JDBC基本
スライド教材

Javaとデータベースの橋渡し技術

2025年8月15日

目次

このJDBC基本学習資料では、Javaとデータベースの接続から実践的な活用まで段階的に学習を進められるよう構成されています。

進捗状況: 2/14

はじめに - JDBCとは

JDBC（Java Database Connectivity）は、Javaプラットフォームで標準的なデータベースアクセス方法を提供するAPIです。 データベースメーカーに依存しない統一的なインターフェースでデータベース操作を行えます。

JDBCの主な特徴

- データベース独立性 - 様々なDBMSに対応
- 標準化されたAPI - java.sqlパッケージ
- ドライバーベースの接続方式
- 効率的なリソース管理とトランザクション制御

> **なぜJDBCが重要なのか**
>
> エンタープライズアプリケーション開発において、データベースとの確実で安全な連携を実現する基盤技術として必須のスキルです。

JDBCアーキテクチャ

- JDBCドライバーマネージャー
- JDBC API（java.sql）
- データベース固有ドライバー
- データベース管理システム

対応データベース例

- PostgreSQL（おすすめ）
- Oracle Database
- MySQL / MariaDB
- Microsoft SQL Server
- SQLite（テスト環境）

基本概念 - JDBC API構造

JDBC APIは java.sql パッケージに含まれる主要インターフェースで構成されており、 各インターフェースが特定の役割を担ってデータベース操作を実現します。

主要インターフェース

Connection
データベースとの接続を表す

Statement
SQLクエリの実行を担当

PreparedStatement
パラメータ化クエリの実行

ResultSet
クエリ結果の取得と処理

その他の重要なコンポーネント

- **DriverManager** - ドライバー管理
- **DataSource** - コネクションプール対応
- **CallableStatement** - ストアドプロシージャ
- **DatabaseMetaData** - DB情報の取得

実行の流れ

1. Connectionの取得
2. Statement作成
3. SQLの実行
4. ResultSetの処理
5. リソースのクローズ

環境準備 - ドライバー設定

1. **JDBCドライバーの選択**
   使用するデータベースに対応したドライバーを入手
   - PostgreSQL, Oracle, MySQL等
2. **クラスパス設定**
   ドライバーJARファイルを実行時パスに追加
   - IDE: ライブラリに追加
   - Maven/Gradle: dependency設定
3. **接続情報の準備**
   データベース接続に必要な情報を用意
   - 接続URL、ユーザー名、パスワード

主要JDBCドライバー

	PostgreSQL JDBC postgresql-42.6.0.jar	取得
C	Oracle JDBC ojdbc8.jar	取得
	MySQL Connector/J mysql-connector-java-8.0.x	取得
	SQLite JDBC sqlite-jdbc-3.x.x.jar	取得

Maven依存関係例

```
<dependency>
  <groupId>org.postgresql</groupId>
  <artifactId>postgresql</artifactId>
  <version>42.6.0</version>
</dependency>
```

基本構成 - 接続の確立

JDBCでデータベースに接続するには、DriverManagerを使用して接続文字列（URL）、ユーザー名、パスワードを指定します。

接続の基本手順

- **ドライバーのロード** - Class.forName()またはauto-load
- **接続の取得** - DriverManager.getConnection()
- **接続テスト** - isValid()メソッドで確認
- **リソースクローズ** - try-with-resourcesを活用

接続文字列の例

```
// PostgreSQL
jdbc:postgresql://localhost:5432/mydb

// MySQL
jdbc:mysql://localhost:3306/mydb
```

接続パラメータ

- **ホスト名** - localhost, IPアドレス
- **ポート番号** - PostgreSQL:5432, Oracle:1521
- **データベース名** - 接続対象のDB
- **認証情報** - ユーザー名・パスワード

ベストプラクティス

- 接続情報を設定ファイルで管理
- try-with-resourcesでリソース管理
- 接続プールの活用を検討
- タイムアウト設定の適切な値

動作原理 - SQLの実行フロー

JDBCでのSQL実行は、Statementオブジェクトを通じて行われます。executeQuery（SELECT）とexecuteUpdate（INSERT/UPDATE/DELETE）の使い分けが重要です。

実行メソッドの種類

- **executeQuery()** - SELECT文の実行（ResultSet返却）
- **executeUpdate()** - INSERT/UPDATE/DELETE（影響行数返却）
- **execute()** - 任意のSQL実行（汎用的）
- **executeBatch()** - バッチ処理の実行

実行フロー

1. Connectionの取得
2. Statementの作成
3. SQLの実行
4. 結果の処理
5. リソースのクローズ

トランザクション管理

- **自動コミット** - デフォルトで有効
- **手動制御** - setAutoCommit(false)
- **コミット** - commit()で確定
- **ロールバック** - rollback()で取消

バッチ処理の利点

- 複数のSQL文を一括実行
- ネットワーク通信の最適化
- パフォーマンスの向上
- トランザクションの一元管理

コア機能 - PreparedStatement活用

PreparedStatementは、パラメータ化クエリを使用してSQLインジェクション攻撃を防ぎ、 パフォーマンスの向上も実現する重要な機能です。

主要な利点

- **セキュリティ** - SQLインジェクション対策
- **パフォーマンス** - SQL文の事前コンパイル
- **可読性** - パラメータによる明確な構造
- **再利用性** - 同一SQL文の繰り返し実行

> ⚠ SQLインジェクション対策
>
> 文字列連結でSQL構築する代わりに、?プレースホルダーとパラメータバインディングを必ず使用しましょう。

基本的な使用方法

- **プレースホルダー** - ? でパラメータ位置を指定
- **パラメータバインド** - setString(), setInt()等
- **実行** - executeQuery() / executeUpdate()
- **クローズ** - try-with-resourcesを活用

パラメータ設定メソッド

- setString() - 文字列
- setInt() - 整数
- setDate() - 日付
- setTimestamp() - 日時
- setNull() - NULL値

応用機能 - ResultSet操作

ResultSetは、SELECT文の実行結果を表すオブジェクトで、カーソル型のデータアクセスを提供します。 効率的なデータ取得と処理の実現に重要な役割を果たします。

ResultSetの種類

- **FORWARD_ONLY** - 前方向のみ移動（デフォルト）
- **SCROLL_INSENSITIVE** - 双方向スクロール可能
- **SCROLL_SENSITIVE** - 変更に敏感なスクロール
- **READ_ONLY** - 読み取り専用

カーソル移動メソッド

- next() - 次の行へ移動
- previous() - 前の行へ移動
- first() - 最初の行へ
- last() - 最後の行へ

データ取得メソッド

- **getString()** - 文字列データの取得
- **getInt()** - 整数データの取得
- **getDate()** - 日付データの取得
- **getTimestamp()** - 日時データの取得

メタデータの活用

- ResultSetMetaData - 列情報の取得
- getColumnCount() - 列数の取得
- getColumnName() - 列名の取得
- getColumnType() - データ型の取得

データ取扱 - トランザクション制御

トランザクション制御は、データの整合性を保つための重要な機能です。 複数のSQL文を一つの単位として処理し、すべて成功するか、すべて失敗するかを制御します。

トランザクションの基本

- **自動コミット** - setAutoCommit(true) デフォルト設定
- **手動制御** - setAutoCommit(false) で無効化
- **コミット** - commit() で変更を確定
- **ロールバック** - rollback() で変更を取消

ACID特性

- Atomicity (原子性)
- Consistency (一貫性)
- Isolation (独立性)
- Durability (永続性)

分離レベル

- **READ_UNCOMMITTED** - 最低レベル
- **READ_COMMITTED** - 標準レベル
- **REPEATABLE_READ** - 反復読み取り
- **SERIALIZABLE** - 最高レベル

セーブポイント

- setSavepoint() - セーブポイント作成
- rollback(savepoint) - 部分ロールバック
- releaseSavepoint() - 解放
- 複雑な処理の段階的制御

実践応用 - DAOパターン実装

DAO（Data Access Object）パターンは、データアクセス層を抽象化し、ビジネスロジックとデータベース操作を分離する重要な設計パターンです。

DAOパターンの利点

- **関心の分離** - データアクセスとビジネスロジック
- **コードの再利用** - 共通データ操作の抽象化
- **保守性向上** - DB変更時の影響を局所化
- **テスト容易性** - モック・スタブの活用

基本的なCRUD操作

- Create - データの新規作成
- Read - データの取得・検索
- Update - データの更新
- Delete - データの削除

実装構成要素

- **エンティティクラス** - データ構造の表現
- **DAOインターフェース** - 操作の定義
- **DAO実装クラス** - 具体的なDB操作
- **コネクション管理** - DataSourceの活用

設計のポイント

- 例外処理の統一化
- try-with-resourcesでリソース管理
- PreparedStatementの活用
- トランザクション境界の明確化

学習指針 - ベストプラクティス

JDBCを実際の開発で効果的に活用するための重要なベストプラクティスと 推奨アプローチをまとめました。

リソース管理

- **try-with-resources** - 自動リソースクローズ
- **コネクションプール** - HikariCP等の活用
- **タイムアウト設定** - 適切な値での制御
- **リークの防止** - 確実なクローズ処理

パフォーマンス最適化

- PreparedStatementの再利用
- バッチ処理での一括実行
- 適切なフェッチサイズ設定

セキュリティ対策

- **SQLインジェクション対策** - PreparedStatement必須
- **認証情報管理** - 設定ファイル・環境変数
- **最小権限原則** - 必要最小限のDB権限
- **暗号化通信** - SSL/TLS接続の使用

次のステップ

- Spring Data JPA - ORM活用
- MyBatis - SQLマッパー
- Hibernate - JPA実装
- jOOQ - タイプセーフSQL

よくある質問（FAQ）

Q1. データベースに接続できません

- ドライバーJARファイルがクラスパスに含まれているか確認
- 接続URL、ユーザー名、パスワードが正しいか確認
- データベースサーバーが起動しているか確認

Q2. ResultSetがクローズされているエラー

- ResultSetを使用する前にStatementがクローズされていないか確認
- try-with-resourcesで適切にリソース管理を行う
- 複数の結果セットを同時に処理する場合は注意

Q3. 文字化けが発生します

- 接続URLに文字エンコーディングを指定
- ?useUnicode=true&characterEncoding=UTF-8 を追加
- データベースの文字セット設定を確認

Q4. パフォーマンスが悪いです

- PreparedStatementを使用してSQL文をキャッシュ
- バッチ処理で複数操作を一括実行
- コネクションプールを導入してオーバーヘッド削減

Q5. デッドロックが発生します

- トランザクションの範囲を適切に設計
- 常に同じ順序でテーブルにアクセス
- 長時間のトランザクションを避ける

⚠ 重要な注意点

- try-with-resourcesを必ず使用
- PreparedStatementでSQLインジェクション対策
- 適切な例外処理とログ出力
- 接続プールでリソース最適化

参考リンク・資料

公式ドキュメント

Oracle JDBC Documentation
https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/

Java Platform SE 8 API
java.sqlパッケージの詳細仕様

JDBCドライバー

PostgreSQL JDBC Driver
https://jdbc.postgresql.org/

MySQL Connector/J
https://dev.mysql.com/downloads/connector/j/

Oracle JDBC Driver
Oracle公式サイトからダウンロード

ライブラリ・フレームワーク

HikariCP
高性能コネクションプール

Spring Data JPA
JPA活用の次のステップ

MyBatis
SQLマッパーフレームワーク

次の学習ステップ

- **Spring Boot + JPA** - エンタープライズ開発
- **MyBatis実践** - SQLを活かした開発
- **データベース設計** - 効率的なスキーマ設計
- **パフォーマンスチューニング** - 大規模対応