Thao tác với CSDL và ORM

Mục tiêu

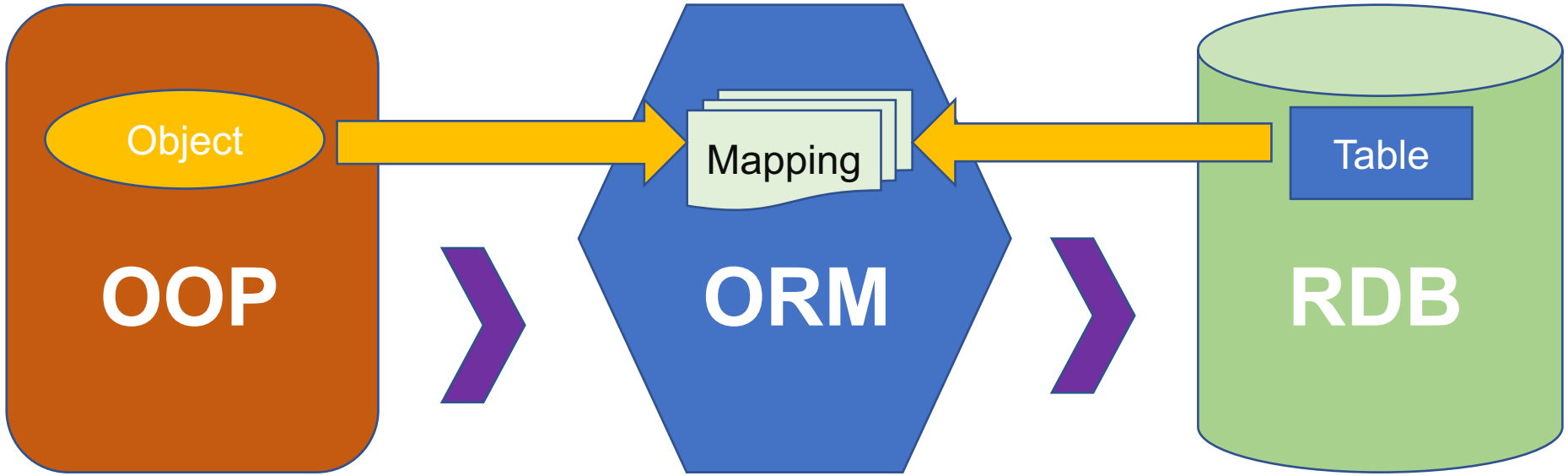
- Trình bày được khái niệm ORM
- Trình bày được JPA
- Trình bày được Entity
- Trinh bày được Entity Manager
- Trình bày được hoạt động của Spring Data JPA
- Triển khai được Spring Data JPA để thao tác cơ bản với CSDL

Thảo luận

ORM

ORM

- ORM (Object-Relational Mapping) là kỹ thuật liên kết giữa các đối tượng trong lập trình với các đối tượng trong CSDL
- ORM cho phép truy xuất dễ dàng đến dữ liệu thông qua các đối tượng lập trình
- ORM giúp lập trình viên tập trung thao tác với các đối tượng, không cần quá quan tâm đến CSDL thực tế đang dùng



Các góc nhìn khác nhau về dữ liệu

- Góc nhìn từ quản trị dữ liệu

```
CREATE TABLE persons (
            id integer NOT NULL,
            name varchar(50) NOT NULL,
            salary float, PRIMARY KEY(id)
);
INSERT INTO persons (id, name) VALUES (1, 'John
Doe');
UPDATE persons SET salary=2000 WHERE id=1;
```

- Góc nhìn từ lập trình viên

```
public class Person {
   public String name;
   public float salary;
   public Person(String name) { ... }
}
Person p = new Person("John Doe");
PersistenceLayer.save(p);
p.setSalary(2000);
PersistenceLayer.update(p);
```

Ưu điểm và nhược điểm của ORM

- Ưu điểm:
  - Quản lý dữ liệu tập trung trong code
  - Các thao tác với dữ liệu được thực hiện tự động
  - Tránh được các lỗi cú pháp SQL
  - Hỗ trợ giao dịch (transaction)
  - Có thể cache dữ liệu để truy xuất nhanh hơn
- Nhược điểm:
  - Đối với các dự án lớn, hiệu năng truy xuất dữ liệu thường hạn chế
  - Lập trình viên dễ bị rơi vào bẫy truy xuất dữ liệu quá nhiều (vì quá dễ để truy xuất), ảnh hưởng đến hiệu năng của hệ thống
  - Đối với các thao tác phức tạp, có thể cần đến việc sử dụng SQL thuần

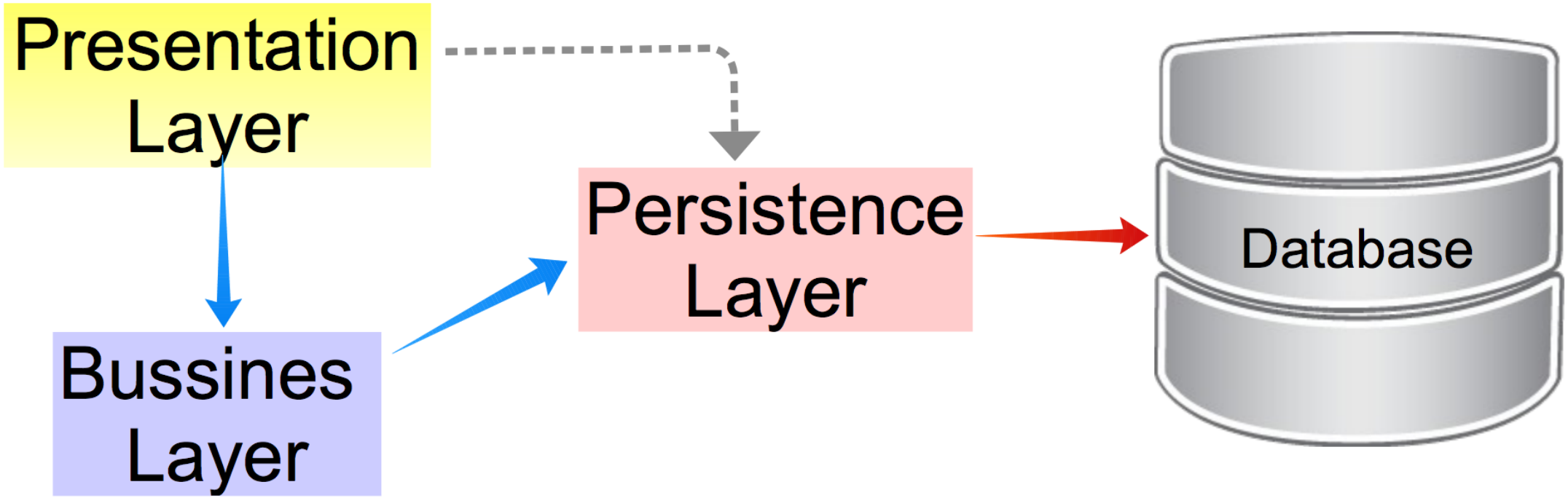
Một số ORM framework dành cho Java

- Apache Cayenne
- ActiveJDBC
- EclipseLink
- Ebean
- OpenJPA
- Hibernate
- JPA (Java Persistence API)

Thảo luận

JPA – Java Persistence API

Persistence Layer

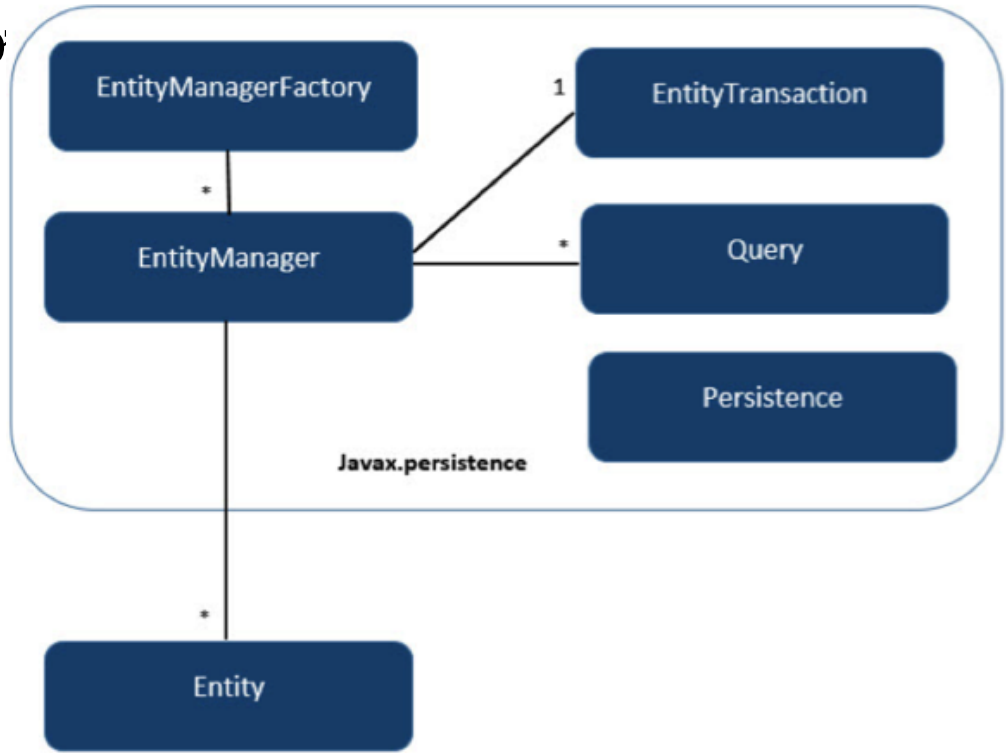


Entity

- Entity là đối tượng đại diện cho dữ liệu ở trong ứng dụng
- Entity thường là POJO (Plain Old Java Object)
- Entity sẽ được ánh xạ (mapping) tới một bảng trong CSDL
- Trong JPA, một entity cần tuân thủ:
  - Được gắn với annotation javax.persistence.Entity
  - Có một constructor là public và không có tham số
  - Không được khai báo final
  - Các biến đối tượng cần được khai báo là private, protected hoặc ở mức package-private

Persistence Context & Entity Manager

- Persistence Context là tập các thể hiện của entity được quản lý, tồn tại trong một kho dữ liệu
- Interface EntityManager:
  - Khai báo các phương thức để tương tác với
  - Tạo hoặc xoá các thể hiện của entity
  - Tìm kiếm entity theo khoá chính
  - Thực thi các câu lệnh truy vấn lên entity



Quản lý entity

- Container-managed Entity Manager:

```
@PersistenceContext
EntityManager em;
```

- Application-managed Entity Manager:

```
@PersistenceUnit
EntityManagerFactory emf;

EntityManager em = emf.createEntityManager();
```

Tìm một entity

- Ví dụ:

```
@PersistenceContext
EntityManager em;
public void enterOrder(int custID, Order newOrder) {
   Customer cust = em.find(Customer.class, custID);
   cust.getOrders().add(newOrder);
   newOrder.setCustomer(cust);
}
```

Lưu trữ entity

- Ví dụ:

```
@PersistenceContext
EntityManager em;
...
public LineItem createLineItem(Order order, Product product, int quantity) {
    LineItem li = new LineItem(order, product, quantity);
    order.getLineItems().add(li);
    em.persist(li);
    return li;
}
```

Xoá entity

- Ví dụ:

```
public void removeOrder(Integer orderId) {
    try {
        Order order = em.find(Order.class, orderId);
        em.remove(order);
    }...
```

Câu lệnh truy vấn động

- Phương thức createQuery() của lớp EntityManager giúp tạo các câu truy vấn động (dynamic query)
- Ví dụ:

```
public List findWithName(String name) {
    return em.createQuery(
        "SELECT c FROM Customer c WHERE c.name LIKE :custName")
        .setParameter("custName", name)
        .setMaxResults(10)
        .getResultList();
}
```

Named Query

- Phương thức createNamedQuery() của lớp EntityManager giúp tạo các câu truy vấn tĩnh (static query)
- Ví dụ, khai báo named query:

```
@NamedQuery(
        name="findAllCustomersWithName",
        query="SELECT c FROM Customer c WHERE c.name LIKE :custName"
)
```

- Sử dụng named query

```
@PersistenceContext
public EntityManager em;
        ...
        customers = em.createNamedQuery("findAllCustomersWithName")
        .setParameter("custName", "Smith")
        .getResultList();
```

Named Parameter

- Tên của các tham số bắt đầu bằng dấu (:)
- Ví dụ:

```
public List findWithName(String name) {
   return em.createQuery(
         "SELECT c FROM Customer c WHERE c.name LIKE :custName")
         .setParameter("custName", name)
         .getResultList();
}
```

- Sử dụng phương thức setParameter() để truyền giá trị

Positional Parameter

- Vị trí của các tham số bắt đầu bằng dấu (?)
- Ví dụ:

```
public List findWithName(String name) {
   return em.createQuery(
         “SELECT c FROM Customer c WHERE c.name LIKE ?1”)
         .setParameter(1, name)
         .getResultList();
}
```

Demo

JPA – Java Persistence API

Quản lý transaction

```
@PersistenceContext
   EntityManagerFactory emf;
   EntityManager em;
   @Resource
   UserTransaction utx;
.....
            em = emf.createEntityManager();
            try {
                        utx.begin();
                        em.persist(SomeEntity);
                        em.merge(AnotherEntity);
                        em.remove(ThirdEntity);
                        utx.commit();
            } catch (Exception e) {
                        utx.rollback();
            }
```

Thảo luận

Spring Data JPA

Spring Data JPA

Cải tiến JPA tiêu chuẩn

Đơn giản hoá tầng truy xuất dữ liệu

Tự tạo repository

Truy vấn DSL

Ghi log, phân trang

Có thể tuỳ biến nếu cần thiết

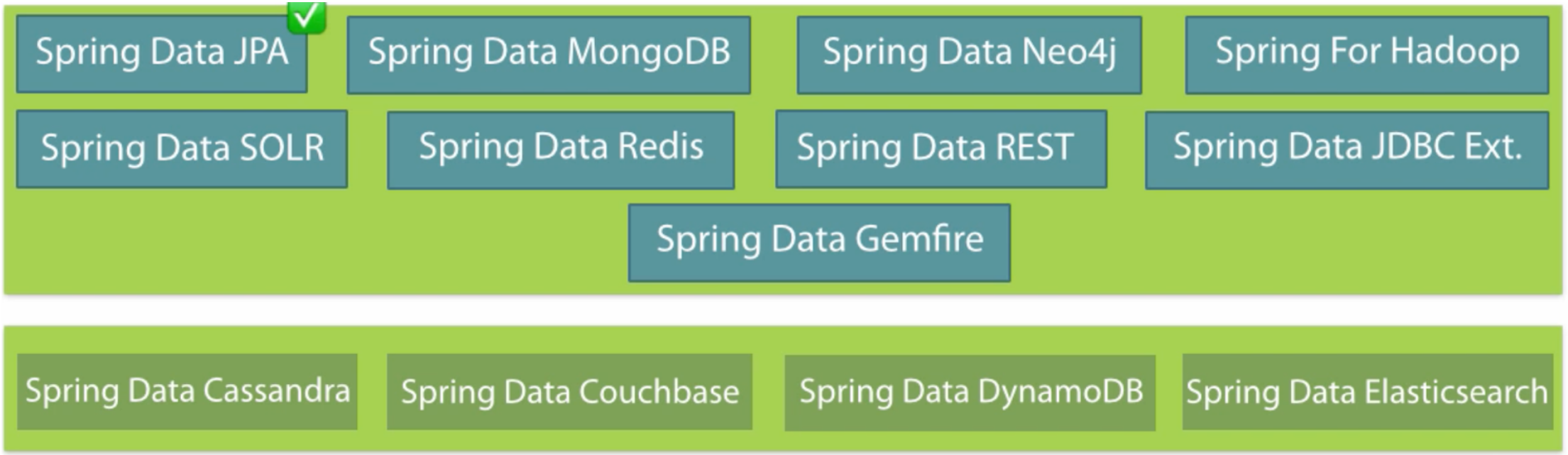
Lựa chọn Tầng truy xuất dữ liệu

JDBC Spring JDBC	JEE 7 Batch Spring Batch Hadoop	ORM JPA/Hibernate Spring Data JPA	NoSQL MongoDB Spring Data Mongo
• Đơn giản • Thuần SQL	• Rất nhiều câu lệnh ghi SQL được thực hiện	• Dễ truy xuất các mối quan hệ	• Nhóm các dữ liệu có quan hệ với nhau

Spring Data

Spring Data Commons

- Repository
- Cross-Store persistency
- Dynamic query generation



Cấu hình Spring Data JPA

build.gradle

```
dependencies {
   compile group: 'org.springframework.data', name: 'spring-data-jpa', version: '2.0.7.RELEASE'
   compile group: 'jstl', name: 'jstl', version: '1.2'
   compile group: 'mysql', name: 'mysql-connector-java', version: '8.0.11'
   testCompile group: 'org.junit.jupiter', name: 'junit-jupiter-engine', version: '5.2.0'
}
```

Cấu hình DataSource

- Có thể cấu hình thông qua XML hoặc Annotation:

```
@Bean
public DataSource dataSource(){
    DriverManagerDataSource dataSource = new DriverManagerDataSource();
    dataSource.setDriverClassName("com.mysql.cj.jdbc.Driver");
    dataSource.setUrl("jdbc:mysql://localhost:3306/phone_store");
    dataSource.setUsername( "root" );
    dataSource.setPassword( "123456" );
    return dataSource;
}
```

Cấu hình Entity Manager

```
@Bean
@Qualifier(value = "entityManager")
public EntityManager entityManager(EntityManagerFactory entityManagerFactory) {
   return entityManagerFactory.createEntityManager();
}

@Bean
public LocalContainerEntityManagerFactoryBean entityManagerFactory() {
   LocalContainerEntityManagerFactoryBean em
         = new LocalContainerEntityManagerFactoryBean();
   em.setDataSource(dataSource());
   em.setPackagesToScan(new String[] { "com.code.phonestore.model" });

   JpaVendorAdapter vendorAdapter = new HibernateJpaVendorAdapter();
   em.setJpaVendorAdapter(vendorAdapter);
   em.setJpaProperties(additionalProperties());
   return em;
}
```

Cấu hình Model

```
@Entity
@Table(name = "phones")
public class Phone {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    @Column(nullable = false)
    private String name;

    @ManyToOne(targetEntity = Manufacture.class, cascade = {CascadeType.PERSIST, CascadeType.REMOVE})
    private Manufacture manufacture;

    //Constructors
    //Getters/Setters
}
```

Truy xuất dữ liệu qua Entity Manager

```
@Transactional
public class CountryRepositoryImpl implements CountryRepository {

    @PersistenceContext
    EntityManager em;

    @Override
    public List<Country> findAll() {
        TypedQuery<Country> query = em.createQuery("select c from Country c", Country.class);
        return query.getResultList();
    }

    @Override
    public Country findById(Long id) {
        return em.find(Country.class, id);
    }
   .......
}
```

Các thao tác cơ bản

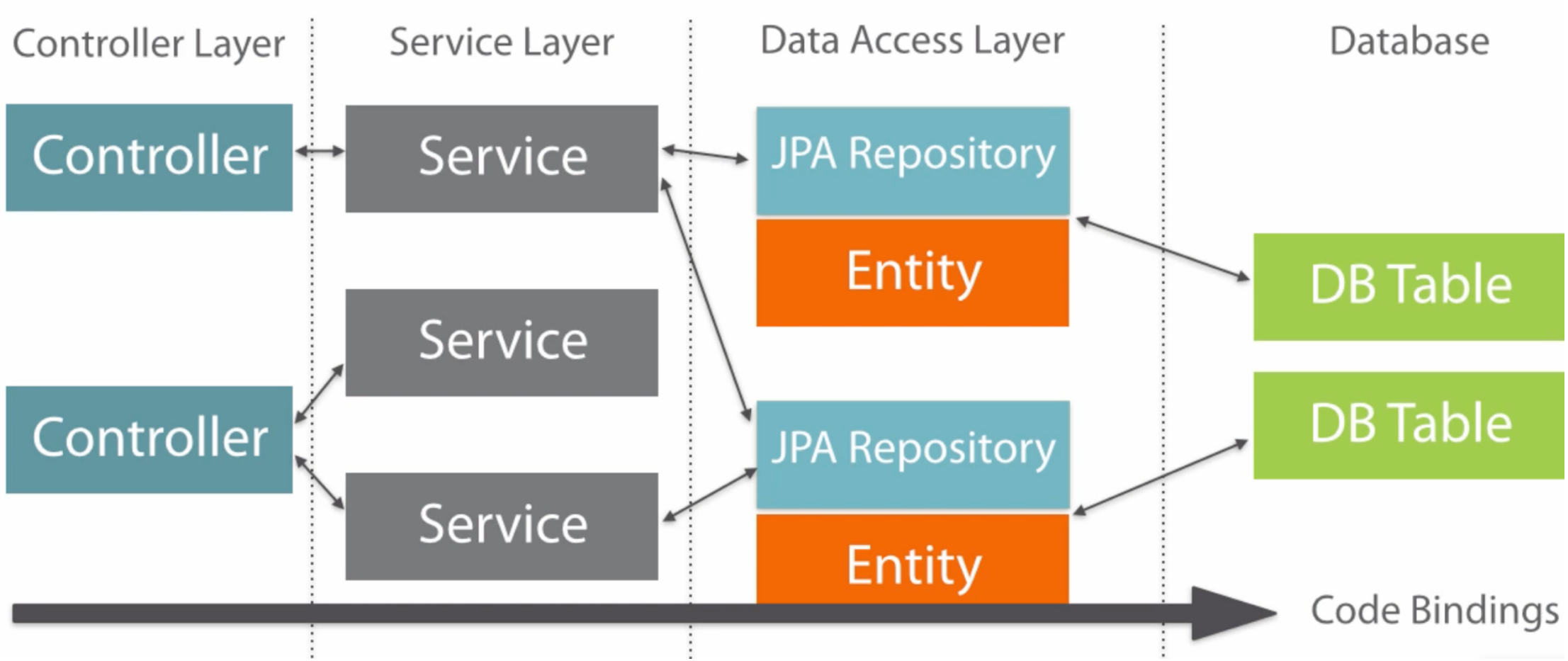
```
public interface Repository <T> {
    List<T> findAll();

    T findById(Long id);

    T save(T model);

    void remove(Long id);
}
```

Kiến trúc Repository



Tổng kết

- ORM là cơ chế ánh xạ dữ liệu giữa môi trường thực thi và CSDL
- JPA là đặc tả của Java dành cho các thao tác với dữ liệu
- Entity là các đối tượng đại diện cho dữ liệu
- Entity Manager là đối tượng quản lý các entity
- Spring Data JPA là framework của Spring hỗ trợ JPA, giúp cho việc triển khai JPA trở nên thuận tiện hơn

Bài
Spring Data JPA Repository

Mục tiêu

- Trình bày được ý nghĩa Spring Data JPA Respository
- Định nghĩa được interface Repository
- Trình bày được CrudRepository
- Trình bày được PagingAndSortingRepository
- Định nghĩa được các phương thức truy vấn
- Khởi tạo được các thực thể Repository
- Tuỳ biến được các Spring Data Repository
- Triển khai được formatter tuỳ biến
- Phục vụ được các tài nguyên tĩnh

Spring Data Repository

- Spring Data Repository giúp giảm thiểu lượng code thông thường lặp đi lặp lại ở tầng truy xuất dữ liệu.
- Spring Data Repository định nghĩa một interface chính tên là Repository. Interface này nắm bắt entity cần quản lý và kiểu dữ liệu id của entity đó.
- Interface CrudRepository kế thừa từ interface Repository, chứa các phương thức thông dụng dành cho các thao tác CRUD.
- Interface PagingAndSortingRepository cung cấp các phương thức hỗ trợ cho việc phân trang và sắp xếp các entity.

Interface Repository

- Interface chính của Spring Data JPA là *Repository*

  public interface **Repository<T, ID>**

  - Trong đó:
    - T là kiểu dữ liệu của entity muốn quản lý
    - ID là kiểu dữ liệu của id của entity muốn quản lý
- *Repository* là một ‘marker interface’: chỉ sử dụng để đánh dấu/phân loại mà không khai báo các phương thức

Các interface kế thừa Repository

- CrudRepository<T,ID>
- PagingAndSortingRepository<T,ID>
- ReactiveCrudRepository<T,ID>
- ReactiveSortingRepository<T,ID>
- RevisionRepository<T,ID,N>
- RxJava2CrudRepository<T,ID>
- RxJava2SortingRepository<T,ID>

Interface CrudRepository

- Hỗ trợ thực hiện các câu lệnh CRUD cơ bản

```
public interface CrudRepository<T, ID extends Serializable> extends Repository<T, ID> {

  <S extends T> S save(S entity);

  Optional<T> findById(ID primaryKey);

  Iterable<T> findAll();

  long count();

  void delete(T entity);

  boolean existsById(ID primaryKey);

  // … more functionality omitted.
}
```

Lưu một entity

Tìm một entity theo Id

Lấy về tất cả các entity

Lấy về số lượng entity

Xoá một entity

Kiểm tra sự tồn tại của một entity theo Id

Interface PagingAndSortingRepository

- Kế thừa từ CrudRepository
- Bổ sung khả năng phân trang và sắp xếp

```
public interface PagingAndSortingRepository<T, ID extends Serializable> extends
CrudRepository<T, ID> {

    Iterable<T> findAll(Sort sort);

    Page<T> findAll(Pageable pageable);
}
```

Các interface tuỳ biến

- Có thể định nghĩa các interface để thực hiện các thao tác đặc thù
- Ví dụ:

```
interface UserRepository extends CrudRepository<User, Long> {

    long countByLastname(String lastname);
}
```

- Hoặc:

```
interface UserRepository extends CrudRepository<User, Long> {

    long deleteByLastname(String lastname);

    List<User> removeByLastname(String lastname);
}
```

Thảo luận

Phương thức truy vấn (Query method)

Phương thức truy vấn (Query method)

- Query method là những phương thức được khai báo trong repository interface có nhiệm vụ lấy thông tin từ cơ sở dữ liệu.
- Query method giúp cho việc lấy thông tin từ cơ sở dữ liệu mà không cần viết một câu query nào.
- Ví dụ: Lấy thông tin của đối tượng Customer

```
public interface CustomerRepository extends PagingAndSortingRepository<Customer, Long> {
}

public class CustomerServiecImpl implements CustomerService{
        @Autowired
        private CustomerRepository customerRepository;
        @Override
        public Customer findById(Long id) {
                return customerRepository.findOne(id);
        }
}
```

Các bước khai báo các phương thức truy vấn

1. Khai báo một interface kế thừa Repository hoặc các interface con của nó
2. Khai báo các phương thức truy vấn tuỳ biến
3. Cấu hình Spring để tạo các proxy instance cho các repository tuỳ biến
   - Có thể sử dụng XML hoặc Annotation
4. Sử dụng repository tuỳ biến thông qua cơ chế Injection

Khai báo các interface repository

- Kế thừa interface Repository hoặc các interface con
- Xác định kiểu của entity và kiểu của Id
- Ví dụ:

```
interface PersonRepository extends Repository<Person, Long> {
  …
}
```

- Có thể không kế thừa các interface repository có sẵn bằng cách sử dụng annotation *@RepositoryDefinition*

Khai báo các phương thức truy vấn

- Cơ chế hoạt động của repository store cho phép 2 hình thức để tạo ra các câu lệnh truy vấn:
  - Dựa vào tên của phương thức. Ví dụ:

    List<Person> findByEmailAddressAndLastname(EmailAddress emailAddress, String lastname);

  - Dựa vào câu lệnh truy vấn được khai báo cụ thể. Ví dụ:

```
@Query("select u from User u where u.emailAddress = ?1")
User findByEmailAddress(String emailAddress);
```

Tạo các đối tượng repository

- Có thể sử dụng XML hoặc Java Configuration để cấu hình tạo các đối tượng repository
- Spring Data sẽ dò tìm và tạo ra các proxy tương ứng với từng repository để thao tác với dữ liệu
- Tên của bean được sinh ra dựa trên tên của repository. Chẳng hạn UserRepository sẽ có một bean là userRepository

Tạo các đối tượng repository: Java Configration

- Ví dụ:

```
@Configuration
@EnableJpaRepositories("com.acme.repositories")
class ApplicationConfiguration {

   @Bean
   EntityManagerFactory entityManagerFactory() {
      // …
   }
}
```

Tạo câu truy vấn

- Spring Data repository giúp cho việc tạo các câu truy vấn tự động
- Cơ chế này duyệt qua tên của phương thức với các tiền tố như *find…By, read…By, query…By, count…By* và *get…By* để xây dựng câu truy vấn
- Có thể sử dụng thêm một số từ khoá khác, chẳng hạn như Distinct, Asc, Desc, Or, And…

Ví dụ khai báo các phương thức truy vấn

```
interface PersonRepository extends Repository<User, Long> {

  List<Person> findByEmailAddressAndLastname(EmailAddress emailAddress, String lastname);

  // Enables the distinct flag for the query
  List<Person> findDistinctPeopleByLastnameOrFirstname(String lastname, String firstname);
  List<Person> findPeopleDistinctByLastnameOrFirstname(String lastname, String firstname);

  // Enabling ignoring case for an individual property
  List<Person> findByLastnameIgnoreCase(String lastname);
  // Enabling ignoring case for all suitable properties
  List<Person> findByLastnameAndFirstnameAllIgnoreCase(String lastname, String firstname);

  // Enabling static ORDER BY for a query
  List<Person> findByLastnameOrderByFirstnameAsc(String lastname);
  List<Person> findByLastnameOrderByFirstnameDesc(String lastname);
}
```

Xử lý các tham số đặc biệt

- Có thể truyền thêm các tham số đặc biệt vào trong câu truy vấn, chẳng hạn như Pageable, Slice và Sort
- Spring Data repository sẽ nhận diện và chuyển thành câu truy vấn phù hợp
- Ví dụ:

```
Page<User> findByLastname(String lastname, Pageable pageable);

Slice<User> findByLastname(String lastname, Pageable pageable);

List<User> findByLastname(String lastname, Sort sort);

List<User> findByLastname(String lastname, Pageable pageable);
```

Pageable, Slice

- Pageable là đối tượng hỗ trợ phân trang
- Các phương thức thông dụng của Pageable: *first(), getOffset(), getPageNumber(), getPageSize(), getSort(), hasPrevisous(), isPaged(), next(), previousOrFirst()*
- Slice là đối tượng hỗ trợ phân trang, nhưng không biết được tổng số lượng trang (Pageable biết tổng số lượng trang)
- Slice có thể tốt hơn Pageable về mặt hiệu năng khi làm việc với nhiều dữ liệu (do không phải đếm tổng số lượng trang)
- Các phương thức thông dụng của Slice: *getContent(), getNumber(), getPageable(), getSize(), getSort(), hasNext(), hasPrevious(), isFirst(), isLast()*

Sort

- Sort là đối tượng hỗ trợ sắp xếp khi truy vấn
- Các constructor thông dụng:
  - **Sort**(**Sort.Direction** direction, **List**<**String**> properties)
- Các phương thức static thông dụng:
  - **by**(**List**<**Sort.Order**> orders)
  - **by**(**Sort.Direction** direction, **String**... properties)
  - **by**(**Sort.Order**... orders)
- Các phương thức thông dụng của Sort:
  - and(Sort)
  - Ascending()
  - Descending()
  - Interator()

Hạn chế số lượng kết quả

- Sử dụng từ *top* hoặc *first* để hạn chế số lượng kết quả
- Có thể thêm một giá trị số để quy định số lượng kết quả
- Nếu không có giá trị số thì 1 kết quả sẽ được trả về
- Ví dụ:

```
User findFirstByOrderByLastnameAsc();

User findTopByOrderByAgeDesc();

Page<User> queryFirst10ByLastname(String lastname, Pageable pageable);

Slice<User> findTop3ByLastname(String lastname, Pageable pageable);

List<User> findFirst10ByLastname(String lastname, Sort sort);

List<User> findTop10ByLastname(String lastname, Pageable pageable);
```

Sử dụng Spring Data

- Spring Data có thể được sử dụng trong container hoặc độc lập
- Trong các container, có thể sử dụng cơ chế CDI (Context and Dependency Injection) để tạo các bean repository
- Trong các ứng dụng độc lập, cần thêm thư viện Spring Data vào trong classpath và tự khởi tạo các đối tượng repository
- Ví dụ:

```
RepositoryFactorySupport factory = … // Instantiate factory here
    UserRepository repository = factory.getRepository(UserRepository.class);
```

Hỗ trợ web

- Spring Data hỗ trợ tích hợp với các tầng web
- Có thể sử dụng annotation @EnableSpringDataWebSupport trong Java Configuration để cho phép tích hợp
- Ví dụ:

```
@Configuration
@EnableWebMvc
@EnableSpringDataWebSupport
class WebConfiguration {


}
```

- Annotation @EnableSpringDataWebSupport sẽ thêm các thành phần DomainClassConverter và HandlerMethodArgumentResolvers vào trong ứng dụng web

DomainClassConverter

- DomainClassConverter là thành phần cho phép tự động xác định các tham số trong request của web và chuyển thành các entity tương ứng
- Ví dụ:

```
@Controller
@RequestMapping("/users")
class UserController {

   @RequestMapping("/{id}")
   String showUserForm(@PathVariable("id") User user, Model model) {
      model.addAttribute("user", user);
      return "userForm";
   }
}
```

HandlerMethodArgumentResolvers

- HandlerMethodArgumentResolvers là thành phần cho phép tự động nhận diện và chuyển đổi các đối tượng Pageable và Sort tương ứng với request của controller
- Ví dụ:

```
@Controller
@RequestMapping("/users")
class UserController {

   @Autowired
   private UserRepository repository;

   @RequestMapping
   String showUsers(Model model, Pageable pageable) {
      model.addAttribute("users", repository.findAll(pageable));
      return "users";
   }
}
```

Ánh xạ các tham số với Pageable

Tham số	**Giải thích**
page	Số trang. Mặc định là 0
size	Kích thước trang. Mặc định là 20.
sort	Trật tự sắp xếp (ASC | DESC). Mặc định là ascending. Ví dụ: *?sort=firstname&sort=lastname,asc*

Cấu hình sử dụng JPA Repository

```
@Configuration
@EnableJpaRepositories
@EnableTransactionManagement
class ApplicationConfig {

    @Bean
    public DataSource dataSource() {

        EmbeddedDatabaseBuilder builder = new EmbeddedDatabaseBuilder();
        return builder.setType(EmbeddedDatabaseType.HSQL).build();
    }

    .......
}
```

```
@Bean
  public LocalContainerEntityManagerFactoryBean entityManagerFactory() {

     HibernateJpaVendorAdapter vendorAdapter = new HibernateJpaVendorAdapter();
     vendorAdapter.setGenerateDdl(true);

     LocalContainerEntityManagerFactoryBean factory = new LocalContainerEntityManagerFactoryBean();
     factory.setJpaVendorAdapter(vendorAdapter);
     factory.setPackagesToScan("com.acme.domain");
     factory.setDataSource(dataSource());
     return factory;
  }

  @Bean
  public PlatformTransactionManager transactionManager(EntityManagerFactory entityManagerFactory) {

     JpaTransactionManager txManager = new JpaTransactionManager();
     txManager.setEntityManagerFactory(entityManagerFactory);
     return txManager;
  }
```

Converter và Formatter

- Converter và Formatter hỗ trợ chuyển đổi dữ liệu nhập vào sang kiểu dữ liệu thích hợp
- Ví dụ, Spring sẽ tự động thử chuyển đổi dữ liệu từ một trường *<input type="date">* sang một đối tượng java.util.Date
- Converter là các thành phần sử dụng chung cho toàn bộ hệ thống, có thể sử dụng converter ở bất cứ tầng nào của ứng dụng.
- Formatter thì chỉ được thiết kế để sử dụng ở tầng web (web tier)

Định nghĩa Converter

- Triển khai interface *Converter*

  public interface Converter<S, T>

  - Trong đó:
    - S là kiểu dữ liệu nguồn
    - T là kiểu dữ liệu đích

- Ví dụ, định nghĩa converter để chuyển từ kiểu String sang LocalDate:

```
public class StringToLocalDateConverter implements Converter<String, LocalDate> {
    @Override
    public LocalDate convert(String source) {
        //Conversion
    }
}
```

Đăng ký converter

```
@Configuration
@ComponentScan("com.code.converter")
@EnableWebMvc
public class ApplicationConfig extends WebMvcConfigurerAdapter {
    @Override
    public void addFormatters(FormatterRegistry registry) {
        StringToLocalDateConverter stringToLocalDateConverter = new
StringToLocalDateConverter("MM-dd-yyyy");
        registry.addConverter(stringToLocalDateConverter);
    }
}
```

Định nghĩa Formatter

- Triển khai interface Formatter

  public interface Formatter<T>

  - Trong đó T là kiểu dữ liệu đích
- Ví dụ, chuyển đổi sang kiểu dữ liệu LocalDate:

```
public class LocalDateFormatter implements Formatter<LocalDate> {
   @Override
   public LocalDate parse(String text, Locale locale) throws ParseException {
     //String to LocalDate
   }
   @Override
   public String print(LocalDate date, Locale locale) {
     //LocalDate to String
   }
}
```

Đăng ký formatter

```
@Configuration
@EnableWebMvc
@ComponentScan("com.code.formatter")
public class ApplicationConfig extends WebMvcConfigurerAdapter {

    @Override
    public void addFormatters(FormatterRegistry registry) {
        LocalDateFormatter localDateFormatter = new LocalDateFormatter("MM-dd-yyyy");
        registry.addFormatter(localDateFormatter);
    }
}
```

Tổng kết

- Spring Data cung cấp các interface repository để tự động hoá các thao tác với CSDL
- Interface CrudRepository hỗ trợ các thao tác CRUD cơ bản
- Interface PagingAndSortingRepository hỗ trợ phân trang và sắp xếp
- Có thể khai báo các interface tuỳ biến kế thừa từ interface Repository hoặc các interface con của nó
- Có thể định nghĩa các phương thức truy vấn tuỳ biến
- Câu lệnh truy vấn có thể được sinh ra trực tiếp dựa trên tên của phương thức truy vấn