プレゼンテーション技術比較表

技術名称	ベース部分	開発効率	保守性	拡張性	将来性
JSP2.0 JSTL Taglibs	JSP	**評価**:○ タグファイル、タグライブラリによって、JSPの複雑な部分を分担できるようになった。	**評価**:△ JSP1.2に比べて飛躍的に保守性が向上した訳ではないが、JSTL、Taglibsなどの技術によって保守性を向上させる方法が増えたと言うべきか。	**評価**:○ 様々なフレームワーク、ツールなど組み合わせて使用できる。	**評価**:○ 最近、その他の技術に注目が集まっていますが、 JSP2.0も十分高機能な為、今後の拡張も期待でき、ドキュメントに関しても豊富である。
Velocity	VTL	**評価**:○ テンプレート部分（HTMLなど）をベースにデータ部や制御箇所を外部に持たすことによって、テンプレート部分をデザイナなどに委託できる。	**評価**:△ 評価を下し難いが、HTML部分をHTMLエディタで表示することができない為、レイアウト部分の保守性は高いとは言い難い。	**評価**:○ Strutsをはじめ様々なフレームワークがサポートしている為、JSPの対抗馬として拡張性も高い。	**評価**:○ StrutsやWebwork2などのMVCフレームワークとの相性の良さ、VTLの分かり易さから今後もサポートや拡張に期待できる。
Tapestry	独自の文法	**評価**:○ HTMLファイルにそのままjwc属性を指定してコンポーネントを割り当てる為、デザイナとプログラマの分担が非常に行いやすい。	**評価**:△ HTML形式の為、レイアウト部分の保守性は高いが、設定ファイルがページごとに増えていく為、保守が困難である。	**評価**:△ 独自性が高く、他のフレームワークなどとは組み合わせることはできない。	**評価**:△ EclipseのpluginとしてSpindleがあり、ドキュメントも充実しているが、拡張性のなさから将来的に広がりはないか。
Turbine	依存しない	**評価**:△ ビューに重きを置いたフレームワークで画面一つに対していくつかの制御クラスを割当てる為、細かく制御できるが、イメージが持ち難く、効率は良いとは言えない。	**評価**:△ 細かく設定する分、制御するクラスが1画面で多数存在してしまうことがある。	**評価**:○ ビューはデフォルトVelocityだが、JSPなども設定を変えるだけで使用できる。	**評価**:△ 過去にTorque（O/Rマッピング）とFulcrum（サービスフレームワーク）が存在したがそれぞれ独立した。それ以降あまり脚光は浴びていないか。
Cocoon	XSLT XSP XSLFO SVG WAP	**評価**:× XMLよりHTMLを生成する為、レイアウトイメージが分かりにくい。 また、XMLベースのアプリケーションである為、パフォーマンスが悪く、サーバのスペックも必要となります。	**評価**:△ XSLTで開発を行う場合は、ロジックを記述する必要は無いが、XSPを使用する場合はロジックを要する為、初期のJSPのようにビュー部分が複雑になる。	**評価**:△ Tomcatとの相性は良いみたいだが、Cocoon自体の設定が複雑であり、その他フレームワークとの併用については現在のところ考えられていないか。	**評価**:△ バージョン1から2に上がった際に設定の複雑さが多少解消されたようだが、まだまだ複雑な部分が多く、今後のバージョンでどこまで解消されるかになるだろう。
JSF	依存しない	**評価**:○ UIコンポーネントはPOJOのプロパティと紐付いており、それぞれにアクションを定義できるなど、幅広い画面設計とモデル部の相互性を提供しています。 また、GUIベースの開発が中心となる為、ツールの普及により飛躍的に開発効率があがることも想定される。 ただし、入力チェックに関して標準で提供している数が少ないという欠点もある。	**評価**:△ モデル⇔ビューの型変換を標準サポートしている為、モデル変更時も影響は少ない。 ただし、複雑な画面作成時にデメリットが多い為、そこが今後の課題になるか。	**評価**:○ Spring Webフレームワークが JSFをサポートしていたり、大手ベンダーの開発ツールがサポートしていることもあり、ツールも充実している。	**評価**:○ 大手ベンダーのツールサポートもあって、今後もツールやバージョンアップに期待がかかる。