.RequestContextHandlingFilter	フィルタクラス
Dispatcherservlet	org.springframework.web.servlet.DispatcherServlet	コントローラサーブレットリクエストに対する処理フロー全体を制御する
RequestContextSupport	jp.terasoluna.fw.web.rich.context.support.RequestContextSupport	制御情報を保持するためのクラス
HandlerMapping	jp.terasoluna.fw.web.rich.springmvc.servlet.handler.BeanNameUrlHandlerMappingEx	リクエストとコントローラーのマッピングを行うインタフェース
Binder	org.springframework.web.bind.ServletRequestDataBinder	クエリ形式のリクエストを JavaBean へバインドする
	org.springframework.web.bind.ServletRequestDataBinder	XML 形式のリクエストを JavaBean へバインドする
Validator	jp.terasoluna.fw.validation.springmodules.*	springmodules-validator を継承した入力チェッククラス
BusinessLogic	POJO	フレームワークに依存しない Plain Old Java Object
	jp.terasoluna.fw.service.rich.BLogic	BLogic インタフェースを継承したビジネスロジッククラス
DAO	jp.terasoluna.fw.dao.*	DAO インタフェース
View	jp.terasoluna.fw.web.rich.springmvc.servlet.view.castor.CastorView	Castor を利用して HTTP レスポンス生成するクラス
	jorg.springframework.web.servlet.view.velocity.VelocityView	Spring が提供する Velocity を利用して HTTP レスポンス生成するクラス
	jp.terasoluna.fw.web.rich.springmvc.servlet.view.filedownload.AbstractFileDownloadView	バイナリファイルをダウンロードする際に利用する View 抽象クラス。このクラスを継承して使用する。

フレームワークの全体図は図 2 および表 1 で示されるものになります。

2.3. リクエストデータ解析

リクエストデータ解析とは HTTP リクエストを JavaBean に変換する機能です。TERASORUNA（Rich 版）で用意している HTTP リクエストの形式にはクエリ形式と XML 形式の 2 通りがあります。

2.3.1. クエリ形式

org.springframework.web.bind.ServletRequestDataBinder を用いてクエリ形式の HTTP リクエストを JavaBean にバインドします。マルチパート形式のリクエストにも対応が可能です。

クエリ形式の例

```
param1=100
param2=2000
param3=30
```

2.3.2. XML 形式

org.springframework.web.bind.ServletRequestDataBinder を継承した jp.terasoluna.fw.web.rich.springmvc.bind.XMLServletRequestDataBinder を用いて XML 形式の HTTP リクエストを JavaBean にバインドします。実際の変換処理は jp.terasoluna.fw.oxm.mapper.OXMapper を実装した jp.terasoluna.fw.oxm.mapper.castor.CastorOXMapperImpl に委譲されます。クエリ形式とは異なり、形式チェックを行うことができます。XML スキーマによる形式チェック後、JavaBean に変換されます。

XML 形式の例

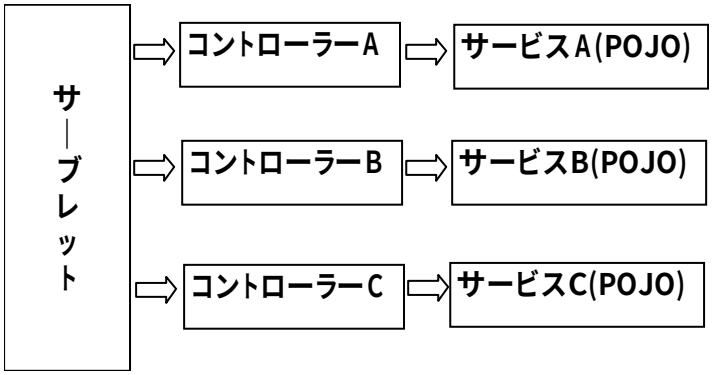
```
<sub-param>
  <param1>100</param1>
  <param1>2000</param1>
  <param1>30</param1>
</sub-param>
```

2.4. ビジネスロジック実行

ビジネスロジックの実装には POJO を用いる方法と、BLogic インタフェースを継承したクラスを用いる方法の 2 通りがあります。

2.4.1. POJO を用いる方法

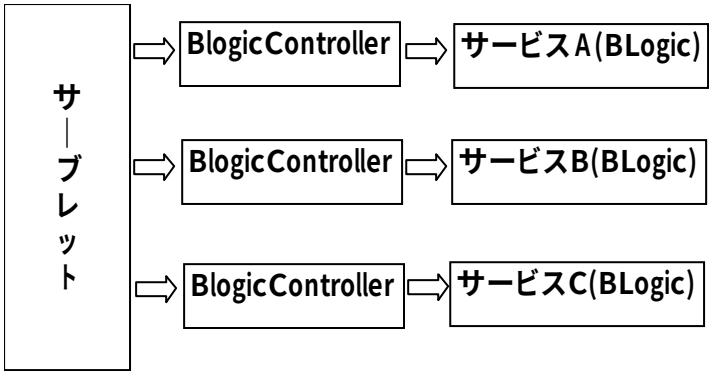
図 3　POJO を用いる方法



サービス（ビジネスロジック）を POJO にすることができます。そのため、疎結合なサービスになります。また、自由にインタフェース設計することが可能です。
しかし、上記の図で示すとおり、［サービス A］には［コントローラーA］、［サービス B］には［コントローラーB］のように、個々のサービスにそれぞれ専用のコントローラーが必要になります。既に POJO のビジネスロジックがある場合や仕様変更の確立が高い場合などを考慮したアーキテクチャといえます。

2.4.2. BLogic インタフェースを継承したクラスを用いる方法

図 4　BLogic インタフェースを継承したクラスを用いる方法



jp.terasoluna.fw.service.rich.BLogic インタフェースを継承したサービス（ビジネスロジック）クラスを作成します。前述の POJO を使用する場合と異なり、専用のコントローラーを新たに作成せずにすでに準備されている jp.terasoluna.fw.web.rich.springmvc.controller.BLogicController を利用するだけで実装可能になります。
しかし、BLogic に依存することになるため、サービスの再利用性が低下します。新規開発の場合や再利用を考慮するよりもスピードを重視する場合に向いているアーキテクチャになります。

2.5. レスポンスデータ作成

レスポンスデータ生成とはビジネスロジック実行結果後の JavaBean を HTTP レスポンスに変換する機能です。サーブレットは渡されたビュー名をもとにビューを作成し、その後 JavaBean をビューに渡します。HTTP レスポンスの形式には Castor ビュー、Velocity ビュー、ファイルダウンロードの 3 通りがあります。

2.5.1. Castor ビュー

JavaBean を XML 形式に変換してクライアントへ出力します。2.3.2 のリクエスト解析と同様に、jp.terasoluna.fw.oxm.mapper.OXMapper を実装した jp.terasoluna.fw.oxm.mapper.castor.CastorOXMapperImpl にて変換を行います。マッピング定義ファイルに基づく変換を行う方法と、フレームワークで自動変換する方法の 2 種類があります。マッピング定義ファイルは JavaBean ⇒ XML にも XML ⇒ JavaBean の双方向に使用できます。
しかし、後述する Velocity ビューに比べるとパフォーマンスの点で劣ります。

2.5.2. Velocity ビュー

JavaBean を XML、CSV、HTML など様々な形式に変換してクライアントへ出力します。前述の Castor ビューに比べてパフォーマンスが高い特徴があります。
しかし、生成される電文の妥当性が保障されないため、生成されるデータに対し妥当性検証を行う必要があります。

2.5.3. ファイルダウンロードビュー

jp.terasoluna.fw.web.rich.springmvc.servlet.view.filedownload.AbstractFileDownloadView を継承したクラスによって、JavaBean をバイナリファイル（入力ストリーム）に変換してクライアントに出力します。

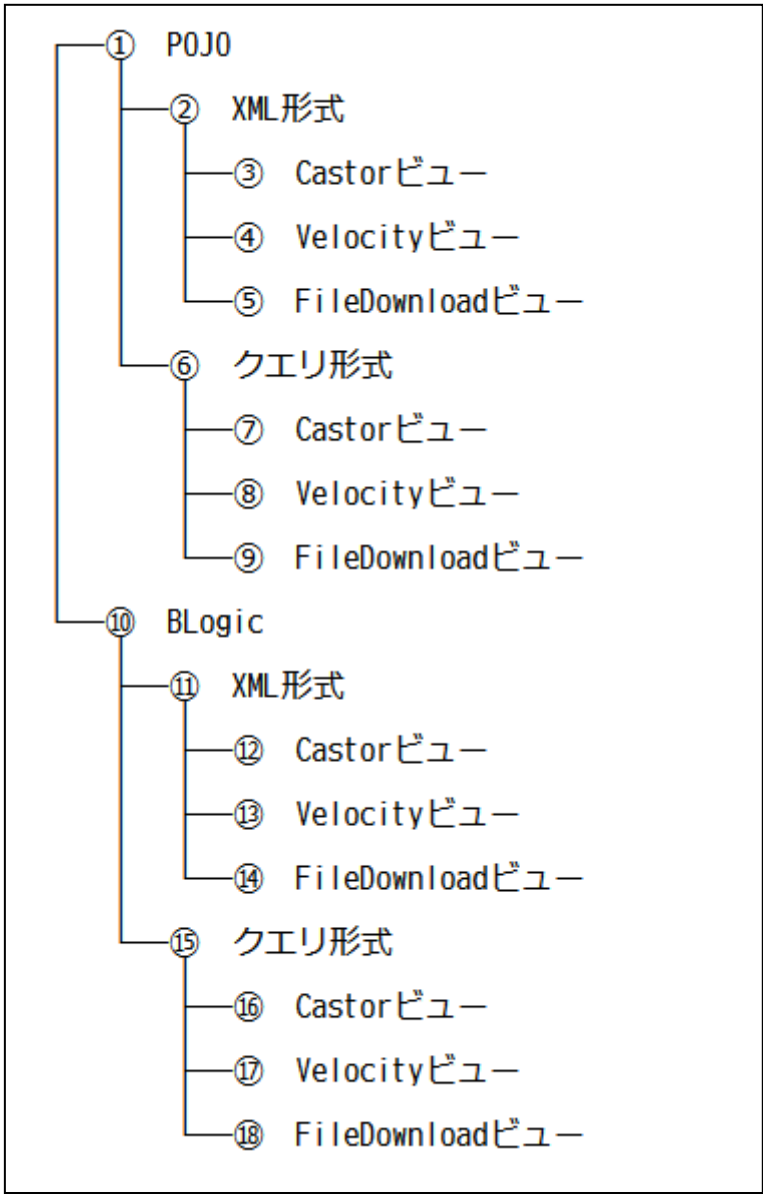
2.6. ビジネスロジックとリクエストとレスポンスの組み合わせ

TERASOLUNA（Rich 版）では 2.3～2.5 で説明したリクエスト・ビジネスロジック・ビューを組み合わせて作成します。ビジネスロジック（2 種類）X　リクエスト形式（2 種類）X　ビュー（3 種類）＝12 種類および、親 Bean 定義ファイル 6 種類の合計 18 種類のコントローラ抽象 Bean 定義が用意されており、それをもとに作成します。表 2 及び図 5 にその組み合わせを示します。

表 2 コントローラ抽象 Bean 定義

	Bean id	ビジネスロジック	リクエスト	ビュー
①	pojoController	POJO		
②	pojoXmlRequestController	POJO	XML 形式	
③	pojoXmlRequestCastorViewController	POJO	XML 形式	Castor
④	pojoXmlRequestVelocityViewController	POJO	XML 形式	Velocity
⑤	pojoXmlRequestDefaultFileDownloadViewController	POJO	XML 形式	FileDownload
⑥	pojoQueryRequestController	POJO	クエリ形式	
⑦	pojoQueryRequestCastorViewController	POJO	クエリ形式	Castor
⑧	pojoQueryRequestVelocityViewController	POJO	クエリ形式	Velocity
⑨	pojoQueryRequestDefaultFileDownloadViewController	POJO	クエリ形式	FileDownload
⑩	blogicController	BLogic		
⑪	blogicXmlRequestController	BLogic	XML 形式	
⑫	blogicXmlRequestCastorViewController	BLogic	XML 形式	Castor
⑬	blogicXmlRequestVelocityViewController	BLogic	XML 形式	Velocity
⑭	blogicXmlRequestDefaultFileDownloadViewController	BLogic	XML 形式	FileDownload
⑮	blogicQueryRequestController	BLogic	クエリ形式	
⑯	blogicQueryRequestCastorViewController	BLogic	クエリ形式	Castor
⑰	blogicQueryRequestVelocityViewController	BLogic	クエリ形式	Velocity
⑱	blogicQueryRequestDefaultFileDownloadViewController	BLogic	クエリ形式	FileDownload

図 5　　コントローラ抽象 Bean 定義　階層図



例えば、Blogic インターフェースを使用したビジネスロジックを使い、入力形式が XML 形式で出力に Castor ビューを使用する場合、SpringMVC の Bean 定義ファイル（サーブレット名-servlet.xml）に⑫の抽象コントローラ Bean 定義を記述します。そして、その親のコントローラ抽象 Bean 定義⑩・⑪も記述する必要があります。⑩・⑪・⑫の記述は以下のようになります。

```
<!-- コントローラ抽象 Bean 定義　（⑩ビジネスロジック：BLogic）　-->
<bean id="blogicController" abstract="true"
 class="jp.terasoluna.fw.web.rich.springmvc.controller.BLogicController">
  <property name="ctxSupport" ref="ctxSupport"/>
  <property name="validator" ref="beanValidator" />
</bean>

<!-- コントローラの抽象 Bean 定義
   （⑪ビジネスロジック：BLogic、受信リクエスト：XML 形式）　-->
<bean id="blogicXmlRequestController" abstract="true" parent="blogicController">
  <property name="dataBinderCreator" ref="xmlDataBinderCreator"/>
</bean>

<!-- コントローラの抽象 Bean 定義
   （⑫ビジネスロジック：BLogic、受信リクエスト：XML 形式、ビュー：Castor）-->
<bean  id="blogicXmlRequestCastorViewController"
 abstract="true" parent="blogicXmlRequestController"/>
```

また、コントローラ設定ファイル（サーブレット名-controller.xml）へのコントローラの定義とビジネスロジック設定ファイル（サーブレット名-businessLogic.xml）へのビジネスロジックの定義を記述する必要があります。

コントローラ設定ファイル（サーブレット名-controller.xml）は以下のように記述します。

```
<bean name="/sampleController"
  parent="blogicXmlRequestCastorViewController" scope="prototype">
  <property name="blogic" ref="sampleService" />
</bean>
```

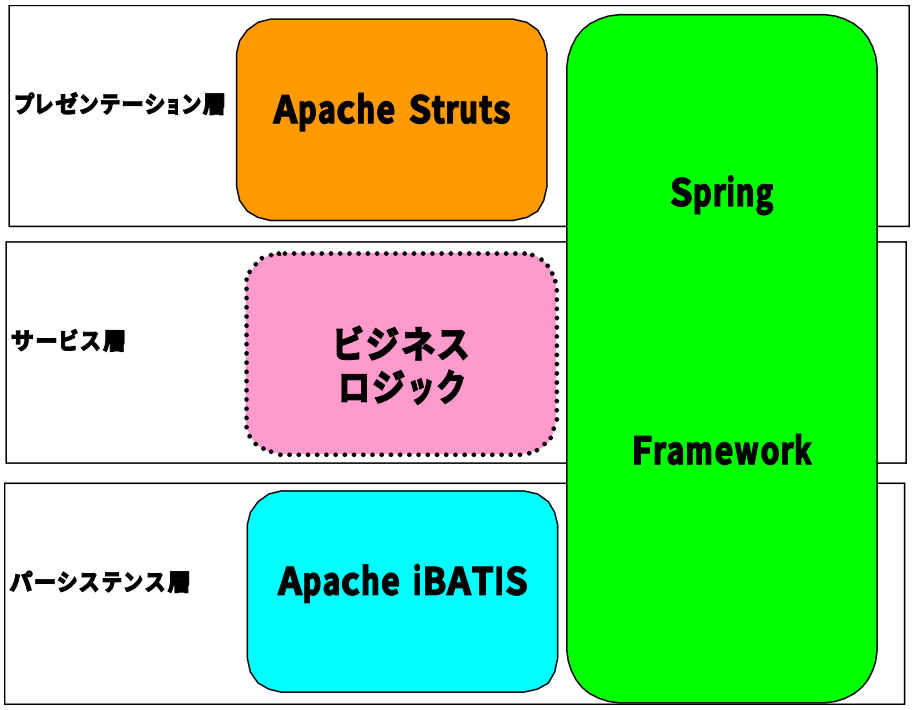
ビジネスロジック設定ファイル（サーブレット名-businessLogic.xml）は以下のように記述します。

```
<bean id="sampleService"
  class="jp.co.bbreak.terasoluna.rich.blogic.SampleBLogic"
  scope="prototype">
</bean>
```

3．TERASOLUNA（Rich 版）と TERASOLUNA（Web 版）との違い

TERASOLUNA フレームワークには TERASOLUNA（Rich 版）と同様な Java のサーバーサイドフレームワークである TERASOLUNA（Web 版）があります。では、いったい何が異なるのでしょうか？その説明を行います。

図 6　Web 版　フレームワーク概略



TERASOLUNA（Rich 版）と同様の Web アプリケーションフレームワークであり、プレゼンテーション層は Apache Struts フレームワーク、パーシステンス層は Apache iBATIS を採用し、それらを統括的に管理する DI コンテナおよび横断的な処理を行うための AOP 機能として Spring Framework を選択しています。TERASOLUNA（Rich 版）との違いはプレゼンテーション層に Spring MVC を使うのか、それとも Apache Struts を使うのかということになります。

表 3　　Rich 版と Web 版の比較

	TERASOLUNA（Rich 版）	TERASOLUNA（Web 版）
プレゼンテーション層	Spring MVC	Apache Struts
DI コンテナ	Spring Framework	
パーシステンス層	Aapche iBASIT	

この２つのフレームワークはともに Web アプリケーションのプレゼンテーション層を受け持つ MVC モデルのフレームワークです。MVC モデルは各モジュール（Model,View,Controller）が独立し、プログラムの可視性が高まり、保守性や仕様変更に強くなります。しかし、各モジュールの独立性に関しては Apache Struts と Spring MVC でも差異があります。Apache Struts はコントローラである Action クラスを継承しますが、Spring MVC はコントローラである Controller インターフェースを継承します。Strtus のモデルは ActionForm クラスを継承しなければなりませんが、Spring MVC ではモデルを POJO で扱うことができます。ビューの独立性も Struts に比べて Spring MVC は高いため、様々なビュー技術を簡単に使用することができます。つまり、Spring MVC は Struts に比べて、各モジュールの独立性が高いということになります。DI（Dependency Injection）の概念が表れる以前の MVC フレームワークである Struts と、Spring という DI コンテナの代表的なフレームワークが登場した後の MVC フレームワークである Spring MVC 違いとも言えるでしょう。

TERASOLUNA（Web 版）は Struts タグや独自のカスタムタグなどの UI コンポーネントが豊富であるため、UI を必要とするアプリケーションの開発に適しています。

TERASOLUNA（Rich 版）は Castor による XML 変換機能を利用し、UI を必要としない Ajax 等のリッチクライアントへのレスポンスを返すアプリケーションの開発に適しています。なお、Velocity ビューを使用することによって UI を必要とするアプリケーションを作成することも可能です。

4. まとめ

TERASOLUNA（Rich 版）は XML と Java のオブジェクトを自動的にマッピングする XML 電文通信機能により、リッチクライアント向けの入出力を行うアプリケーションを容易く開発できます。通常の Web アプリケーションにありがちなビジネスロジックとは関係ない作業から開放され、ビジネスロジックに注力することができます。
TERASOLUNA（Rich 版）は高い生産性で高品質かつ保守性があるリッチクライアントを開発できるフレームワークとなっています。

5.参考文献

- TERASOLUNA Server Framework for Java(Rich 版)　アーキテクチャ説明書
- TERASOLUNA Server Framework for Java(Rich 版)　機能説明書
- TERASOLUNA Server Framework for Java(Rich 版)　チュートリアル

以上の 3 つのドキュメントについては
http://sourceforge.jp/projects/terasoluna/releases/
より『terasoluna-server4jrich-doc_2.0.2.0.zip』をダウンロードすることにより取得できます。

- 『Spring 入門　より良い Web アプリケーションの設計と実装』
  著者：長谷川　裕一、伊藤　清人、岩永　寿来、大野　渉
  出版社：技術評論社
  ISBN：4-7741-2341-2
- 『Spring2.0 入門　Java・オープンソース・Web 開発自由自在』
  著者：長谷川　裕一、麻野　耕一、伊藤　清人、岩永　寿来、大野　渉
  出版社：技術評論社
  ISBN：4-7741-3000-1

開発部
荒川　正義