JDBC i JPA

Jakub Szlenk

JDBC – podstawowy mechanizm obsługi baz danych w Javie

- JDBC to uniwersalne API, które pozwala w jednolity sposób korzystania z różnych systemów baz danych
- JDBC jest zorientowane głównie na relacyjne bazy danych

JDBC – podstawowy mechanizm obsługi baz danych w Javie

- Aby skorzystad z JDBC we własnej aplikacji należy pobrać konkretną implementację, specyficzną dla danego systemu baz danych
- W przypadku bazy danych MySQL mamy do czynienia ze sterownikiem MySQL Connector Java
- Jest to pojedynczy plik JAR, który należy dodad do swojej aplikacji

JDBC – podstawowy mechanizm obsługi baz danych w Javie

Zalety:

- jedno API do wszystkich baz danych
- szybkość działania
- prosta architektura

JDBC – podstawowy mechanizm obsługi baz danych w Javie

Wady:

- zapytania wywoływane przy użyciu JDBC są nadal specyficzne dla używanego silnika bazodanowego
- przy skomplikowanych projektach używanie samego JDBC powoduje konieczność tworzenia dużej ilości kodu

JDBC – podstawowy mechanizm obsługi baz danych w Javie

Podstawowe elementy API JDBC (java.sql):

- Connection – interfejs reprezentujący połączenie z bazą danych. Pozwala na wykonywanie zapytao SQL i kontrolę transakcji
- Statement – reprezentuje pojedyncze zapytanie. Pozwala na pozyskanie informacji o wynikach (liczba zmodyfikowanych rekordów/zbiór wyników)

JDBC – podstawowy mechanizm obsługi baz danych w Javie

Podstawowe elementy API JDBC (java.sql):

- ResultSet –reprezentuje zbiór wyników, pobranych np. zapytania SELECT
- PrepareStatement – reprezentuje zapytanie preparowane,tj. takie, które można wywoływad wielokrotnie z różnymi wartościami parametrów
- CallableStatement – reprezentuje procedurę składowaną, o ile są one obsługiwane przez dany silnik bazodanowy

JDBC – podstawowy mechanizm obsługi baz danych w Javie

Podstawowe elementy API JDBC (java.sql):

- Driver – reprezentuje pojedynczą instancję sterownika, wczytanego dynamicznie, najczęściej z biblioteki dołączonej do projektu
- ResultSetMetaData – informacje: kolumny zbioru wyników

JDBC – podstawowy mechanizm obsługi baz danych w Javie

- Przykład (rejestracja sterownika i nawiązanie połączenia):

```
try {
Driver sterownik = new com.mysql.jdbc.Driver();
DriverManager.registerDriver(sterownik);
Connection conn =
sterownik.connect("jdbc:mysql://localhost/szkolenie?user=sa&amp;amp;
password=", null);
} catch (SQLException e) {
e.printStackTrace();
}
```

JDBC – podstawowy mechanizm obsługi baz danych w Javie

- Przykład (rejestracja sterownika i nawiązanie połączenia) cd:

```
Statement stmt = conn.createStatement();
if (stmt.execute("SELECT * from produkt")) {
ResultSet rs = stmt.getResultSet();
while (rs.next())
out.println(rs.getString("nazwa"));
}
```

Separacja kodu obsługi JDBC od serwletów

- Najprostsze zastosowanie JDBC polega na umieszczeniu kodu obsługi JDBC wewnątrz serwletów
- Takie rozwiązanie narusza jednak zasadę separacji zagadnień – serwlet jako element warstwy kontrolera (czasem również widoku) nie powinien zawierać kodu obsługi danych

Separacja kodu obsługi JDBC od serwletów

- W związku z tym, cały kod wykonujący operacje z wykorzystaniem JDBC API powinien być umieszczony w odrębnych klasach, tak, aby programista serwletów wykonywał jedynie ogólne operacje na danych, a nie operacje JDBC (np. dodajUzytkownika() zamiast statement.execute(...))

Separacja kodu obsługi JDBC od serwletów

- Dobrym i prostym testem na sprawdzenie, czy kod obsługi JDBC i serwletów jest odpowiedź na pytanie: Czy jeśli zamiast sterownika JDBC jako źródło danych zostanie użyty np. plik XML, to kody serwletów ulegną istotnej zmianie?
- Jeśli odpowiedź jest przecząca, to znaczy, że separacja została przeprowadzona prawidłowo

JPA - struktura i zasady działania

- JPA – Java Persistence API – standard określający zasady wykorzystywania danych relacyjnych przy użyciu obiektów w języku Java
- JPA powstało na bazie istniejących wcześniej rozwiązań ORM (object-relational mapper), takich jak Hibernate

JPA - struktura i zasady działania

- JPA jest niezależny od konkretnego systemu baz danych, podobnie jak JDBC
- JPA funkcjonuje na znacznie wyższym poziomie abstrakcji, niż JDBC – na ogół nie wykorzystuje się przy jego użyciu zapytao w natywnych językach SQL

JPA - struktura i zasady działania

- JPA pozwala na operowanie danymi na poziomie encji – wykonanie standardowych operacji (C-reate, R-ead, U-pdate, D-elete) polega na wywołaniu prostych metod
- Ponadto, JPA udostępnia własny język zapytań, JPQL, który jest również niezależny od silnika bazodanowego (!)L

JPA - struktura i zasady działania

- **Kluczowe elementy JPA:**

- Klasa Persistence: pozwala na tworzenie fabryk menedżerów encji na podstawie danych z pliku konfiguracyjnego
- Interfejs EntityManagerFactory: fabryka menedżerów encji, działająca na podstawie danych z pliku konfiguracyjnego (jednostki utrwalania)

JPA - struktura i zasady działania

- **Kluczowe elementy JPA:**

- Interfejs EntityManager – centralny składnik JPA, wykorzystywany do wykonywania operacji na encjach, realizacji zapytań JPQL i delegowania zadań dot. transakcji
- Interfejs Query – reprezentuje pojedyncze zapytanie (zarówno pobierające, jak i modyfikujące)

JPA - struktura i zasady działania

- **Kluczowe elementy JPA:**

- Interfejs EntityTransaction – reprezentuje pojedynczą transakcję JPA
- Poza wymienionymi elementami, kluczową rolę w JPA odgrywają adnotacje, które służą do oznaczania klas encji w celu prawidłowego mapowania ich na relacje w bazie danych

Adnotacje JPA jako sposób oznaczania klas encji

- Jeszcze kilka lat temu przeważającym podejściem w deklarowaniu informacji wiążących klasy POJO (Plain Old Java Object) z relacjami baz danych było użycie pliku XML
- Aktualnie wszelkie informacje potrzebne do wykonania prawidłowego mapowania na ogół deklaruje się za pomocą adnotacji dla klas, pól i metod

Adnotacje JPA jako sposób oznaczania klas encji

- **Najważniejsze adnotacje**

- dla klas – @Entity, @Table – pozwalają na oznaczenie klasy jako klasy encji i powiązanie jej z konkretną relacją w bazie danych
- dla pól/metod – @Id, @Column, @Basic, @GeneratedValue – pozwalają na określanie podstawowych informacji o powiązaniach kolumna-pole

Adnotacje JPA jako sposób oznaczania klas encji

- **Najważniejsze adnotacje**

- dla związków pomiędzy relacjami – @JoinColumn, @OneToOne, @OneToMany, @ManyToMany
- dla relacji dziedziczenia, występującej pomiędzy klasami Java – @Inheritance, @PrimaryKeyJoinColumn
- dla reprezentowania złożonych kluczy głównych: @EmbeddedId, @IdClass

Wdrożenie JPA w aplikacji webowej

- Tradycyjnie JPA stosuje się w aplikacjach biznesowych, gdzie zarządzaniem menedżerami encji zajmuje się serwer
- W takiej sytuacji programista korzysta jedynie z interfejsów EntityManager, Query, EntityTransaction

Wdrożenie JPA w aplikacji webowej

- W aplikacjach webowych programista musi sam pozyskać niezbędne obiekty i sam zarządza menedżerami encji
- Istnieją różne koncepcje rozwiązania tego problemu, m.in. utworzenie fabryki menedżerów encji przy starcie aplikacji webowej (ServletContextListener!)

Wdrożenie JPA w aplikacji webowej

- Najważniejsze, aby ilość kodu wymagana do pozyskania obiektu menedżera encji była jak najmniejsza oraz nie występowały problemy z dostępem współbieżnym (kontekst aplikacji!)
- W ten sposób, w zakresie wykorzystywania danych można osiągnąd dużą częśd możliwości aplikacji biznesowej