Programowanie aplikacji w Java

Maciej Gowin

Zjazd 6 - dzień 2

Linki

Opis

https://maciejgowin.github.io/wsb-java/

Kod źródłowy przykładów oraz zadań

https://github.com/MaciejGowin/wsb-programowanie-aplikacji-java

Hibernate w praktyce: konfiguracja

Punktem wejściowym konfiguracji Hibernate jest plik konfiguracyjny zawierający podstawowe informacje wymagane do uruchomienia kontekstu Hibernate.

Podstawowy plik konfiguracyjny zawiera:

- szczegóły połączenia wraz z informacjami o typie bazy danych
- referencje do plików definiujących sposób mapowania danych

Istnieją 2 typy formatów dla plików konfiguracyjnych: `properties` oraz `xml`, odpowiednio: `hibernate.properties` lub `hibernate.cfg.xml`.

```
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE hibernate-configuration SYSTEM
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>

        <property name = "hibernate.dialect">
            org.hibernate.dialect.MySQLDialect
        </property>

        <property name = "hibernate.connection.driver_class">
            com.mysql.jdbc.Driver
        </property>

        <property name = "hibernate.connection.url">
            jdbc:mysql://localhost/booking_system
        </property>

        <property name = "hibernate.connection.username">
            root
        </property>

        <property name = "hibernate.connection.password">
            root
        </property>

        <mapping resource = "Airport.hbm.xml"/>

    </session-factory>
</hibernate-configuration>
```

Hibernate w praktyce: konfiguracja mapowań

Wraz z głównym plikiem konfiguracyjnym, dostarczamy również referencje do plików definiujących konfigurację mapowań - połączeń pomiędzy klasami Java a tabelami baz danych.

Hibernate w praktyce: konfiguracja mapowań

```
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD//EN"
        "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
    <class name = "pl.wsb.programowaniejava.maciejgowin.przyklad55.Airport" table = "airports">
        <id name = "code" type = "string" column = "code" />
        <property name = "name" column = "name" type = "string"/>
        <property name = "latitude" column = "latitude" type = "double"/>
        <property name = "longitude" column = "longitude" type = "double"/>
    </class>
</hibernate-mapping>
```

Hibernate w praktyce: konfiguracja typów mapowań

Hibernate definiuje typy mapowań, czyli to, w jaki sposób typ danych klasy zostanie przekonwertowany na typ danych w tabeli.

Typ mapowania - typ Java - typ ANSI SQL
integer - java.lang.Integer - INTEGER
long - long or java.lang.Long - BIGINT
short - short or java.lang.Short - SMALLINT
float - float or java.lang.Float - FLOAT
double - double or java.lang.Double - DOUBLE
big_decimal - java.math.BigDecimal - NUMERIC

Hibernate w praktyce: konfiguracja typów mapowań

Typ mapowania - typ Java - typ ANSI SQL

character - java.lang.String - CHAR(1)

string - java.lang.String - VARCHAR

byte - byte or java.lang.Byte - TINYINT

boolean - boolean or java.lang.Boolean - BIT

yes/no - boolean or java.lang.Boolean - CHAR(1) ('Y' or 'N')

true/false - boolean or java.lang.Boolean - CHAR(1) ('T' or 'F')

Hibernate w praktyce: konfiguracja typów mapowań

Typ mapowania - typ Java - typ ANSI SQL

date - java.util.Date or java.sql.Date - DATE

time - java.util.Date or java.sql.Time - TIME

timestamp - java.util.Date or java.sql.Timestamp - TIMESTAMP

calendar - java.util.Calendar - TIMESTAMP

calendar_date - java.util.Calendar - DATE

Hibernate w praktyce: klasa Configuration

Konfiguracja jest inicjalizowana poprzez stworzenie instancji klasy `Connection`. Odbywa się to przeważnie jednorazowo, zaraz po uruchomieniu aplikacji. Hibernate będzie starał się wyszukać konfiguracji w ścieżce wyszukiwań, używając wspomnianych wcześniej domyślnych nazw plików.

```
new Configuration().configure()
```

Wywołanie metody `configure()` spowoduje załadowanie pliku konfiguracyjnego o domyślnej nazwie `hibernate.cfg.xml`.

Hibernate w praktyce: klasa SessionFactory

Po utworzeniu konfiguracji istotne jest utworzenie instancji `SessionFactory`, która finalizuje konfigurację kontekstu Hibernate oraz pozwala na tworzenie instancji `Session`.

Instancja `SessionFactory` jest obiektem skomplikowanym i jest tworzony zazwyczaj podczas uruchamiania aplikacji, a następnie ponownie wykorzystywany.

Dany obiekt `SessionFactory` odpowiada jednej, konkretnej bazie danych. Może być bezpiecznie używany w środowiskach wielowątkowych.

```
SessionFactory factory = new Configuration().configure().buildSessionFactory();
/* ... */
sessionFactory.openSession();
```

Hibernate w praktyce: klasa Session

Instancja `Session` definiuje fizyczne połączenie z bazą danych. Została zaprojektowana z myślą o możliwości tworzenie nowych obiektów tej klasy dla każdej operacji na bazie danych. Przechowywane obiekty są zapisywane oraz wyszukiwane przy użyciu tej klasy.

Obiekty `Session` nie powinny być przechowywany w pamięci zbyt długo. Powinny być tworzone oraz usuwane w razie konieczności.

```
Session session = sessionFactory.openSession();
/* ... */
session.save(airport);
```

Hibernate w praktyce: klasa Session

W kontekście `Session` każda instancja, która jest zapisywane, aktualizowana, usuwana czy też wyszukiwania w danym momencie może znajdować się w jednym z 3 stanów:

- `transient` - nowa instancja przechowywanej klasy, która nie jest związana z sesją, nie jest przechowywane w bazie danych oraz nie posiada przypisanego identyfikatora.
- `persistent` - instancja, która zostaje powiązana z sesją przechodzi ze stanu `transient` do `persistent`. Posiada reprezentacje w bazie danych, identyfikator oraz jest powiązana z sesją.
- `detached` - po zamknięciu sesji, instancje w stanie `persistent` stają się `detached`.

Java w praktyce: POJO

Plain Old Java Object (POJO) to proste typy bez referencji do jakiegokolwiek frameworku. Upraszczając, są to uproszczone kontenery danych.

Nie posiadają one opisanych reguł dotyczących konwencji nazewniczych oraz restrykcji dotyczących poziomów dostępu dla własności.

```
public class Customer {
    public String firstName;
    public String lastName;
    private LocalDate dateOfBirth;

    public LocalDate customerDateOfBirth() {
        return dateOfBirth;
    }
}
```

Z takim podejściem wiąże się kilka problemów, które wynikają z braku konwencji, która pomogłaby w lepszym wykorzystaniu tych klas.

Java w praktyce: JavaBean

W celu lepszego wykorzystania obiektów POJO wprowadzono pojęcie JavaBean, które rozszerza ideę POJO. JavaBean charakteryzują się ścisłymi zasadami:

- `poziomy dostępu` - pola klasy są prywatne, a dostęp do nich odbywa się przez tzw. settery i gettery.
- `nazwy metod` - metody setter oraz getter są tworzone na podstawie konwencji: setX i getX (lub setX i isX dla typu boolean), gdzie X to nazwa pola.
- `konstruktor domyślny` - bezargumentowy konstruktor domyślny musi być dostępny, dzięki czemu instancje klasy mogą zostać utworzone bez konieczności definiowania argumentów (np. podczas procesu serializacji i deserializacji).
- `serializowalność` - implementacja interfejsu `Serializable` pozwala na przechowywanie stanu.

Java w praktyce: JavaBean

```
public class Customer {

    private String firstName;
    private String lastName;
    private LocalDate dateOfBirth;

    public Customer() {
    }

    public Customer(String firstName, String lastName, LocalDate dateOfBirth) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.dateOfBirth = dateOfBirth;
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public LocalDate getDateOfBirth() {
        return dateOfBirth;
    }

    public void setDateOfBirth(LocalDate dateOfBirth) {
        this.dateOfBirth = dateOfBirth;
    }
}
```

Java w praktyce: encje

Klasy, których obiekty są zapisywane w tabelach bazy danych, są nazywane klasami persystentnymi (ang. persistent classes). Hibernate współpracuje z klasami, które używają podejścia POJO, w szczególności podejście JavaBean.

Klasy te nazywane są też często encjami (ang. entity), co nawiązuje do nomenklatury znanej z teorii baz danych.

Hibernate w praktyce: klasa Transaction

Transakcja, znana z systemów bazodanowych, definiuje jednostkę pracy. Hibernate posiada wbudowany mechanizm zarządzania transakcjami.

```
session.getTransaction().begin();
/* ... */
session.getTransaction().commit();
/* ... */
session.getTransaction().rollback();
```

Hibernate w praktyce: klasa Query

Obiekty typu `Query` używają zapytać SQL lub HQL (Hibernate QL, język specyficzny dla Hibernate) oraz pozwalają na wyszukiwanie danych. Pozwala na zdefiniowanie ograniczeń zapytania, wykonanie go oraz pobranie wyników.

```
Query<Airport> query = session.createQuery("FROM Airport", Airport.class);
List<Airport> list = query.list();
```

Programowanie: przykład 56

Pobieranie automatycznie wygenerowanego klucza głównego.

```
package pl.wsb.programowaniejava.maciejgowin.przyklad56;

import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

import java.time.LocalDate;
import java.util.Optional;

public class Main {

    public static void main(String[] args) {
        try (SessionFactory factory = new Configuration().configure().buildSessionFactory()) {

            CustomerManager customerManager = new CustomerManager(factory);

            // Delete customers
            customerManager.deleteCustomers();

            // List customers
            System.out.printf("Check (1): %s%n", customerManager.getCustomers());

            // Add customer
            Optional<Integer> newId = customerManager.addCustomer("Maciej", "Gowin", LocalDate.parse("1986-11-21"));

            System.out.printf("Check (2): %s%n", newId);

            // List customers
            System.out.printf("Check (3): %s%n", customerManager.getCustomers());
        } catch(Exception e) {
            System.out.printf("Failed: %s%n", e.getMessage());
        }
    }
}
```

Hibernate w praktyce: mapowanie przez adnotacje

Do tej pory definiowaliśmy konfiguracje mapowania poprze użycie definicji zawartej w plikach `xml`.

Wiąże się to z kilkoma niedogodnościami:

- utrzymywanie definicji mapowań odbywa się w zewnętrznym pliku,
- każda zmiana wymaga zmian w klasie, tabeli, jak i pliku xml,
- pliki xml mogą być duże oraz ciężkie do utrzymywania.

Rozwiązaniem może być użycie adnotacji, które zdefiniowane na klasie pozwalają na opisanie mapowania w łatwy sposób bez konieczności utrzymywanie zewnętrznego pliku.

Programowanie: przykład 57

Użycie adnotacji oraz konfiguracji poprzez plik `hibernate.properties`.

```
package pl.wsb.programowaniejava.maciejgowin.przyklad57;

import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

import java.time.LocalDate;
import java.util.Optional;

public class Main {

    public static void main(String[] args) {
        try (SessionFactory factory = new Configuration()
                .addAnnotatedClass(Customer.class)
                .buildSessionFactory()) {

            CustomerManager customerManager = new CustomerManager(factory);

            // Delete customers
            customerManager.deleteCustomers();

            // List customers
            System.out.printf("Check (1): %s%n", customerManager.getCustomers());

            // Add customer
            Optional<Integer> newId = customerManager.addCustomer("Maciej", "Gowin", LocalDate.parse("1986-11-21"));

            System.out.printf("Check (2): %s%n", newId);

            // List customers
            System.out.printf("Check (3): %s%n", customerManager.getCustomers());
        } catch(Exception e) {
            System.out.printf("Failed: %s%n", e.getMessage());
        }
    }
}
```

Hibernate w praktyce: plik properties

Format pliku `properties` operaty jest na liście kluczy i wartości. Zwyczajowo nazwy kluczy używają przestrzeni nazw oddzielone znakiem `.` choć nie jest to wymagane.

Silnik Hibernate będzie poszukiwał konkretnych, z góry ustalonych wartości podobnie jak to miało miejsce w przypadku pliku `xml`.

```
hibernate.dialect = org.hibernate.dialect.MySQLDialect
hibernate.connection.driver_class = com.mysql.jdbc.Driver
hibernate.connection.url = jdbc:mysql://localhost/booking_system
hibernate.connection.username = root
hibernate.connection.password = root
```

Hibernate w praktyce: konfiguracja

Do tej pory konfigurowaliśmy kontekst przy użyciu plików `hibernate.cfg.xml`. Jeżeli przeniesiemy konfiguracje do pliku `hibernate.properties` oraz zastąpimy konfigurację mapowania poprzez adnotacje plik `hibernate.cfg.xml` jest zbędny.

Aby pozbyć się automatycznego ładowanie tego pliku, wystarczy usunąć wywołanie metody `configure()`. Pozostałe wartości zostaną załadowane z pliku `hibernate.properties`.

Hibernate w praktyce: konfiguracja

Jeżeli używamy klas z adnotacjami, musimy wskazać silnikowi Hibernate, w jakich klasach znajdują się definicje mapowań. Możemy to zrobić, dodając każdą z klas z osobna:

```
new Configuration()
    .addAnnotatedClass(Customer.class)
    .buildSessionFactory()
```

Lub też poprzez dodanie całego pakietu:

```
new Configuration()
    .addPackage("com.mydomain.entities")
    .buildSessionFactory()
```

Hibernate w praktyce: adnotacje

Wszystkie z użytych adnotacji pochodzą z pakietu `javax.persistence`. Są to zatem elementy zdefiniowane przez standard JPA. Rolą Hibernate jest dostarczenie ich obsługi.

Użyliśmy kilku podstawowych adnotacji, w tym:

- @Entity
- @Table
- @Column
- @Id
- @GeneratedValue

Hibernate w praktyce: adnotacja @Entity

Adnotacja używana na poziomie klasy, która oznacza ją jako klasę encyjną, posiadającą konstruktor bezargumentowy o zasięgu przynajmniej `protected`.

Hibernate w praktyce: adnotacja @Table

Adnotacja używana na poziomie klasy, która pozwala na specyfikację szczegółów dotyczących tabeli, która zostanie użyta do zapisania danych encji.

Dostarcza 5 atrybutów:

- `name` - nazwa tabeli,
- `catalog` - definiuje katalog,
- `schema` - definiuje schemat,
- `uniqueConstraints` - definicja kolumn, których wartości są unikatowe,
- `indexes` - definiuje indeksy.

Hibernate w praktyce: adnotacja @Column

Adnotacja oznacza daną kolumnę w tabeli oraz sposób odwzorowania danego pola.

Dostarcza szereg atrybutów, w tym:

- `name` - nazwa kolumny zdefiniowana explicite, domyślnie nazwa pola,
- `length` - definicja rozmiaru dla danego typu, np. długości ciągu znaków,
- `nullable` - oznaczenie czy wartości dla kolumny mogą przyjmować wartość `null`,
- `unique` - znaczenie wartości dla kolumny, które są unikatowe.

Hibernate w praktyce: adnotacja @Id

Adnotacja oznacza dane pole/kolumnę jako klucz główny. Klucz główny może składać się z jednego lub więcej pól w zależności od definicji tabeli.

Domyślnie adnotacje ta postara się dobrać, jak najlepszą strategię dla generowanych klucz głównych. Zachowanie to może zostać nadpisane przy użyciu `@GeneratedValue`.

Hibernate w praktyce: adnotacja @GeneratedValue

Definiuje strategię dla generowanych kluczy głównych.

Istnieją 4 strategie: TABLE, SEQUENCE, IDENTITY oraz AUTO, przy czym my skupimy się na strategi `IDENTITY`. Opiera się ona na kluczach generowanych na poziomie bazy (na podstawie specjalnych wpisów w bazie).

> Nie każdy baza danych pozwala na użycie każdej ze strategi.

Hibernate w praktyce: automatyczne generowanie struktury

Hibernate pozwala na automatyczne generowanie schematu tabel bazy danych na podstawie klas i adnotacji. Aby wygenerować schemat, należy dodać tę informację do pliku konfiguracyjnego:

```
hibernate.hbm2ddl.auto = update
```

Hibernate w praktyce: automatyczne generowanie struktury

Istnieje kilka opcji konfiguracyjnych dla `hibernate.hbm2ddl.auto`.

- `validate` - sprawdza poprawność schematu, nie zmianie bazy. validate the schema, makes no changes to the database.
- `update` - aktualizuje schemat.
- `create` - tworzy schemat, usuwając poprzednie dane.
- `create-drop` - usuwa schemat po zamknięciu sesji, przeważnie na końcu działania programu.
- `none` - nie zmienia schematu, nie zmienia bazy.

Programowanie: przykład 58

Automatyczne generowanie struktury.

```
package pl.wsb.programowaniejava.maciejgowin.przyklad58;

import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

import static java.time.LocalDateTime.parse;

public class Main {

    public static void main(String[] args) {
        try (SessionFactory factory = new Configuration()
                .addAnnotatedClass(Flight.class)
                .buildSessionFactory()) {

            FlightManager flightManager = new FlightManager(factory);

            flightManager.deleteFlights();

            System.out.printf("Check (1): %s%n", flightManager.getFlights());
            System.out.printf("Check (2): %s%n", flightManager
                    .addFlight("DUB", "WRO", parse("2022-01-14T17:15"), parse("2022-01-14T19:45")));
            System.out.printf("Check (3): %s%n", flightManager
                    .addFlight("DUB", "KRK", parse("2022-01-20T11:30"), parse("2022-01-20T12:55")));
            System.out.printf("Check (4): %s%n", flightManager
                    .addFlight("WRO", "DUB", parse("2022-01-20T08:00"), parse("2022-01-20T09:15")));
            System.out.printf("Check (5): %s%n", flightManager.getFlights());
        } catch(Exception e) {
            System.out.printf("Failed: %s%n", e.getMessage());
        }
    }
}
```

Hibernate w praktyce: Session i zapisywanie

Nowo utworzony obiekty przyjmuje stan transient. Poprzez przypisanie go do sesji zmienia stan na persistent.

```
Customer customer = new Customer("Maciej", "Gowim");
Integer id = (Integer) session.save(customer);
```

Jeżeli klucz jest kluczem generowanym, zostanie on utworzony podczas wywołania metody.

Użyliśmy metody `save()`. Dostępna jest też metoda `persist()` jednak jej zachowanie jest nieco inne.

Hibernate w praktyce: Session i zapisywanie

persist()

Zmienia stan z transient na persistent, aczkolwiek nie gwarantuje, że identyfikator zostanie przypisany od razu. Stanie się to dopiero podczas synchronizacji sesji ze stanem bazy danych (automatycznie po wykonaniu transakcji lub po wywołaniu metody `flush()`.

Metoda ta gwarantuje też, że nie zostanie wywołane wyrażenie INSERT, jeżeli nie działamy w zakresie transakcji.

save()

Gwarantuje zwrócenie identyfikatora. Jeżeli wyrażenie INSERT musi zostać wywołane do wygenerowanie identyfikatora, zostanie ono wykonane od razu, nie zważając na fakt, czy dzieje się to w zakresie lub poza zakresem transakcji.

Hibernate w praktyce: Session i ładowanie

Metoda `load()` pozwala na pobranie obiektów, jeżeli znany jest ich identyfikator. Ładuje obiekt, ustawiając jego stan na persistent.

```
Customer customer = session.load(Customer.class, 1);
```

Możliwe jest też załadowanie wartości do danego obiektu.

```
Customer customer = new Customer();
session.load(customer, 1);
```

Hibernate w praktyce: Session i ładowanie

Należy zwrócić uwagę na fakt, że metoda ta zgłasza wyjątek w przypadku braku pasującego wiersza w bazie.

W przypadku, gdy nie jesteśmy pewni czy dany wiersz istnieje w bazie powinniśmy użyć metody `get()`. Pobiera ona element natychmiastowo oraz zwraca `null` w przypadku braku wiersza.

```
Customer customer = session.get(Customer.class, 1);
```

Hibernate w praktyce: Session i ładowanie

Istnieje możliwość ponownego załadowanie obiektów w dowolnym momencie za pomocą metody `refresh()`.

```
Customer customer = new Customer("Maciej", "Gowin");
session.save(customer);
session.flush();
session.refresh(customer);
```

Hibernate w praktyce: Session i modyfikacja persistent

Obiekty transakcyjne w stanie persistent (np. załadowane z bazy danych lub wyszukane w kontekście sesji) mogą zostać dowolnie zmodyfikowane, a zmiany zostaną zapisane na bazie danych, gdy sesja zostanie wywołana (np. poprzez metodę `flush` lub zakończenie transakcji).

W takim przypadku nie jest wymagane wywołanie metody `update()` odpowiedzialnej za aktualizacje wartości w bazie.

Hibernate w praktyce: Session i modyfikacja obiektów persistent

Wynika z tego, że najprostszym sposobem modyfikacji obiektów jest ich uprzednie załadowanie, zmienienie wartości w kontekście sesji.

```
Customer customer = session.load(Customer.class, 1);
customer.setFirstName("Piotr");
session.flush();
```

> W niektórych przypadkach takie operacje mogą być niewydajne, ponieważ konieczne jest załadowanie obiektu.

Hibernate w praktyce: Session i modyfikacja detached

W przypadku, gdy chcemy zaktualizować obiekt w stanie detached (np. obiekt załadowany przez jedną sesję, zmieniony poza jej kontekstem, zapisywany przez inna sesję)
do zapisu możemy użyć metod `update()` oraz `merge()`.

```
Customer customer = session1.get(Customer.class, 1);

// poza zasięgiem transakcji
customer.setFirstName("Piotr");

session2.update(customer);
```

Hibernate w praktyce: Session i modyfikacja detached

Wywołanie `update()` może zostać zamienione wywołaniem `merge()`, jednak ich działanie nieznacznie się różni.

update()

W przypadku, gdy w kontekście sesji będzie istniał jakikolwiek obiekt w stanie persistent o takim identyfikatorze, zostanie zwrócony błąd.

merge()

Zmiany zostaną zawsze wykonane, nie biorąc pod uwagę obecnego stanu sesji.

Hibernate w praktyce: Session i detekcja stanu

Metoda `saveOrUpdate` pozwala na automatyczne sprawdzenie stanu obiektu oraz:

- zapisuje obiekt poprzez wygenerowanie identyfikatora lub
- aktualizuje obiekt powiązany z danym identyfikatorem.

```
Customer existingCustomer = session1.load(Customer.class, 1);
existingCustomer.setFirstName("Piotr");

Customer newCustomer = new Customer("Jan", "Nowak");

session2.saveOrUpdate(existingCustomer); // aktualizacja istniejącego (identyfikator zdefiniowany)
session2.saveOrUpdate(newCustomer); // zapisanie nowego (brak identyfikatora)
```

Hibernate w praktyce: Session i usuwanie

Metoda `delete()` pozwala na usunięcie obiektu z kontekstu sesji. Zmienia jego stan na transient.

```
Customer existingCustomer = session.load(Customer.class, 1);
session2.delete(existingCustomer);
```

Programowanie: zadanie 28

Przetestuj działanie metod: save, persist, load, get, update, merge, saveOrUpdate oraz delete.