# 13. Java に関する知識 II

## 1. 科目の概要

Java 言語がしばしば利用される Web アプリケーション開発を題材として、Java プログラミングの応用知識を解説する。EJB によるアプリケーションロジックの実装、JSP によるプレゼンテーションの実装、JDBC によるデータベースの操作方法など、Java を使用した Web アプリケーション開発の骨子を説明する。

## 2. 習得ポイント

本科目の学習により習得することが期待されるポイントは以下の通り。

| 習得ポイント | 説 明 | シラバスの対応コマ |
|---|---|---|
| II-13-8. オブジェクト指向分析 | オブジェクト指向システム分析とは何か説明し、オブジェクト指向システム開発作業の特徴、構成要素、オブジェクト指向開発で利用できるツールなどを紹介する。またJavaのフレームワークでどのように利用されているか、Javaアプリケーションにおけるオブジェクト指向の効果的な適用例について解説する。 | 13 |
| II-13-2. Javaを用いたWebサーバの実装 | Java言語により処理を行うWebサーバ、Webアプリケーションの実装方法について説明する。代表的なアーキテクチャであるServletを紹介し、構築ツールやプラットフォーム、構築上のポイントなど具体的なシステム設計手法について述べる。またデータベースの処理や構築についても触れる。 | 10 |
| II-13-3. Javaアプリケーション設計の手順 | Javaを用いたWebアプリケーションの設計について、機能分析・MVCモデルへの分割といった業務設計のフローやプレゼンテーション・ビジネスロジック・データベースに分割する3層アーキテクチャなど基本的な考え方を解説する。 | 11 |
| II-13-4. Javaアプリケーション開発の作業手順 | Javaアプリケーション開発の具体的な作業手順について、プレゼンテーション内容の設計方法、ビジネスロジックの設計方法、データベースの設計方法など各層における設計作業の手順とノウハウについて説明する。 | 11 |
| II-13-5. JSPによるプレゼンテーションの実装 | JSP (Java Server Pages)の基本と歴史、特徴について示し、JSPを利用したプレゼンテーションの実装について解説する。またJSPと関連する技術や対抗技術、その他、様々なプレゼンテーションの実現方法を紹介する。 | 12 |
| II-13-1. EJBによるアプリケーション開発の基本 | EJB (Enterprise Java Beans)を活用したアプリケーション開発の概要を説明する。EJBコンテナの仕組みやトランザクション処理の手順など、実際のプログラミング事例を示しつつ具体的な手法について解説する。 | 9 |
| II-13-6. EJBによるビジネスロジックの実装 | 3層アーキテクチャにおけるビジネスロジック部分の実装について解説する。具体的な例としてEJBによる実装やプログラミング例を示す。またEJB開発に利用できる各種の開発環境など、ビジネスロジック開発の効果的手法についても説明する。 | 12 |
| II-13-7. JDBCによるデータベースの操作 | JDBC (Java DataBase Connectivity)の基本と歴史、特徴について示し、JDBCを利用したデータベースの操作方法について解説する。JDBCによるデータベース操作の具体的な方法やプログラミング例を示し、データベース操作の基本手順を紹介する。 | 12 |
| II-13-9. デザインパターン | デザインパターンの基本的な考え方や歴史、特徴、デザインパターンの意義について解説する。さらにデザインパターンの効果的な使い方や代表的なパターンのいくつかを紹介し、Javaのフレームワークで利用されているデザインパターンの適用例を示す。 | 14 |
| II-13-10. Javaアプリケーションのチューニング | Webアプリケーションにおけるパフォーマンス向上の基礎を解説する。パフォーマンスチューニングを行う際の戦略やプロファイリングの手法、JVM自体のパラメータチューニング方法など、パフォーマンス向上策に関するポイントを紹介する。 | 15 |

【学習ガイダンスの使い方】

1. 「習得ポイント」により、当該科目で習得することが期待される概念・知識の全体像を把握する。
2. 「シラバス」、「IT 知識体系との対応関係」、「OSS モデルカリキュラム固有知識」をもとに、必要に応じて、従来の IT 教育プログラム等との相違を把握した上で、具体的な講義計画を考案する。
3. 習得ポイント毎の「学習の要点」と「解説」を参考にして、講義で使用する教材等を準備する。

# 3. IT 知識体系との対応関係

「13. Java に関する知識Ⅱ」と IT 知識体系との対応関係は以下の通り。

| 科目名 | 基本レベル（Ⅰ） | | | | | | | | 応用レベル（Ⅱ） | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 13. Javaに関するスキル | <Javaの基本> | <Java言語の基本構造> | <オブジェクト指向プログラミングのメリット> | <Javaによるアプリケーション開発手順> | <Javaによるネットワークプログラミング> | <Servlet/JSP/JDBCによるWebアプリケーション開発の概要> | <JDBCによるデータベースアクセス> | <MVCモデル> | <EJBによるアプリケーション開発> | <Javaによる Server処理実装の特徴と設計方法> | <Javaによる Webアプリケーションの設計／実装> | <Javaによる Server Side Java/Webアプリケーション実装> | <オブジェクト指向システム分析／設計／実装の実践技術> | <デザインパターンによる開発手順> | <Javaのパフォーマンスチューニング> |

[シラバス：http://www.ipa.go.jp/software/open/ossc/download/Model_Curriculum_05_13.pdf]

<IT 知識体系上の関連部分>

| 分野 | | 科目名 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 組織関連事項と情報システム | 1 | IT-IAS 情報保証と情報セキュリティ | IT-IAS1. 基礎的な問題 | IT-IAS2. 情報セキュリティの仕組み（対策） | IT-IAS3. 運用上の問題 | IT-IAS4. ポリシー | IT-IAS5. 攻撃 | IT-IAS6. 情報セキュリティ分野 | IT-IAS7. フォレンジック（情報証拠） | IT-IAS8. 情報の状態 | IT-IAS9. 情報セキュリティサービス | IT-IAS10. 脅威分析モデル | IT-IAS11. 脆弱性 | | |
| | 2 | IT-SP 社会的な観点とプロフェッショナルとしての課題 | IT-SP1. プロフェッショナルとしてのコミュニケーション | IT-SP2. コンピュータの歴史 | IT-SP3. コンピュータを取り巻く社会環境 | IT-SP4. チームワーク | IT-SP5. 知的財産権 | IT-SP6. コンピュータの法的問題 | IT-SP7. 組織の中のIT | IT-SP8. プロフェッショナルとしての倫理的な問題と責任 | IT-SP9. プライバシーと個人の自由 | | | | |
| 応用技術 | 3 | IT-IM 情報管理 | IT-IM1. 情報管理の概念と基礎 | IT-IM2. データベース問合わせ言語 | IT-IM3. データアーキテクチャ | IT-IM4. データモデリングとデータベース設計<br>[13-11] | IT-IM5. データと情報の管理<br>[13-6, 7]<br>[13-11, 13] | IT-IM6. データベースの応用分野 | | | | | | | |
| | 4 | IT-WS Webシステムとその技術 | IT-WS1. Web技術<br>[13-1, 6] | IT-WS2. 情報アーキテクチャ | IT-WS3. デジタルメディア | IT-WS4. Web開発 | IT-WS5. 脆弱性 | IT-WS6. ソーシャルソフトウェア | | | | | | | |
| ソフトウェアの方法と技術 | 5 | IT-PF プログラミング基礎 | IT-PF1. 基本データ構造 | IT-PF2. プログラミングの基本的構成要素 | IT-PF3. オブジェクト指向プログラミング<br>[13-3, 8]<br>[13-13] | IT-PF4. アルゴリズムと問題解決 | IT-PF5. イベント駆動プログラミング | IT-PF6. 再帰 | | | | | | | |
| | 6 | IT-IPT 技術を統合するためのプログラミング | IT-IPT1. システム間通信 | IT-IPT2. データ割り当てと交換 | IT-IPT3. 統合的コーディング<br>[13-3]<br>[13-11, 14] | IT-IPT4. スクリプティング手法 | IT-IPT5. ソフトウェアセキュリティの実現 | IT-IPT6. 種々の問題 | IT-IPT7. プログラミング言語の概要 | | | | | | |
| | 7 | CE-SWE ソフトウェア工学 | CE-SWE0. 歴史と概要 | CE-SWE1. ソフトウェアプロセス | CE-SWE2. ソフトウェアの要求と仕様 | CE-SWE3. ソフトウェアの設計 | CE-SWE4. ソフトウェアのテストと検証 | CE-SWE5. ソフトウェアの保守 | CE-SWE6. ソフトウェア開発・保守ツールと環境 | CE-SWE7. ソフトウェアプロジェクト管理 | CE-SWE8. 言語翻訳 | CE-SWE9. ソフトウェアのフォールトトレランス | CE-SWE10. ソフトウェアの構成管理 | CE-SWE11. ソフトウェアの標準化 | |
| | 8 | IT-SIA システムインテグレーションとアーキテクチャ | IT-SIA1. 要求仕様 | IT-SIA2. 調達/手配 | IT-SIA3. インテグレーション | IT-SIA4. プロジェクト管理 | IT-SIA5. テストと品質保証 | IT-SIA6. 組織の特性 | IT-SIA7. アーキテクチャ<br>[13-13] | | | | | | |
| システム基盤 | 9 | IT-NET ネットワーク | IT-NET1. ネットワークの基礎 | IT-NET2. ルーティングとスイッチング | IT-NET3. 物理層 | IT-NET4. セキュリティ | IT-NET5. アプリケーション分野 | IT-NET6. ネットワーク管理 | | | | | | | |
| | 10 | CE-NWK テレコミュニケーション | CE-NWK0. 歴史と概要<br>CE-NWK13. クラスタシステム | CE-NWK1. 通信ネットワークのアーキテクチャ<br>CE-NWK14. インターネットアプリケーション | CE-NWK2. 通信ネットワークのプロトコル<br>CE-NWK15. 次世代インターネット | CE-NWK3. LANとWAN<br>CE-NWK16. 放送 | CE-NWK4. クライアントサーバコンピューティング | CE-NWK5. データのセキュリティと整合性 | CE-NWK6. ワイヤレスコンピューティングとモバイルコンピューティング | CE-NWK7. データ通信 | CE-NWK8. 組込み機器向けネットワーク | CE-NWK9. 通信技術とネットワーク概要 | CE-NWK10. 性能評価 | CE-NWK11. ネットワーク管理 | CE-NWK12. 圧縮と伸張 |
| | 11 | IT-PT プラットフォーム技術 | IT-PT1. オペレーティングシステム | IT-PT2. アーキテクチャと機構 | IT-PT3. コンピュータインフラストラクチャ | IT-PT4. デプロイメントソフトウェア | IT-PT5. ファームウェア | IT-PT6. ハードウェア | | | | | | | |
| | 12 | CE-OPS オペレーティングシステム | CE-OPS0. 歴史と概要 | CE-OPS1. 並行性 | CE-OPS2. スケジューリングとディスパッチ | CE-OPS3. メモリ管理 | CE-OPS4. セキュリティと保護 | CE-OPS5. ファイル管理 | CE-OPS6. リアルタイムOS | CE-OPS7. OSの概要 | CE-OPS8. 設計の原則 | CE-OPS9. デバイス管理 | CE-OPS10. システム性能評価 | | |
| コンピュータアーキテクチャ | 13 | CE-CAO コンピュータのアーキテクチャと構成 | CE-CA00. 歴史と概要 | CE-CA01. コンピュータアーキテクチャの基礎 | CE-CA02. メモリシステムの構成とアーキテクチャ | CE-CA03. インタフェースと通信 | CE-CA04. デバイスサブシステム | CE-CA05. CPUアーキテクチャ | CE-CA06. 性能・コスト評価 | CE-CA07. 分散・並列処理 | CE-CA08. コンピュータによる計算 | CE-CA09. 性能向上 | | | |
| 複数領域にまたがるもの | 14 | IT-ITF IT基礎 | IT-ITF1. ITの一般的なテーマ | IT-ITF2. 組織の問題 | IT-ITF3. ITの歴史 | IT-ITF4. IT分野（学科）とそれに関連のある分野（学科） | IT-ITF5. 応用領域 | IT-ITF6. IT分野における数学と統計学の活用 | | | | | | | |
| | 15 | CE-ESY 組込みシステム | CE-ESY0. 歴史と概要<br>CE-ESY13. リアルタイムシステム設計 | CE-ESY1. 低電力コンピューティング<br>CE-ESY14. 組込みマイクロコントローラ | CE-ESY2. 高信頼性システムの設計<br>CE-ESY15. 組込みプログラム | CE-ESY3. 組込み用アーキテクチャ<br>CE-ESY16. 設計手法 | CE-ESY4. 開発環境<br>CE-ESY17. ツールによるサポート | CE-ESY5. ライフサイクル<br>CE-ESY18. ネットワーク型組込みシステム | CE-ESY6. 要件分析<br>CE-ESY19. インタフェースシステムと混合信号システム | CE-ESY7. 仕様定義<br>CE-ESY20. センサ技術 | CE-ESY8. 構造設計<br>CE-ESY21. デバイスドライバ | CE-ESY9. テスト<br>CE-ESY22. メンテナンス | CE-ESY10. プロジェクト管理<br>CE-ESY23. 専門システム | CE-ESY11. 並行設計（ハードウェア、ソフトウェア<br>CE-ESY24. 信頼性とフォールトトレランス | CE-ESY12. 実装 |

## 4. OSS モデルカリキュラム固有の知識

OSS モデルカリキュラム固有の知識として、Java 言語のフレームワークによる Web 開発の特徴、開発手法がある。EJB、Servlet による Web システムを OSS を用いて実際に構築する事例を通して、実践的な知識を習得する。

| 科目名 | 第9回 | 第10回 | 第11回 | 第12回 | 第13回 | 第14回 | 第15回 |
|---|---|---|---|---|---|---|---|
| 13.Java に関する知識Ⅱ | (1)EJB の概要<br>(2)トランザクション処理<br>(3)プログラミング事例 | (1)Java によるServer 処理の実装方法<br>(2)データベース処理<br>(3)構築実例の紹介 | (1)業務の設計<br>(2)Java 開発の具体的な作業項目 | (1)構築実習 | (1)開発作業の特徴<br>(2)処理仕様の記述<br>(3)コンポーネントの配置<br>(4)Java による実装<br>(5)システムアーキテクチャの検証 | (1)基本的なデザインパターンとその役割<br>(2)J2SE 標準ライブラリ、Tomcat、Struts フレー<br>(3)デザインパターンの適用例 | (1)パフォーマンスチューニングの戦略<br>(2)プロファイリングの手法<br>(3)JVM のチューニング |

（網掛け部分は IT 知識体系で学習できる知識を示し、それ以外は OSS モデルカリキュラム固有の知識を示している）

| スキル区分 | OSS モデルカリキュラムの科目 | レベル |
|---|---|---|
| プログラミング分野 | 13　Java に関する知識 | 応用 |
| 習得ポイント | II -13-1. オブジェクト指向分析 | |
| 対応する<br>コースウェア | 第 13 回 (オブジェクト指向システム分析／設計／実装の実践技術) | |

# II -13-1. オブジェクト指向分析

オブジェクト指向システム分析とは何か説明し、オブジェクト指向システム開発作業の特徴、構成要素、オブジェクト指向開発で利用できるツールなどを紹介する。またJavaのフレームワークでどのように利用されているか、Java アプリケーションにおけるオブジェクト指向の効果的な適用例について解説する。

【学習の要点】

* オブジェクト指向分析は、システムを相互に作用しあうオブジェクトの集まりとしてモデル化することで、システムが何を行うのか、を明確にすることを目的とする手順である。分析段階では、どのように行うのか、は考慮しない。
* オブジェクト指向分析は、業務領域に存在する情報をオブジェクトとして抽出する作業と、オブジェクト同士の静的な関連を定義した静的モデル、および一連の処理の流れの中で行われるオブジェクト同士の相互作用を定義した動的モデルを洗練していく作業を通して行われる。
* オブジェクト指向分析の結果は UML を用いたダイアグラムとユースケース記述によって表現されることが多い。

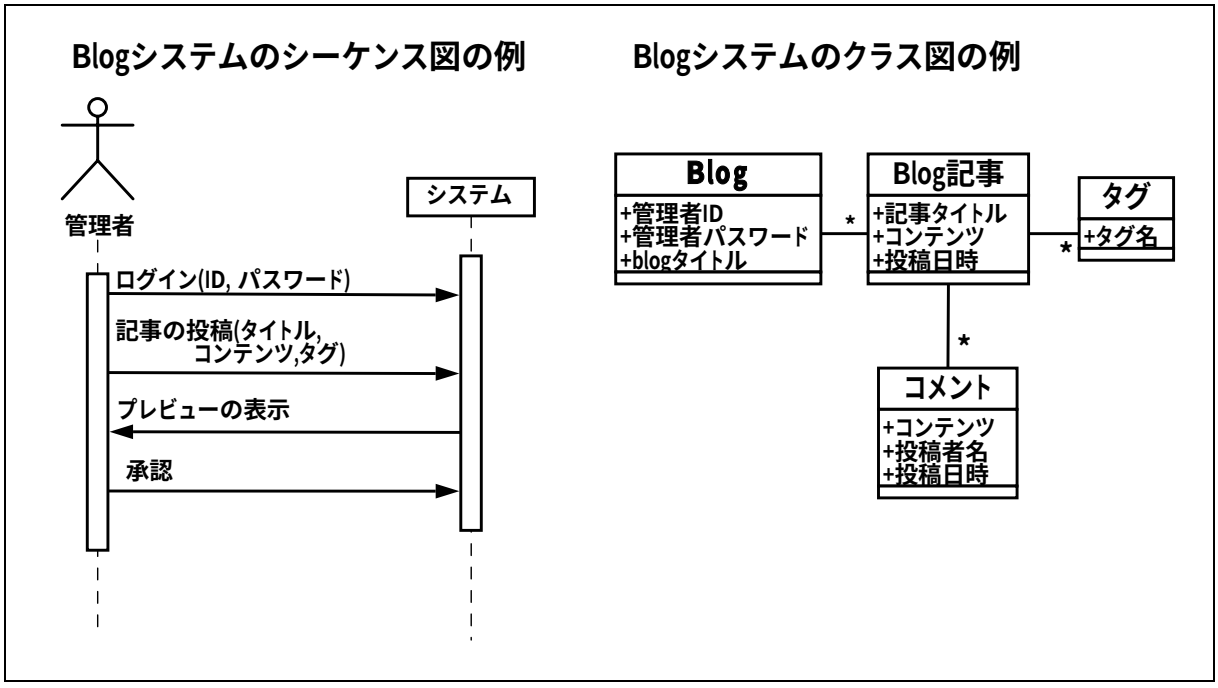


図 II-13-1.　UML によるオブジェクト指向分析

【解説】

1) オブジェクト指向分析

オブジェクト指向分析は、要求分析の手法であり、システムを相互に作用しあうオブジェクトの集まりとしてモデル化(モデリング)することで、システムが何を行うのか、を明確にすることを目的とする。分析段階では、どのように行うのか、は考慮しない。オブジェクト指向分析におけるモデリングでは、システムの静的側面と動的側面を対象にする。

* 動的な振る舞いのモデリング
  動的側面のモデリングでは、ユーザの目的をシステムが実現するまでの流れを時間軸に沿ったやりとりとして表現する。
  - アクタとユースケースの抽出
    アクタとはシステムの外側に位置し、システムに要求を出す役割を持つ要素であり、多くの場合システムの利用者である。ユースケースは、初期段階では、アクタがシステムに対して行う要求、操作を記述したものである。アクタとユースケースの関係は UML のユースケース図を用いて表す。
  - ユースケースの詳細化
    得られたユースケース毎に、時間軸に沿ったシステムとアクタの相互作用、および条件による分岐について詳細化を行う。時間軸に沿った相互作用は UML のシーケンス図で表現できる。条件による分岐は UML のアクティビティ図で表現できる。また、詳細化したユースケースを、適切なフォーマットで文書化したものをユースケース記述と呼ぶ。
* 静的な構造のモデリング
  静的側面のモデリングは、オブジェクトの抽出とオブジェクト同士の関連の定義を通して行い、結果は UML のクラス図で表される。
  - オブジェクトの抽出
    システム化の対象となる情報をグループ化し、オブジェクトとして抽出する。グループは、グループ自体を表す名前と、グループの性質を表現する属性の形式にまとめ、それぞれオブジェクト名、属性名とする。グループ化は実体や概念の単位で行うが、明確な基準があるわけではなく、システムのユーザ、開発者双方にとっての分かり易さを重視して行う。
  - オブジェクト同士の関連の定義
    抽出したオブジェクト同士の参照関係を元に関連、および多重度を定義する。このとき、参照のキーとなる属性を設定したり、重複した情報の整理などを行うことも可能であるが、データベーステーブルの設計を行う場合とは異なり、必須ではない。オブジェクトの抽出の手順と同様、分かり易さを重視する。

動的モデリングの過程で、静的モデルに含まれていない情報や関連が現れた場合には、静的モデルに反映させる。逆に動的モデルに登場しない情報が静的モデルに含まれていた場合には、その情報がシステム化の範囲に含まれるかどうかについて検討を行う。

2) オブジェクト指向分析で利用できるツール

UML の作成を支援するツールとして、ArgoUML や Umbrello などが OSS として公開されている。

3) その他の分析事項

オブジェクト指向分析では扱わないものの別途決める必要のある事項として、応答時間などの非機能要件、および画面遷移、画面レイアウトといったユーザインタフェースの仕様が挙げられる。

| スキル区分 | OSS モデルカリキュラムの科目 | レベル |
|---|---|---|
| プログラミング分野 | 13　Java に関する知識 II | 応用 |
| 習得ポイント | II -13-2. Java を用いた Web サーバの実装 | |
| 対応する<br>コースウェア | 第 10 回 (Java による Server 処理実装の特徴と設計方法) | |

# II -13-2. Java を用いた Web サーバの実装

Java 言語により処理を行う Web サーバ、Web アプリケーションの実装方法について説明する。代表的なアーキテクチャである Servlet を紹介し、構築ツールやプラットフォーム、構築上のポイントなど具体的なシステム設計手法について述べる。またデータベースの処理や構築についても触れる。

【学習の要点】

* Java 言語を用いて HTTP リクエストを処理するサーバを実装する手段として、Servlet を利用することができる。
* Java 言語を用いた Web アプリケーションにおいて、Servlet は単に動的な HTML を生成するだけでなく、JSP や EJB コンテナに処理を振り分ける、MVC (Ⅱ-13-3 参照)におけるコントローラの役割を担うことが多い。
* Java 言語を用いてデータベース接続を行う API として JDBC (Java Database Connectivity) を利用することができる。

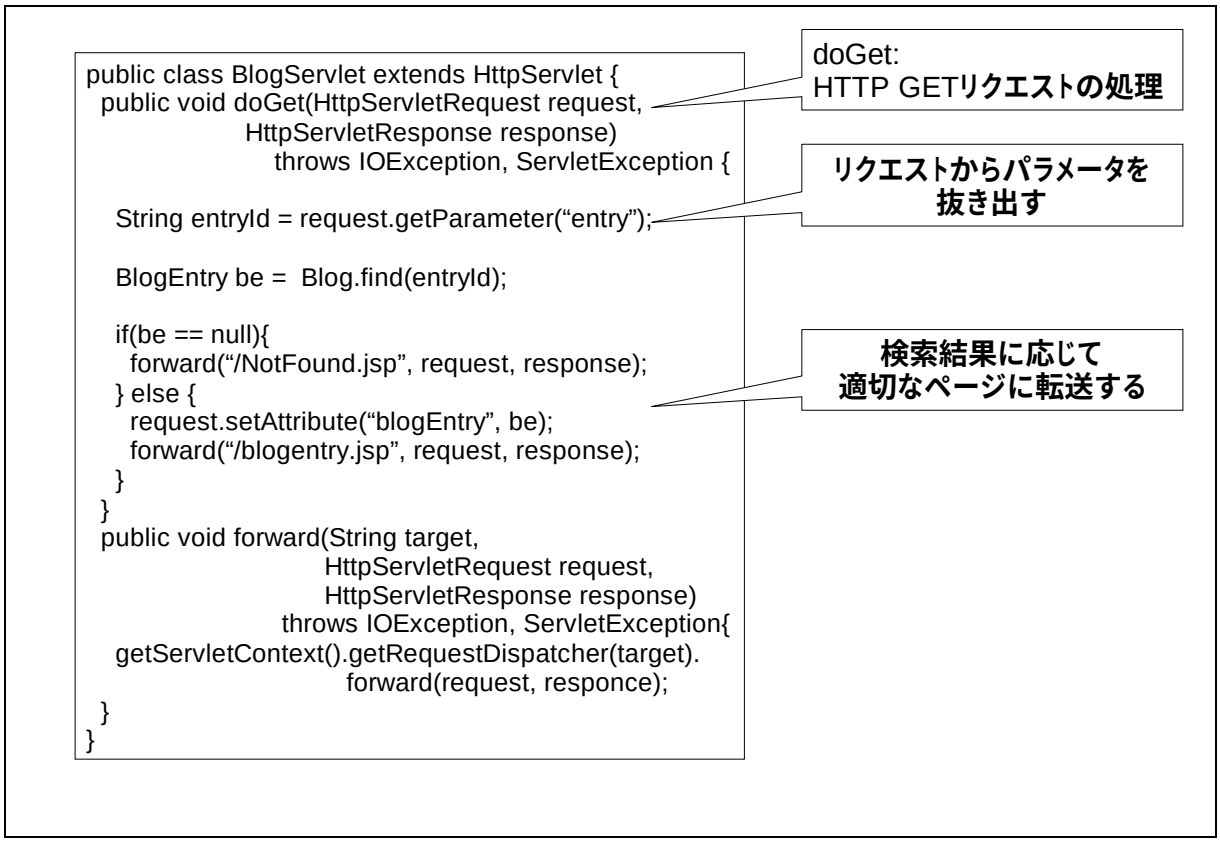


図 II-13-2.　Servlet のソースコード例

【解説】

1) Servlet とは

Servlet とは、Java 言語を用いて Web サーバの機能を拡張し、動的な Web ページの生成を可能にするプログラム、および仕様であり、Java EE 仕様の一部となっている。Servlet 登場以前は、動的な Web ページの生成は主に CGI によって実現されていた。CGI がリクエスト毎に新たなプロセスを起動するのに対し、Servlet は単一のプロセス内で複数リクエストの処理を行うために、リクエストあたりの負担が少ない、というメリットがある。

2) Servlet のプラットフォーム

Servlet、および JSP の実行環境を提供するサーバソフトウェアをサーブレットコンテナとよび、OSS のサーブレットコンテナとして Tomcat などが公開されている。

3) Servlet の機能

* Servlet は GET、POST といった HTTP のメソッドに対応した Java メソッドを実装することで、任意の HTTP リクエストを処理することができる。Servlet で行う処理としては、HTML を組み立てることで動的なページを生成するほか、サーバ内のファイルシステムへのアクセスや、データベースに対する検索、保存など、Java 言語を用いた任意の処理を行うことが可能である。
* Servlet は、異なるページの間での同一クライアントの認識や、クライアント毎の情報管理を行うための Session 管理機能を提供し、Cookie 単体では難しい高度なユーザ情報管理を容易に実現できる。

4) Servlet の役割

* Web アプリケーションの全ての機能を Servlet により実装することも可能であるが、デザインとロジックが分離されないなど、保守性に問題が生じる可能性がある。そのため、今日の Web アプリケーションでは、Servlet は MVC アーキテクチャにおける Controller の役割を担い、View は JSP、Model は JavaBeans、といったように、より適した技術を併用することが多い。
* 多くの Web アプリケーションフレームワークでは、MVC アーキテクチャの Controller としての Servlet の実装はフレームワークから提供され、開発者は XML などの設定ファイルに制御情報を記述する場合が多い。このようなフレームワークの例としては Apache Struts や JavaServer Faces が挙げられる。

5) データベース接続

Java EE 環境では、データベース接続方法として、JNDI を利用して DataSource オブジェクトを取得し、DataSource オブジェクトからデータベース接続オブジェクトを取得する方法が推奨されている。JNDI（Java Naming and Directory Interface)を用いることで接続情報が一元化され、Web アプリケーションの保守性が向上する。また、DataSource を利用することで、コンテナの提供するコネクションプーリング機能などを利用することが可能になる。

| スキル区分 | OSS モデルカリキュラムの科目 | レベル |
|---|---|---|
| プログラミング分野 | 13　Java に関する知識 II | 応用 |
| 習得ポイント | II -13-3. Java アプリケーション設計の手順 | |
| 対応する<br>コースウェア | 第 11 回 (Java による Web アプリケーションの設計／実装) | |

# II -13-3. Java アプリケーション設計の手順

Javaを用いたWebアプリケーションの設計について、機能分析･MVCモデルへの分割といった業務設計のフローやプレゼンテーション･ビジネスロジック･データベースに分割する 3 層アーキテクチャなど基本的な考え方を解説する。

【学習の要点】

* オブジェクト指向分析で得られた成果物をもとにモジュール設計を行う。
* Java を用いた Web アプリケーションの設計に MVC を採用する場合、ビジネスロジックとデータを扱う機能を Model、画面表示機能を View、リクエストに応じて Model と View に処理の振り分けを行う機能を Controller に分割する。
* Java を用いた Web アプリケーションに対し MVC モデルを採用する場合の代表的な設計例として、Model を EJB もしくは JavaBeans で実装し、View を JSP で実装し、Controller を Servlet で実装する設計が挙げられる。
* Java を用いた Web アプリケーションの設計に 3 層アーキテクチャを採用する場合、MVC における View と Controller の機能をプレゼンテーション層、Model のうちビジネスロジックを扱うモジュールをビジネスロジック層、Model のうちデータベースにアクセスするモジュールをデータベース(データアクセス)層に分離する。

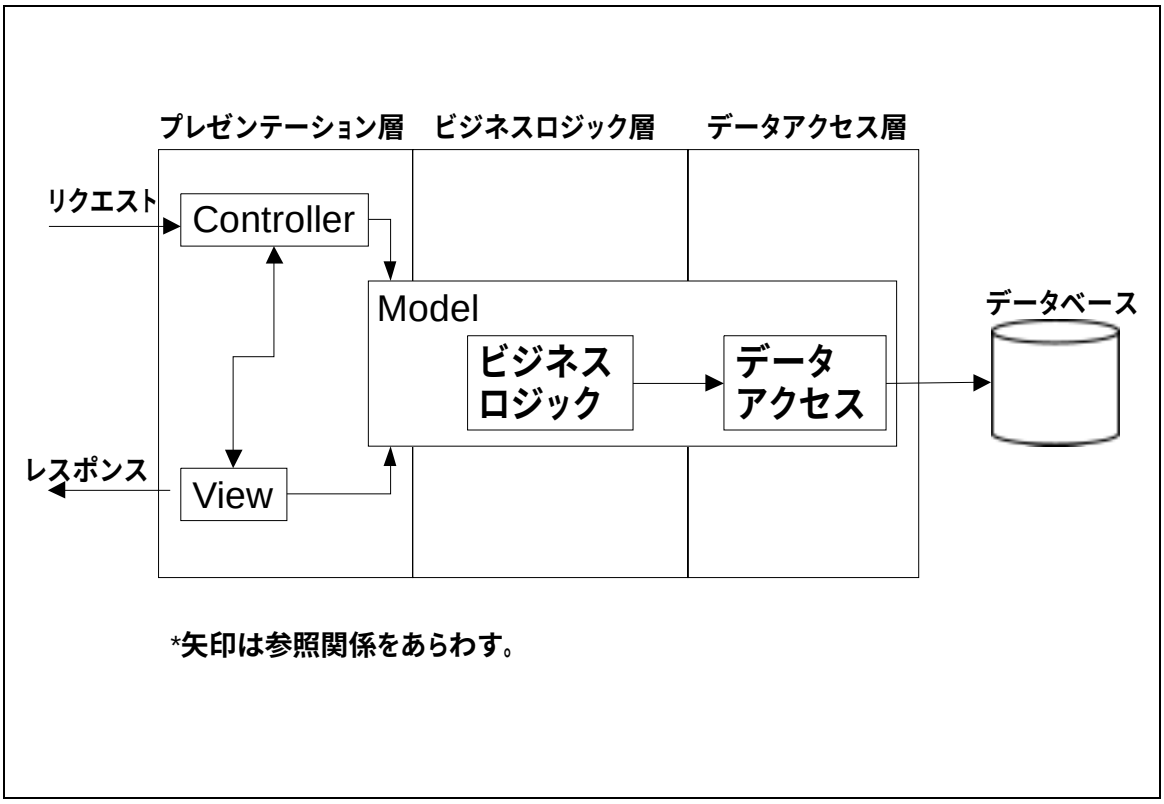


図 II-13-3.　Web アプリケーションの 3 層アーキテクチャ

【解説】

1) Java アプリケーション設計

設計は、オブジェクト指向分析で得られた分析モデルをプログラミング言語で実装可能なモデルに変換する作業である。実装に必要な情報の例としては、ソフトウェアアーキテクチャや、開発言語、使用する外部製品などが含まれる。Java を用いた Web アプリケーションの設計で行うべき手順の例を以下に示す。

* データベース設計
  オブジェクト指向分析で得られた静的モデルを基に、システムに必要なデータ構造を定義し、データベースのスキーマを決定する。
* ソフトウェアアーキテクチャの設計
  フレームワークを使用するか否かを含め、ソフトウェア全体の構造を設計する。
* クラス設計
  ソフトウェアアーキテクチャに基づき、クラスの構成とオブジェクト間のインタフェースを決定する。

2) ソフトウェアアーキテクチャ

Java を用いた Web アプリケーションで広く用いられているアーキテクチャパターンとして、MVC、および 3 層アーキテクチャがあげられる。

* MVC(Model View Controller)
  MVC は GUI を伴うアプリケーションのアーキテクチャパターンであり、アプリケーションを Model、View、Controller の 3 つの主要なコンポーネントに分割する。
  - Model はデータとロジックをカプセル化したコンポーネントであり、EJB もしくは JavaBeans で実装される場合が多い。
  - View は UI を提供するコンポーネントであり、JSP で実装される場合が多い。
  - Controller は入力を処理し、Model や View に処理を振り分けるコンポーネントであり、Servlet で実装される場合が多い。

  MVC では、Model を他のコンポーネントに対して依存しない設計とすることにより、保守性の高いアプリケーションを作成することができる。
* 3 層アーキテクチャ
  3 層アーキテクチャはアプリケーションをプレゼンテーション層、ビジネスロジック層、データベース層(データアクセス)の 3 つのレイヤに分割するアーキテクチャパターンである。
  - プレゼンテーション層は、ユーザとソフトウェアのインタフェースを扱う。
  - ビジネスロジック層は、アプリケーションのロジックを扱う。
  - データベース(データアクセス)層は、データベースへのアクセスを扱う。

  Web アプリケーションにおける 3 層アーキテクチャは、MVC を基本としたプレゼンテーション層と共に設計される場合が多い。この場合、プレゼンテーション層自体は、Controller と View に対応する層であると考えられる。ビジネスロジック層は、Model 内で使用されるモジュールのうちロジックを扱うモジュールに対応する層であると考えられる。データベース(データアクセス)層は、Model 内で使用されるモジュールのうちデータベースにアクセスするモジュールに対応する層であると考えられる。

| スキル区分 | OSS モデルカリキュラムの科目 | レベル |
|---|---|---|
| プログラミング分野 | 13　Java に関する知識 II | 応用 |
| 習得ポイント | II -13-4. Java アプリケーションの開発の作業手順 | |
| 対応するコースウェア | 第 11 回 (Java による Web アプリケーションの設計／実装) | |

# II -13-4. Java アプリケーションの開発の作業手順

Java アプリケーション開発の具体的な作業手順について、プレゼンテーション内容の設計方法、ビジネスロジックの設計方法、データベースの設計方法など各層における設計作業の手順とノウハウについて説明する。

【学習の要点】

* プレゼンテーション層のアーキテクチャ設計の際には、MVC アーキテクチャを設計の基盤として用いる場合が多い。
* ビジネスロジック層の設計の手法としては、トランザクションスクリプト、ドメインモデル、テーブルモジュールが知られている。
* データベース(データアクセス)層の設計の手法としては、行データゲートウェイ、テーブルデータゲートウェイなどが知られている。

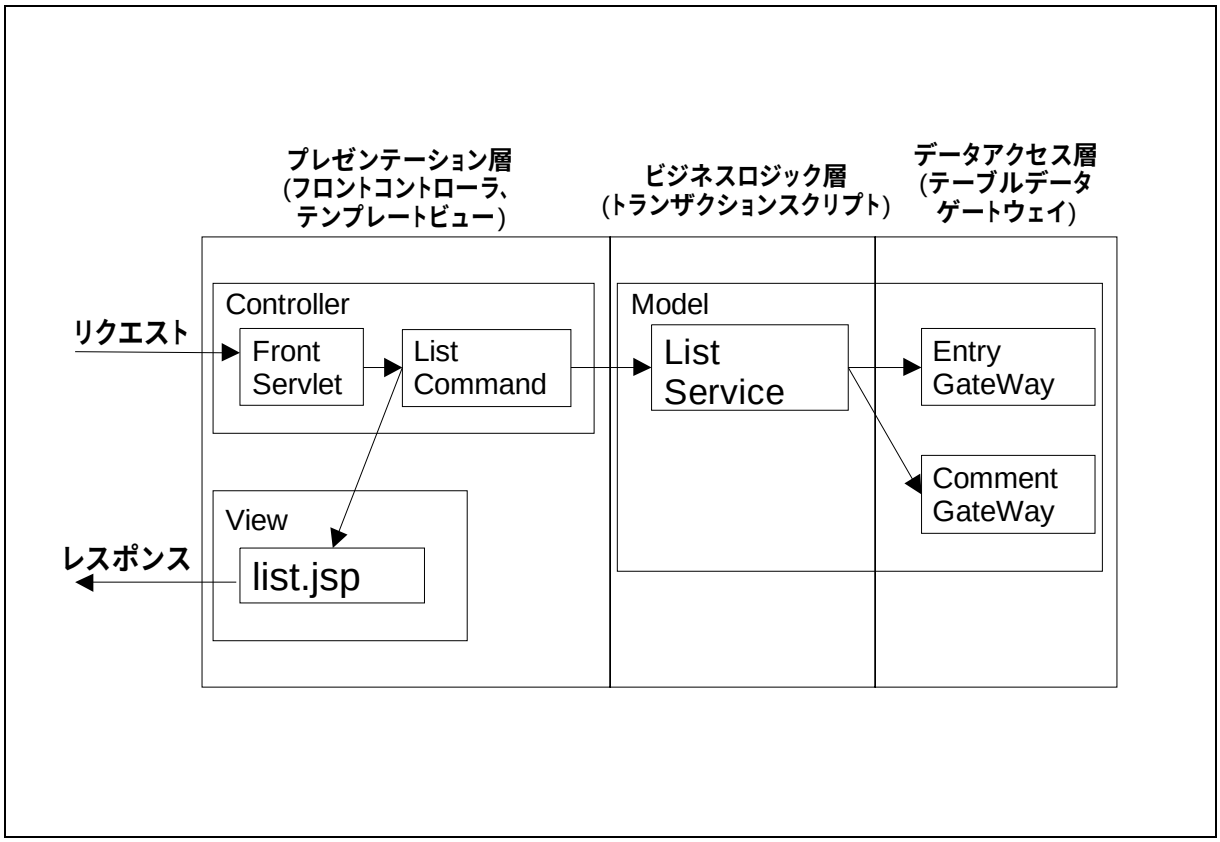


図 II-13-4.　ソフトウェアアーキテクチャの設計例

【解説】

1) プレゼンテーション層の設計

Web アプリケーションの場合、プレゼンテーション層の設計の際には、MVC アーキテクチャを設計の基盤として用いることがほとんどであり、それ以外のアーキテクチャを用いることは稀である。3 層アーキテクチャでは Model の機能はビジネスロジック層、およびデータベース層に委譲されるため、プレゼンテーション層では、View と Controller に関する設計を行う。プレゼンテーション層で用いられる設計のパターンの例を以下に挙げる。

* フロントコントローラ
  Controller に関する設計手法である。1 つのオブジェクトで全ての HTTP リクエストを一旦受付けた後、適切なオブジェクトに処理を委譲する。リクエストに対する共通処理を追加するのが容易であるというメリットがある。また、Struts を始めとした多くの Web アプリケーションフレームワークは、フロントコントローラを採用したプレゼンテーション層を提供する。
* テンプレートビュー
  View に関する設計手法である。通常の HTML の中に特別なタグやマークを使用して動的な要素を埋め込み、実行時にサーバソフトウェアが動的な要素を処理することで、HTML などに変換する。JSP など、多くの実装が提供されている。

2) ビジネスロジック層の設計

ビジネスロジック層の設計の手法としては、トランザクションスクリプト、ドメインモデル、テーブルモジュールが知られている。

* トランザクションスクリプト
  必要なロジックを、機能毎に一連の手続きとして実装する手法。直感的でわかりやすいが、提供する機能が増えた場合に、コードの重複が起きやすく、変更に弱くなる場合がある。
* ドメインモデル
  データと手続きのカプセル化を重視し、ロジックを各オブジェクトに実装したメソッドの組み合わせで実現する手法。オブジェクト指向の特長を生かすことで保守性の高い設計を実現することができるが、設計には高いスキルが要求される。
* テーブルモジュール
  ロジックを機能毎やオブジェクト毎ではなく、データベーステーブル毎にモジュール化する手法。トランザクションスクリプトとドメインモデルの中間的な特徴を持つ。

3) データベース(データアクセス)層の設計

データベース(データアクセス)層の設計の手法としては、行データゲートウェイ、テーブルデータゲートウェイなどが知られている。

* 行データゲートウェイ
  データベーステーブルに対応したゲートウェイクラスを用意し、挿入、更新、削除をインスタンスメソッドとして実装する手法。レコードの検索結果はゲートウェイクラスのインスタンスのセットとして返却される。
* テーブルデータゲートウェイ
  データベーステーブルに対応したゲートウェイクラスを用意し、1つのインスタンスで全てのテーブル操作を実現する手法。レコードの検索結果は実行環境が提供するレコードセットとして返却される。

| スキル区分 | OSS モデルカリキュラムの科目 | レベル |
|---|---|---|
| プログラミング分野 | 13　Java に関する知識 II | 応用 |
| 習得ポイント | II -13-5. JSP によるプレゼンテーションの実装 | |
| 対応する<br>コースウェア | 第 12 回 (Java による Server Side Java／Web アプリケーション実装) | |

# II -13-5. JSP によるプレゼンテーションの実装

JSP (Java Server Pages)の基本と歴史、特徴について示し、JSP を利用したプレゼンテーションの実装について解説する。また JSP と関連する技術や対抗技術、その他、様々なプレゼンテーションの実現方法を紹介する。

【学習の要点】

* JSP は、Servlet 2.1 API の拡張として実装され、バージョン 1.2 より Java Community Process の元で仕様の策定が行われている。
* JSP は HTML 内に Java コードを埋め込むことで、動的なコンテンツの提供を実現する。
* JSP の対抗技術として Tapestry や, JSF(JavaServer Faces)で使用される Facelets などがある。

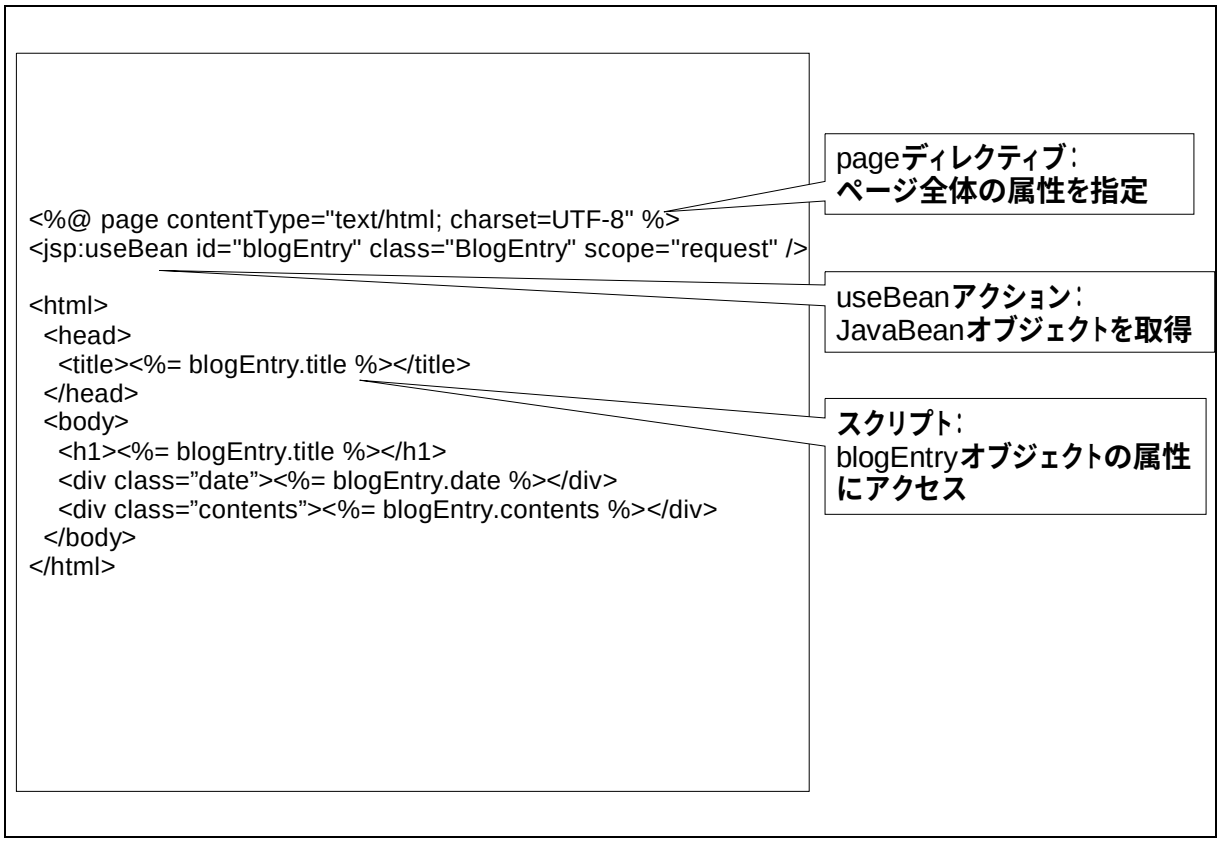


図 II-13-5.　JSP のソースコード例

【解説】

1) JSP とは

JSP は、動的な HTML あるいは XML を生成することを可能にする技術である。Servlet 2.1 API の拡張として実装され、バージョン 1.2 より Java Community Process の元で仕様の策定が行われている。JSP は HTML 内に専用のタグを用いて Java コードなどを埋め込むことで動的なコンテンツを記述する。JSP ファイルは実行時に Servelet コンテナによってバイトコードにコンパイルされる。

2) JSP の構成要素

JSP は以下の要素から構成される。

* 静的な要素
  HTML や、ECMAScript など静的な要素。
* JSP ディレクティブ
  JSP コンパイラに対する指示文を記述する。以下の 3 種類が存在する。
  - include ディレクティブ
    include ディレクティブを記述した場所に指定したファイルを挿入する。
  - page ディレクティブ
    JSP ページ全体に影響するオプションを指定する。
  - taglib ディレクティブ
    タグライブラリの使用を宣言する。
* JSP スクリプト
  Java コードを記述する。
* JSP アクション
  外部の JavaBeans オブジェクトを利用する処理や、他のページに転送する処理などを記述する。

3) JSP の関連技術

JSP は Web アプリケーション用フレームワークの一部としても利用されている。Struts や JSF では、JSP で使用するカスタムタグライブラリが提供されており、JSP で作成したビューと、他のコンポーネントと連携させることが可能になっている。

4) JSP の対抗技術

JSP と同様に動的な Web ページを実現する対抗技術として Tapestry、および JSF で JSP の後継技術として開発された Facelets をとりあげる。

* Tapestry
  Tapestry は Struts や JSF と同様の領域をカバーする Web アプリケーションフレームワークである。View を生成するためのテンプレートに JSP のように独自タグを用いる必要が無いため、動的な要素を追加した後でもコンテナを起動させることなくレイアウトが確認でき、HTML 用のデザインツールとの連携も行いやすいというメリットを持つ。
* Facelets
  JSF (JavaServer Faces) では従来、JSP を View を実現する技術として用いていたが、JSF と JSP の組み合わせに問題が発生したことから、Facelets が JSP に代わる技術として開発された。Facelets は Tapestry と同様、HTML タグを用いて動的な View を記述することができる。

| スキル区分 | OSS モデルカリキュラムの科目 | レベル |
|---|---|---|
| プログラミング分野 | 13　Java に関する知識 II | 応用 |
| 習得ポイント | II -13-6. EJB によるアプリケーション開発の基本 | |
| 対応するコースウェア | 第 9 回 (EJB によるアプリケーション開発) | |

# II -13-6. EJB によるアプリケーション開発の基本

EJB (Enterprise Java Beans)を活用したアプリケーション開発の概要を説明する。EJB コンテナの仕組みやトランザクション処理の手順など、実際のプログラミング事例を示しつつ具体的な手法について解説する。

【学習の要点】

* EJB はモジュール化されたエンタープライズアプリケーションを構築するためのアーキテクチャであり、トランザクションやセキュリティ等、ビジネスロジックの実装に有用な機能が提供される。
* EJB はビジネスロジックを記述した Enterprise Bean を EJB コンテナにデプロイすることで利用可能となる。
* EJB コンテナは、Enterprise Bean の実行環境を提供し、JSP や Servlet を始めとする外部コンポーネントからの呼び出しに応じて Enterprise Bean に記述された処理を実行する機能を持つ。また、データベースへの接続機能やトランザクションの管理機能を提供する。
* EJB それ自体には、ユーザインタフェースを提供する機能が存在しないため、プレゼンテーション層の機能を提供するソフトウェアと組み合わせて利用されることが多い。

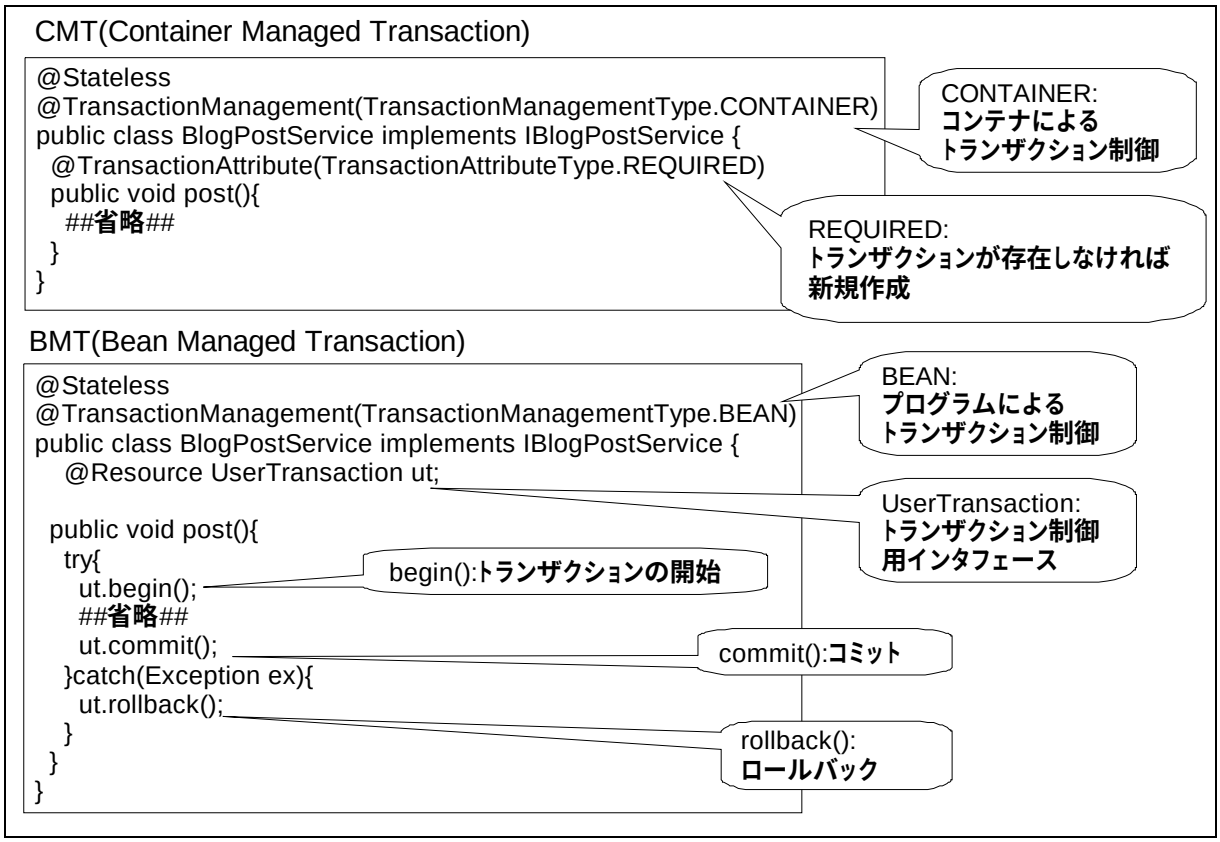


図 II-13-6.　EJB のトランザクション制御

【解説】

1) EJB とは

EJB はエンタープライズアプリケーションにおけるビジネスロジックの実現を容易にすることを目的として策定されたアーキテクチャ仕様である。Java EE の仕様の一部として作成され Java Community Process の元で改良が進められている。EJB を利用することで、トランザクションやセキュリティ等、ビジネスロジックの実装に有用な機能が提供されるほか、エンタープライズアプリケーションのモジュール化を推進し、再利用を容易にする効果があると考えられている。

2) EJB3

EJB1.x および EJB2.x では、仕様が複雑で実装がしづらいなどの要因により、EJB を利用した開発は必ずしも開発者に支持されてこなかった。2006 年に発表された EJB3.0 では、その反省を踏まえ開発のし易さ(Ease of Development)が重視されており、POJO(継承を伴わないシンプルな Java オブジェクト)とアノテーションを中心とした仕様となっている。

3) EJB の実行環境

EJB の実行環境を EJB コンテナと呼び、プロプライエタリな製品のほか、JBoss や Apache Geronimo などの OSS も公開されている。EJB コンテナは Servlet や JSP などの呼び出しに応じて Enterprise Bean を動作させ、ビジネスロジックを実現する。さらに、トランザクション管理機能やデータベース接続機能などがコンテナから提供される。機能の多くは、アノテーションや設定ファイルによる宣言的な記述により利用することが可能である。

4) EJB のトランザクション管理機能

EJB ではトランザクション管理の方式として CMT と BMT の 2 種類がある。どちらを利用するかは Session Bean クラスに対する@TransactionManagement アノテーションによって指定し、指定しなかった場合は CMT となる。

* CMT(Container Managed Transaction)
  トランザクションの開始と終了は EJB コンテナによって制御される。基本的な動作は、メソッドの開始時にトランザクションが開始され、メソッドが正常終了した際にコミットを行う。メソッドが例外をスローした場合にはロールバックする。EJB のクラスまたはメソッドに@TransactionAttribute アノテーションを付加することにより、制御方法を指定可能である。
* BMT(Bean Managed Transaction)
  トランザクションの開始と終了は JTA(Java Transaction API)の UserTransaction インタフェースを利用して、プログラムによって制御する。
  - トランザクションの開始は UserTransaction インタフェースの begin メソッドによって行う。
  - コミットは UserTransaction インタフェースの commit メソッドによって行う。
  - ロールバックは UserTransaction インタフェースの rollback メソッドによって行う。

5) EJB のインタフェース

EJB それ自体には、ユーザインタフェースを提供する機能が存在しないため、プレゼンテーション層の機能を提供するモジュールと組み合わせて利用されることが多い。クライアントとなるモジュールが同一 JVM 内である場合はローカルインタフェース、別 JVM である場合はリモートインタフェースを通じてアクセスする。

| スキル区分 | OSS モデルカリキュラムの科目 | レベル |
|---|---|---|
| プログラミング分野 | 13　Java に関する知識 II | 応用 |
| 習得ポイント | II -13-7. EJB によるビジネスロジックの実装 | |
| 対応する<br>コースウェア | 第 12 回 (Java による Server Side Java／Web アプリケーション実装) | |

# II -13-7. EJB によるアプリケーション開発

3 層アーキテクチャにおけるビジネスロジック、およびデータアクセス部分の実装について解説する。具体的な例として EJB3.0 仕様に基づいた Entity クラス、および Session Bean のプログラミング例を示す。

【学習の要点】

* EJB3.0 では、POJO にアノテーションを付加することで、Enterprise Bean を記述する。
* EJB3.0 仕様の一部である JPA(Java Persistent API)では、Entity クラスを利用した O/R マッピング仕様が規定されている。
* 3 層アーキテクチャにおけるビジネスロジック部分の処理は Session Bean に記述する。

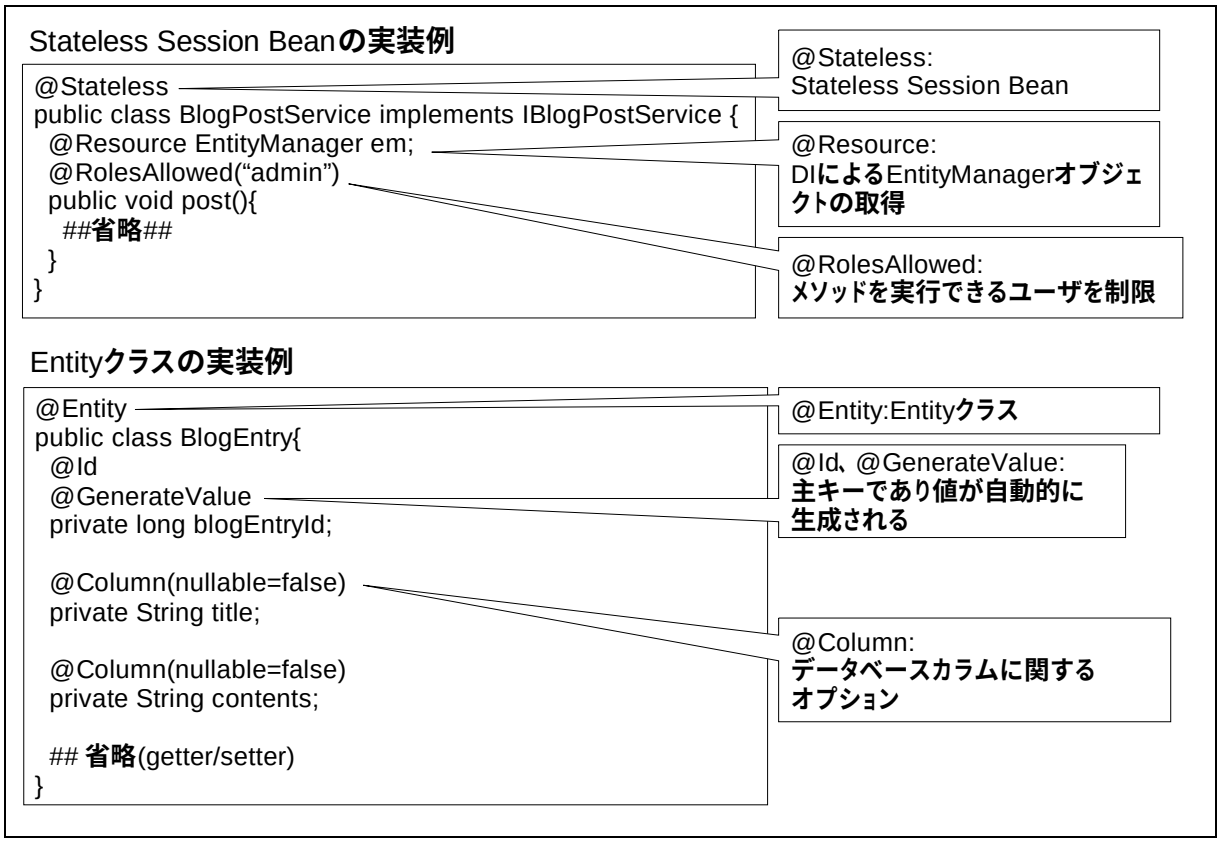


図 II-13-7.　Session Bean と Entity クラスのソースコード例

【解説】

1) Entity クラス

Entity クラスはデータベースに保存されるオブジェクトを実装するためのクラスであり、EJB3.0 仕様の一部である JPA(Java Persistent API)に仕様が規定されている。Entity クラスとして実装したクラスは、EJB コンテナによってデータベーステーブルと関連付けられる。レコードの検索、保存は JPA の EntitiyManager インタフェースを利用して行い、検索結果は Entitiy クラスのインスタンスセットとして取得できる。なお、EJB2.x 以前の仕様では、Entity Bean がデータベース接続技術として用いられていた。EJB2.x 以前の Entity Bean と、EJB3.0 の JPA 仕様に基づく Entity クラスは全く別の仕様であるが、後者を「EJB3.0 における Entity Bean」と表現する場合もある。

2) Entity クラスの実装

Entity クラスは POJO にアノテーションを付加することで定義する。

* クラスに@Entity アノテーションを付加することにより Entity クラスであることを示す。
* 属性または getter メソッドに、@Id アノテーションを付加することにより、この属性が主キーであることを示す。
* @Id アノテーションを付加した属性または getter メソッドに、@GeneratedValue アノテーションを付加することにより、主キーの値が自動生成されることを示す。
* 属性または getter メソッドに、@Column アノテーションを付加することにより、カラム名や nullable か否かなど、データベースのカラムに関する情報を指定できる。
* 属性または getter メソッドに、@ManyToOne、@OneToOne、@OneToMany、@ManyToMany アノテーションのいずれかを指定することにより他の Entity クラス(テーブル)との関連を指定できる。

3) Session Bean

Session Bean はビジネスロジックを実装するための Bean である。トランザクション属性やセキュリティ属性は Session Bean に対して指定する。内部状態を持つ Stateful Session Bean と持たない Stateless Session Bean が存在する。

4) Session Bean の実装

Session Bean はクラスおよびインタフェースにアノテーションを付加することにより定義する。

* クラスに@Stateful アノテーションを付加することにより Stateful Session Bean であることを示す。
* クラスに@Stateless アノテーションを付加することにより Stateless Session Bean であることを示す。
* Session Bean は、外部インタフェースを実装する必要がある。@Local アノテーションをインタフェースに付加することにより、同一 JVM 内からアクセスするためのインタフェースであることを示す。@Remote アノテーションを付加することにより、別 JVM からアクセスするためのインタフェースであることを示す。
* クラスまたはメソッドに@RolesAllowed アノテーションを付加することにより、メソッドの実行が可能な Role を制限することができる。
* Session Bean 内に EntityManager 型の属性を定義し、@PersistenceContext アノテーションを付加することにより、EntitiyManager オブジェクトが Inject され、Session Bean 内でデータベース操作を行うことが可能になる。

| スキル区分 | OSS モデルカリキュラムの科目 | レベル |
|---|---|---|
| プログラミング分野 | 13　Java に関する知識 II | 応用 |
| 習得ポイント | II -13-8. JDBC によるデータベースの操作 | |
| 対応するコースウェア | 第 12 回 (Java による Server Side Java／Web アプリケーション実装) | |

# II -13-8. JDBC によるデータベースの操作

JDBC (Java DataBase Connectivity)の基本と歴史、特徴について示し、JDBC を利用したデータベースの操作方法について解説する。JDBC によるデータベース操作の具体的な方法やプログラミング例を示し、データベース操作の基本手順を紹介する。

【学習の要点】

* JDBC は Java プログラムからデータベースにアクセスする方法を定義した API である。
* JDBC は JDK1.1 から標準 API に含まれており、JDK6 には JDBC4.0 が含まれている。
* JDBC を利用してデータベースにアクセスする際には、接続方法、および接続先データベースに応じた JDBC ドライバを必要とする。
* JDBC にはデータベース接続、トランザクションの管理、SQL の実行などの操作を行うための標準的な方法が用意されている。

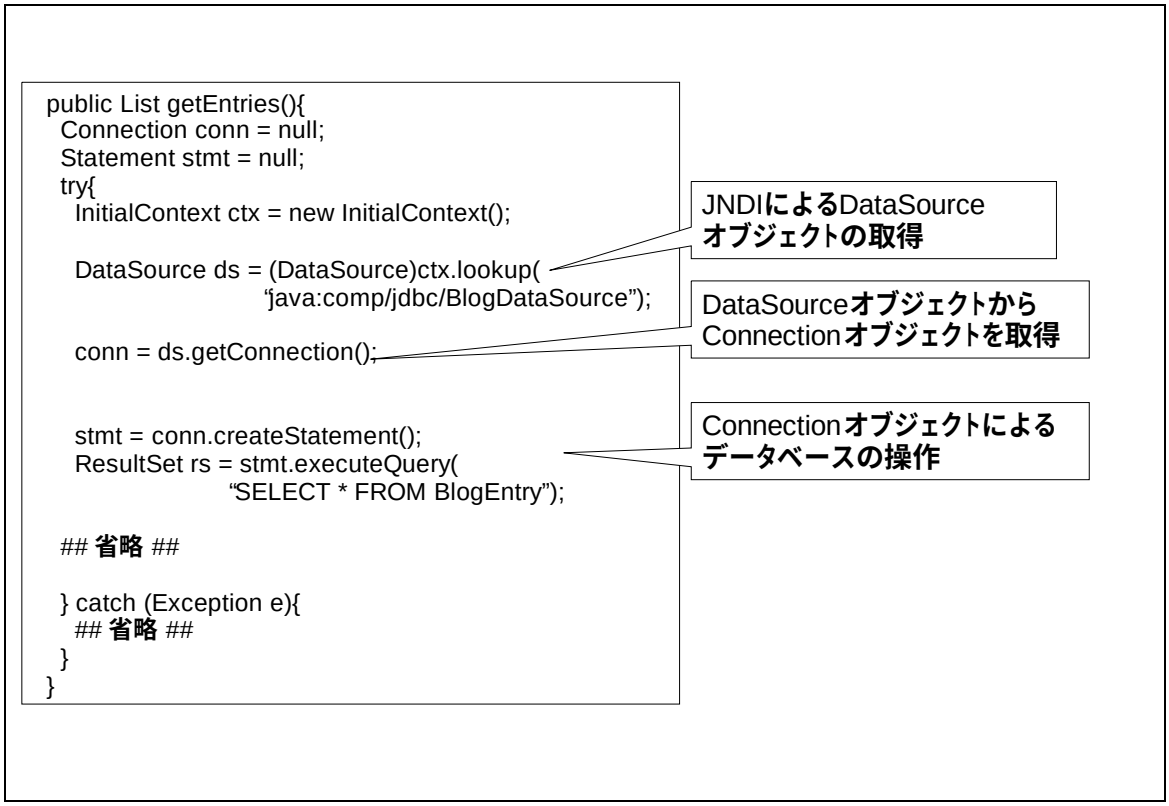


図 II-13-8.　DataSource インタフェースを利用したデータベース操作の例

【解説】

1) JDBC

JDBC は Java プログラムからデータベースにアクセスする方法を定義した API であり、標準化された手順で各種データベースに対するアクセスを行うことができる。JDBC は JDK1.1 から標準 API に含まれており、version 3.0 からは Java Community Process により仕様の策定が行われている。JDK6 には JDBC4.0 が含まれている。

2) JDBC ドライバ

JDBC を用いて各種データベースに接続する場合には、データベースに対応した JDBC ドライバが必要となる。詳細は「I-13-9. JDBC によるデータベースアクセス方法」を参照のこと。JDK6 には Java DB と呼ばれる Java 言語により実装されたデータベースが含まれており、外部のデータベース、および JDBC ドライバを導入することなくデータベースプログラミングを行うことが可能となっている。なお、Java DB は Apache プロジェクト内で開発されている Apache Derby を Sun Microsystems 社がサポートしたものである。

3) Java EE におけるデータベース接続

JDBC の最も基本的な使い方である、DriverManager クラスを利用したデータベースアクセスの方法については、「I-13-9. JDBC によるデータベースアクセス方法」を参照のこと。Java EE では DriverManager クラスの代わりに DataSource インタフェースを利用してデータベース接続を取得する方法が推奨されている。

* DataSource インタフェースの実装は JDBC ドライバの提供元や、フレームワーク、Java EE コンテナなどから提供されている。
* DataSource インタフェースを経由してデータベース接続を取得することにより、接続プーリングなど、高度な機能が利用可能になる。
* 利用する DataSource クラスや JDBC ドライバ、および各種接続情報やパラメータは、コンテナの指定する設定ファイルに記述し、プログラム内からは、JNDI で取得する場合が多い。
* DI(Dependency Injection)の機能を提供するコンテナを利用している場合、DataSource クラスの取得はコンテナによって自動的に行われる。

4) DataSource インタフェースを利用したデータベース接続手順

* JNDI リソースの登録
  登録手順は、コンテナのドキュメントを参照のこと。
* JNDI を利用した DataSource オブジェクトの取得
  - new javax.naming.InitialContext()により、Context オブジェクトを取得する。
  - Context オブジェクトの lookup メソッドにより、DataSource オブジェクトを取得する。
* DI を利用した DataSource オブジェクトの取得
  EJB コンテナでは、Session Bean クラスに@Resources アノテーションを付加することにより、DataSource オブジェクトを DI を利用して取得することができる。
* Connection オブジェクトの取得
  DataSource オブジェクトの getConnection メソッドにより Connection オブジェクトを取得する。
* 取得した Connection オブジェクトを利用してデータベースを操作する方法については、「I-13-9. JDBC によるデータベースアクセス方法」を参照のこと。

| スキル区分 | OSS モデルカリキュラムの科目 | レベル |
|---|---|---|
| プログラミング分野 | 13　Java に関する知識 II | 応用 |
| 習得ポイント | II -13-9. デザインパターン | |
| 対応するコースウェア | 第 14 回 (デザインパターンによる開発手順) | |

# II -13-9. デザインパターン

デザインパターンの基本的な考え方や歴史、特徴、デザインパターンの意義について解説する。さらにデザインパターンの効果的な使い方や代表的なパターンのいくつかを紹介し、Java のフレームワークで利用されているデザインパターンの適用例を示す。

【学習の要点】

* デザインパターンとは、システム設計時にしばしば現れる問題を解決するために過去に用いられてきた優れた設計に対し、体系的に名前付けし、説明を加え、評価したものである。
* デザインパターンは建築分野において Christopher Alexander により提唱されていた概念であるが、1994 年に書籍「Design Patterns: Elements of Reusable Object-Oriented Software(オブジェクト指向における再利用のためのデザインパターン)」が出版されて以降、ソフトウェア開発の分野においても広く用いられている。
* 個々のデザインパターンには、パターンそのものと共に、パターンによって解決可能な問題、発生するトレードオフが明記されている。デザインパターンの適用にあたっては、システムが抱える設計上の問題を明かにした上で、問題に適したパターンを選択し、トレードオフについても考慮する必要がある。
* デザインパターンは特にライブラリやフレームワークの分野で積極的に利用されており、JDK 標準ライブラリなどでも実例を見ることができる。

**【生成に関するパターン】**
- Abstract Factory**パターン**
  **依存しあうオブジェクト群の生成をカプセル化**
- Builder**パターン**
  **複合オブジェクトの作成過程をカプセル化**
- Factory Method**パターン**
  **オブジェクトの生成をサブクラスに任せる**
- Prototype**パターン**
  **プロトタイプのコピーによるインスタンスの生成**
- Singleton**パターン**
  **インスタンスが1つしか存在しないことを保証**

**【構造に関するパターン】**
- Adapter**パターン**
  **インタフェースの変換**
- Bridge**パターン**
  **抽出されたクラスとその実装を分離**
- Composite**パターン**
  **個々のオブジェクトとオブジェクトを合成したものを一様に扱う**
- Decorator**パターン**
  **オブジェクトに責任を動的に追加**
- Facade**パターン**
  **インタフェースを単純化**
- Flyweght**パターン**
  **多数の細かいオブジェクトのサポートを共有により効率化**
- Proxy**パターン**
  **代理オブジェクトにより、アクセスを制限**

**【振る舞いに関するパターン】**
- Chain of Responsibility **パターン**
  **オブジェクトをチェーン状に繋ぎ、要求が処理されるまでチェーンに沿って要求を渡していく**
- Command**パターン**
  **要求をオブジェクトとしてカプセル化**
- Interpreter**パターン**
  **言語の文法表現とインタプリタを一緒に定義する**
- Iterator**パターン**
  **集約オブジェクトの内部を公開せずに、順にアクセスする方法を提供する**
- Mediator**パターン**
  **オブジェクト群の相互作用をカプセル化**
- Memento**パターン**
  **カプセル化を破壊せずにオブジェクトの内部状態を外面化**
- Observer**パターン**
  **オブジェクトの状態の変化を、依存するオブジェクト群に通知**
- State**パターン**
  **オブジェクトの状態と、状態に依存する振る舞いをカプセル化**
- Strategy
  **ある処理に対するアルゴリズムをカプセル化**
- Template Method**パターン**
  **アルゴリズムの中のいくつかのステップをサブクラスに任せる**
- Visitor**パターン**
  **あるオブジェクトの構造上の要素で実行されるオペレーションを表現する**

図 II-13-9.　GoF のデザインパターン

【解説】

1) デザインパターン

デザインパターンは、システム設計時にしばしば発生する問題に対する「再利用可能な」解決策をカタログの形で文書化したものである。デザインパターンの考え方は、建築分野においてChristopher Alexanderによって提唱されたパタン・ランゲージを起源とする。Christopher Alexanderによれば、優れた設計にはパターンが存在し、過去に用いられてきたパターンを組み合わせることによって、複雑な相互作用を持つ構造物に対する優れた設計を行うことができる。デザインパターンの考え方は、1994 年の書籍「Design Patterns: Elements of Reusable Object-Oriented Software(オブジェクト指向における再利用のためのデザインパターン)」によってソフトウェア開発の分野でも広く用いられるようになった。この書籍に収められた23のデザインパターンは著者のErich Gamma、Richard Helm、Ralph Johnson、John Vlissidesの通称Gang of Fourにちなみ、「GoFのデザインパターン」と呼ばれる。

2) デザインパターンの記述

デザインパターンは、再利用がしやすいように、適切な規約に従って分類、整理されている。GoFによれば、パターンは以下の4つの基本的な要素を有している。

* パターン名 - その解法およびその結果を1、2語で記述した名称。
* 問題 - パターンが解決する設計上の問題。
* 解法 - クラスやオブジェクトなど設計の要素、および関連、責任、協調関係。
* 結果 - パターンを適用する際の結果やトレードオフ。

3) デザインパターンのメリットとデメリット

* デザインパターンが提供する解決策の多くは、ソフトウェアの修正と拡張に関するものである。適切なデザインパターンを適用することで、既存のモジュールを修正することなく、ソフトウェアの機能の修正、追加が可能な設計を実現できる。ただし、ソフトウェアのどの部分に将来修正、拡張が発生するのかを見極め、適切なパターンを選択することは設計者の責任となる。
* デザインパターンを適用した際の一般的なデメリットとしては、間接層の導入や、モジュールの細分化により、設計が複雑化し、開発コストが上昇することがあげられる。適切なパターンを選択しなかった場合には、メリットを享受できないばかりか、無用な不利益を被ることにつながるため、デザインパターンの無秩序な適用は避けるべきである。

4) デザインパターンの例

* Strategy パターン
  ある処理を行うアルゴリズムの実装を、特定のインタフェースを持つクラスとしてカプセル化し、アルゴリズムを交換可能とする。Java標準ライブラリの使用例としては、java.util.Comparatorインタフェースを利用したソート順の制御が挙げられる。
* Decorator パターン
  オブジェクトに機能を動的に追加することを可能にする。Java 標準ライブラリの使用例としては、java.io.InputStream オブジェクトに対し、バッファ機能や Zip ファイルの読み込み機能などを動的に付加可能となっている例が挙げられる。

| スキル区分 | OSS モデルカリキュラムの科目 | レベル |
|---|---|---|
| プログラミング分野 | 13　Java に関する知識 II | 応用 |
| 習得ポイント | II -13-10. Java アプリケーションのチューニング | |
| 対応するコースウェア | 第 15 回 (Java のパフォーマンスチューニング) | |

## II -13-10. Java アプリケーションのチューニング

Web アプリケーションにおけるパフォーマンス向上の基礎を解説する。パフォーマンスチューニングを行う際の戦略やプロファイリングの手法、JVM 自体のパラメータチューニング方法など、パフォーマンス向上策に関するポイントを紹介する。

【学習の要点】

* パフォーマンスチューニングにおいて始めになすべきことはボトルネックの特定である。
* ボトルネックを特定するためのツールとしてプロファイラを用いることができる。JDK に標準で付属している hprof を始めとした様々なプロファイラを利用することができる。
* ガベージコレクションが頻繁に発生している場合など、JVM 自体の挙動がパフォーマンスを悪化させている場合には、JVM のパラメータチューニングを行うことによってパフォーマンスの改善を行うことができる場合がある。

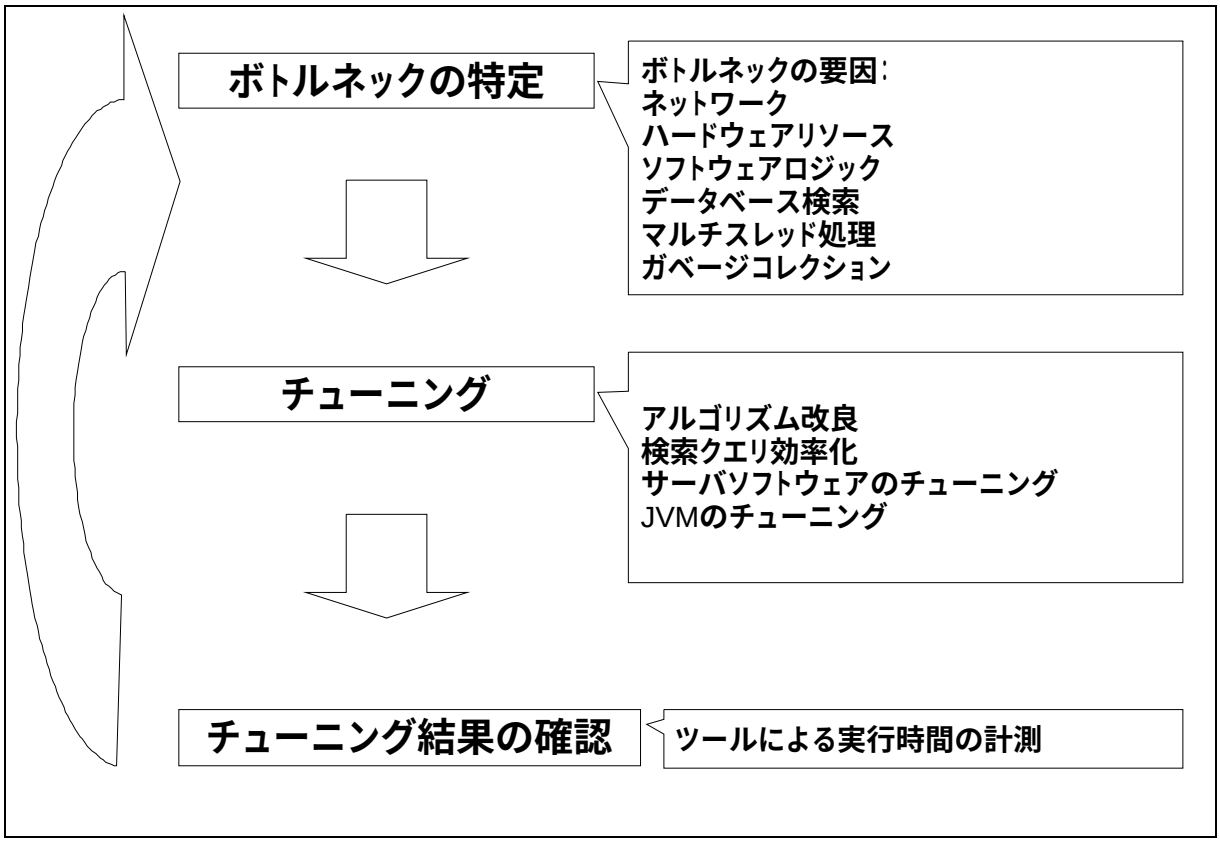


図 II-13-10.　Java アプリケーションのパフォーマンスチューニング

【解説】

1) パフォーマンスチューニングの手順

一般にソフトウェアに性能問題が発生した場合、始めに行うべき作業はボトルネックの特定である。ボトルネックは、性能測定ツールによる処理毎の実行時間の計測や、サーバリソースの使用状況の確認などによって特定する。開発のどの段階であっても、勘や経験のみに頼ったチューニングを安易に行うべきではない。パフォーマンスチューニングは、

* ボトルネックの特定
* チューニング
* 実行時間の計測(チューニング効果の確認)

のサイクルを守って行う。

2) ボトルネックの特定に使用できるツール

* ネットワーク性能を測定するツールとして、netperf を利用することができる。Web アプリケーションとは独立にネットワーク性能を測定することにより、ボトルネックがネットワークにあるのかサーバ内の処理にあるのかを区別することができる。
* Red Hat 系や Debian 系の Linux ディストリビューションでは、sysstat パッケージを導入することにより、CPU や I/O など、リソースの使用状況についての統計情報を得ることができる。リソースの使用状況を確認することにより、ソフトウェアとハードウェアの内どちらがボトルネックなのか、およびソフトウェアの行っている処理の内、ボトルネックになっている処理の種類、を識別することが可能になる。
* JVM にはガベージコレクションのログを出力する機能がある。このログを参照することにより、JVM 自体にボトルネックがあるケースを識別することが可能になる。
* JVM にはスレッドダンプを出力する機能がある。スレッドダンプを参照することにより、排他制御に問題がある場合など、マルチスレッドが効率的に動作していないケースを識別することが可能になる。
* アプリケーションの内部動作を解析するプロファイラとしては、JDK に標準で付属している hprof などを利用できる。プロファイリング結果を参照することにより、アプリケーションのロジックレベルでのボトルネックを識別することが可能になる。
* ユーザが行うリクエストをシミュレートするツールとして JMeter が OSS として公開されている。単一リクエストによる性能測定のほか、複数リクエストの同時実行による負荷テストも行うことが可能である。

3) JVM のチューニング

JVM の挙動がボトルネックであると考えられる場合、JVM のパラメータチューニングによりパフォーマンスが改善する可能性がある。 例として、ガベージコレクションが頻繁に発生し、無応答などの現象を引き起こしている場合には、JVM のヒープ領域大きさをチューニングすることにより、ガベージコレクションの発生頻度を抑えることが可能な場合がある。これらのパラメータは起動時のオプションとして指定可能であるが、設定の変更にあたっては JVM の挙動についての知識が必要となる。