Internet Applications Design and Implementation

2015 - 2016 - 1st edition

(Lecture 9 - Data Abstraction - Part 2)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Costa Seco (joao.seco@fct.unl.pt)
Jácome Cunha (jacome@fct.unl.pt)

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Data Abstraction layers

- Advantages
  - Hides the complexity of a particular query language
  - Allows the portability of database engines (not really models)
  - Prevents attacks such as SQL injection, or XSS
  - Avoids runtime errors in the construction of queries
  - May isolate efficient implementations
    (previously done with prepared and compiled parameterised SQL queries)
  - Allows scalability via customised runtime configurations
    (distribution, transactional behaviour, indexing, …)
  - Avoids early optimization pitfalls, develop first, configure later.
- Pitfalls
  - May lead to inefficient data transmissions: more queries (N+1 queries), more data than needed.

Spring Data (and other ORM Implementations)

- Simple Declaration of generic queries
- Specific declaration of custom queries (JPQL)
- Richer Behaviour from Repositories (e.g. paged)

```
public interface StudentRepository extends PagingAndSortingRepository<Student, Long> {

    List<Student> findByName(String name);

    @Query("select s from Student s where s.name like CONCAT(?,'%')")
    List<Student> search(String name);

    Page<Student> findByName(String name, Pageable pageable);
}
```

- Dependency Injection and component assembly

```
public class StudentController {

    @Autowired
    StudentRepository students;
```

Spring Data (and other ORM Implementations)

- Simple Declaration of generic queries
- Specific declaration of custom queries (JPQL)
- Richer Behaviour from Repositories (e.g. paged)

```
public interface StudentRepository extends PagingAndSortingRepository<Student, Long> {

    List<Student> findByName(String name);
```

```
    @RequestMapping(value= "/page")
    public @ResponseBody Page<Student> getStudentsPaged(
            @RequestParam(required=false, defaultValue =  "0") Integer page,
            @RequestParam(required=false, defaultValue = "3") Integer size) {

        return students.findAll(new PageRequest(page, size));
    }
```

```
public class StudentController {

    @Autowired
    StudentRepository students;
```

Spring Data (and other OF

- Simple Declaration of gener

```
@RequestMapping(value= "/page")
public @ResponseBody Page<Student> getStuden
        @RequestParam(required=false, default
        @RequestParam(required=false, default

    return students.findAll(new PageRequest(p
}
```

```
    List<Student> findByName(String name);

  @Query("select s from Student s where s.na
    List<Student> search(String name);

    Page<Student> findByName(String name, Pag
}
```

- Dependency Injection and c

```
public class StudentController {

    @Autowired
    StudentRepository students;
```

```
 "content": [
    {
      "id": 1,
      "name": "Ingrid Daubechies",
      "age": 19
    },
    {
      "id": 2,
      "name": "Jacqueline K. Barton",
      "age": 18
    },
    {
      "id": 3,
      "name": "Jane Goodall",
      "age": 20
    }
 ],
 "last": false,
 "totalElements": 6,
 "totalPages": 2,
 "size": 3,
 "number": 0,
 "sort": null,
 "first": true,
 "numberOfElements": 3
}
```

JPA Associations

```java
@Entity
@Table(name = "book_category")
public class BookCategory {
    private int id;
    private String name;
    private Set<Book> books;

    public BookCategory(){ … }

    public BookCategory(String name) {…}

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO
    public int getId() { … }

    public void setId(int id) { … }

    public String getName() {…}

    public void setName(String name) {…}

    @OneToMany(mappedBy = "bookCategory",
               cascade = CascadeType.ALL)
    public Set<Book> getBooks() { … }

    public void setBooks(Set<Book> books) { … }
}
```

```java
@Entity
public class Book{
    private int id;
    private String name;
    private BookCategory bookCategory;

    public Book() {}

    public Book(String name) { this.name = name;}

    public Book(String name, BookCategory bookCatego

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    public int getId() { … }

    public void setId(int id) { … }

    public String getName() { … }

    public void setName(String name) { … }

    @ManyToOne
    @JoinColumn(name = "book_category_id")
    public BookCategory getBookCategory() { … }

    public void setBookCategory(BookCategory bookCat
}
```

Eager and Lazy

- Evaluation modes - transferring objects to memory

```
@OneToMany(mappedBy = "course", cascade = CascadeType.ALL, fetch = FetchType.EAGER)
private Set<Enrollment> enrollments;
```

- Eager: transfers all objects related to the root

```
@OneToMany(mappedBy = "course", cascade = CascadeType.ALL, fetch = FetchType.Lazy)
private Set<Enrollment> enrollments;
```

- Lazy: transfers object when needed
  - Needs a transactional environment

```
@Component
public class ProfessorService {

    @Autowired
    CourseRepository courseRepository;

    @Autowired
    ProfessorRepository professors;

    @Transactional
    public void addCourses(String name, String... courses) {
        Professor p = professors.findByName(name);

        Set<Course> cs = p.getCourses();
        for(String c: courses)
            cs.add(courseRepository.findByName(c).get(0));
        professors.save(p);
    }
```

Many To Many

```java
@Entity
public class Course {

    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private long id;

    private String name;

    private int credits;

    @OneToMany(mappedBy = "course",
               cascade = CascadeType.ALL,
               fetch = FetchType.EAGER)
    private Set<Enrollment> enrollments;

    @ManyToMany(mappedBy = "courses")
    private Set<Professor> professors;
}
```

```java
@Entity
public class Professor {

    public Professor(String name) {
        this.setName(name);
    }

    public Professor() {}

    @javax.persistence.Id
     @GeneratedValue(strategy=GenerationType.A
    private long Id;

    private String name;

    @ManyToMany
    private Set<Course> courses;
}
```

creates *professor_courses* table in a uni-directional relation

https://en.wikibooks.org/wiki/Java_Persistence/ManyToMany

http://www.objectdb.com/java/jpa/getting/started

https://hellokoding.com/jpa-many-to-many-extra-columns-relationship-mapping-example-with-spring-boot-maven-and-mysql/

N+1 query problem

```java
@Entity
@Table(name = "book_category")
public class BookCategory {
    private int id;
    private String name;
    private Set<Book> books;

    public BookCategory(){ … }

    public BookCategory(String name) {…}

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    public int getId() { … }

    public void setId(int id) { … }

    public String getName() {…}

    public void setBooks(Set<Book> books) { … }
}
```

```java
@Entity
public class Book{
    private int id;
    private String name;
    private BookCategory bookCategory;

    public Book() {}

    public Book(String name) { this.name = name;}

    public Book(String name, BookCategory bookCategory) { … }

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    public int getId() { … }

    public void setId(int id) { … }

    public String getName() { … }
}
```

```java
@Repository
public interface BookCategoryRepository extends JpaRepository<BookCategory, Integer>{
}
```

```java
for( BookCategory c: categoryRepo.findAll() )
  for( Book b: c.books )
    …
```

```
select * from book_category;
select * from book where bookCategory = ?
…
select * from book where bookCategory = ?
```

N+1 query problem

```java
@Entity
@Table(name = "book_category")
public class BookCategory {
    private int id;
    private String name;
    private Set<Book> books;

    public BookCategory(){ … }

    public BookCategory(String name) {…}

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    public int getId() { … }

    public void setId(int id) { … }

    public String getName() {…}

    public void setBooks(Set<Book> books) { … }
}
```

```java
@Entity
public class Book{
    private int id;
    private String name;
    private BookCategory bookCategory;

    public Book() {}

    public Book(String name) { this.name = name;}

    public Book(String name, BookCategory bookCategory) { … }

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    public int getId() { … }

    public void setId(int id) { … }

    public String getName() { … }

    Category(BookCategory bookCategory) {…}
```

```java
@Repository
public interface BookCategoryRepository extends JpaRepository<BookCategory, Integer>{
}
```

```java
for( Book b : BookRepo )
  System.out.println( b.toString() + b.bookCategory.toString())
```

```
select * from book;
select * from bookCategory where bookCategory = ?
…
select * from bookCategory where bookCategory = ?
```

Custom queries for efficient execution

- ORM relations can produce a large number of queries… which can be optimized by means of a single query being dispatched to the DB

```
@Query(" select s from Student s "+
     "   inner join s.courses en "+
     "   inner join en.course c "+
     " where c.name = :name")
List<Student> searchByCourse(@Param("name") String name);
```

- Optimization may work by summarising data

```
@Query("select new ciai.model.StudentSummary(s.name,s.age) from Student s")
List<StudentSummary> summaryStudents();
```

Java Persistence Query Language (JPQL ⊆ HQL)

- Simple custom queries

```
@Query("select s from Student s where s.name like CONCAT(?,'%')")
List<Student> search(String name);
```

- Complex custom queries

```
@Query(" select s from Student s "+
       "   inner join s.courses en "+
       "   inner join en.course c "+
       " where c.name = :name")
List<Student> searchByCourse(@Param("name") String name);
```

- Safe and Secure query construction
  (both syntax errors, and also SQL-injection attacks)
- Language integration with object in queries

```
@Query("select new StudentSummary(s.name,s.age) from Student s")
List<StudentSummary> summaryStudents();
```

```
@Query("SELECT b FROM Book b INNER JOIN b.categories c WHERE c IN (:categories)")
List<Book> findByCategories(@Param("categories") Collection<Category> categories);
```

http://www.objectdb.com/java/jpa/getting/started

Caching

```
@Cacheable("books")
public Book findBook(ISBN isbn) {...}
```

```
@CacheEvict(cacheNames="books", allEntries=true)
public void loadBooks(InputStream batch)
```