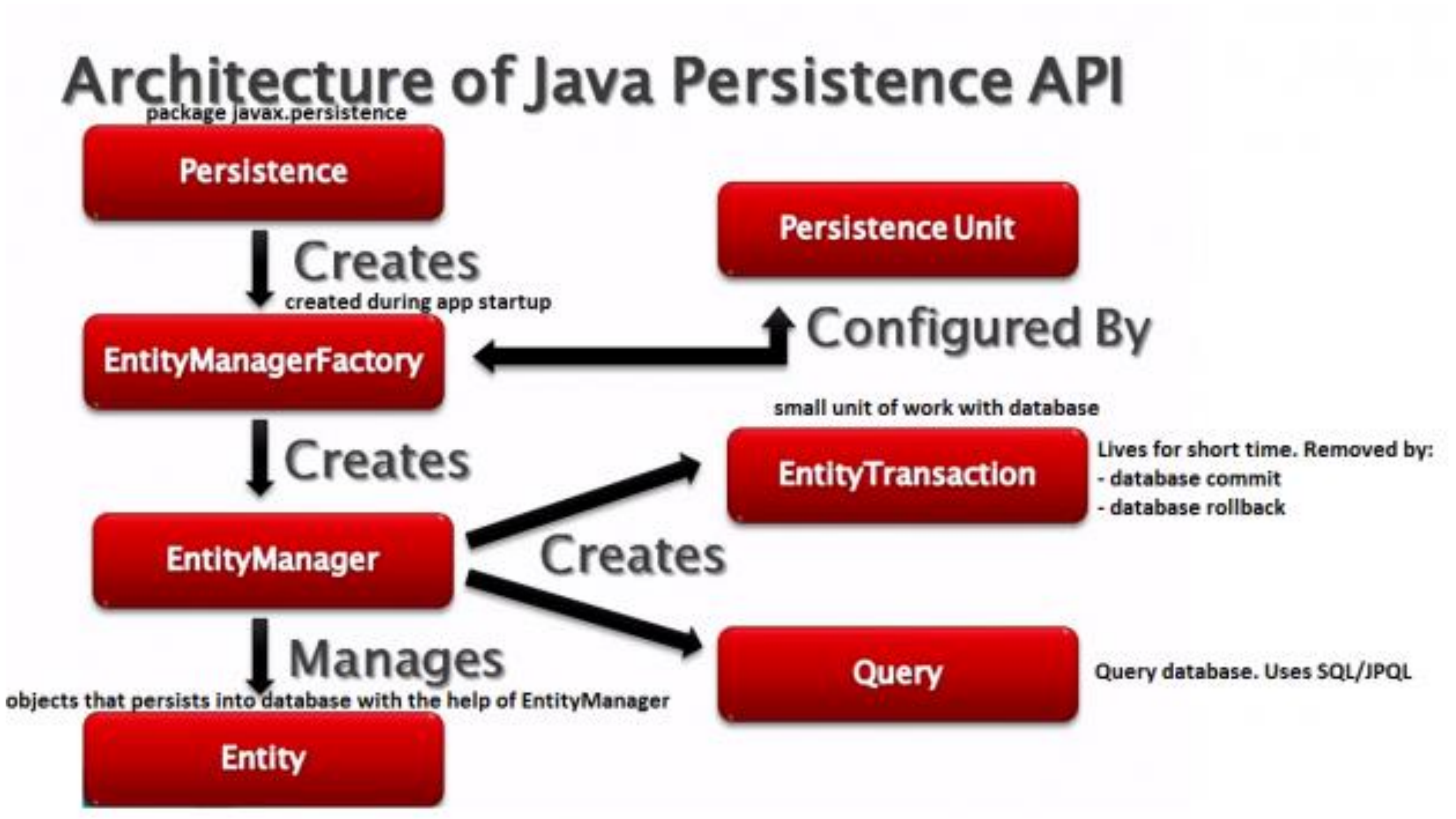
ExcelShore®
Operational Excellence

Cybage®
Delivering Value Scientifically

Spring Data JPA

JPA Architecture


Architecture of Java Persistence API
package javax.persistence
Persistence
Creates
created during app startup
EntityManagerFactory
Creates
EntityManager
Manages
objects that persists into database with the help of EntityManager
Entity
Persistence Unit
Configured By
small unit of work with database
EntityTransaction
Lives for short time. Removed by:
- database commit
- database rollback
Creates
Query
Query database. Uses SQL/JPQL

Example

Step 1: Model or POJO

Entity POJO(Employee.java)

Step 2: Persistence

Persistence.xml

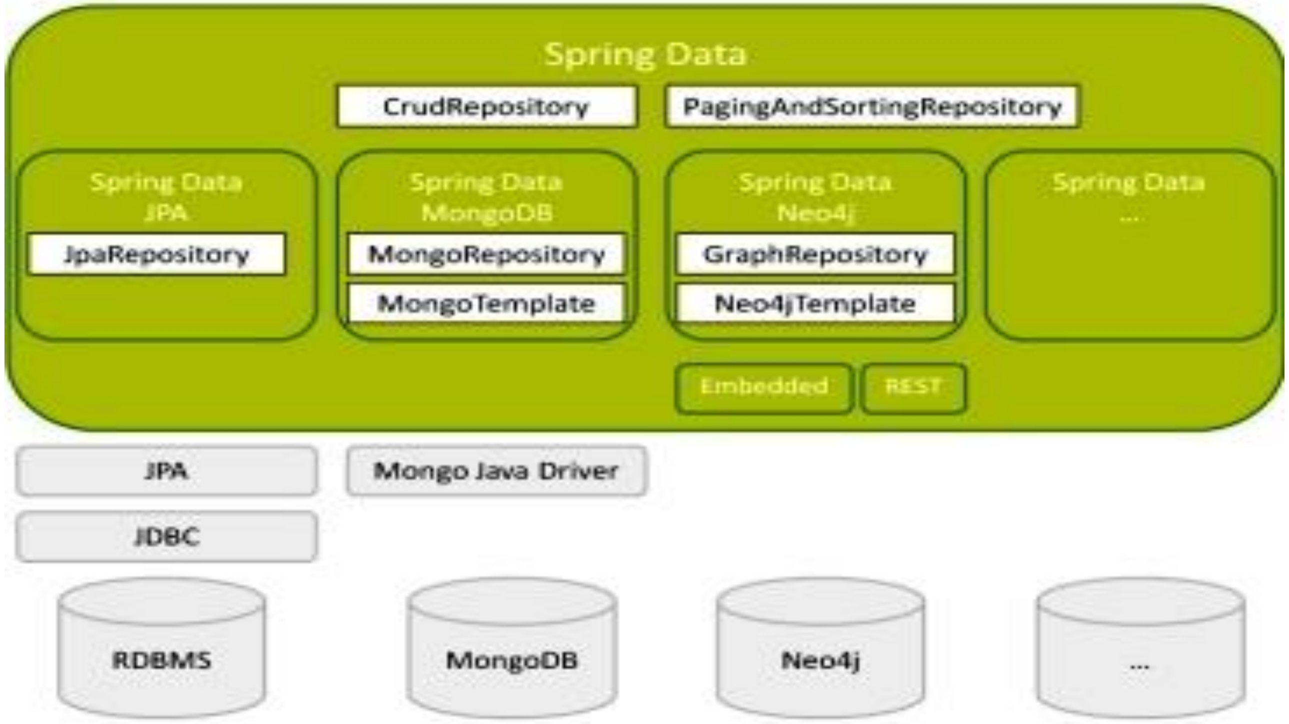
Step 3: Services: (CRUD operations)

1. CreatingEmployee
2. UpdatingEmployee
3. FindingEmployee
4. DeletingEmployee

JPA with Spring MVC

What is Spring Data

- Spring Data is a high level SpringSource project whose purpose is to unify and ease the access to different kinds of persistence stores, both relational database systems and NoSQL data stores.



What is Spring-Data-JPA

- You use @Repository interfaces instead of DAO Implementations,
- Less boilerplate coding of DAO classes for frequent operations like persisting, finding, listing, etc…
- You can define JPA queries by annotations like this:
  @Query("select p from Product p where p.name = :name")
- Or you can define a query by only keywords inside method names:
  findByNameContainingIgnoreCase(String searchString);

Features of Spring Data-JPA

- Sophisticated support to build repositories based on Spring and JPA
- Support for Querydsl predicates and thus type-safe JPA queries
- Transparent auditing of domain class
- Pagination support, dynamic query execution, ability to integrate custom data access code.
- Validation of @Query annotated queries at bootstrap time.
- Support for XML based entity mapping.
- JavaConfig based repository configuration by introducing @EnableJpaRepositories.

No More DAO Implementations

Advantages of such a simplification are :

- a decrease in the number of artifacts that need to be defined and maintained
- consistency of data access patterns ,
- consistency of configuration.

- DAO interface needs to extend the JPA specific Repository interface – JpaRepository , which enables Spring Data to find this interface and automatically create an implementation for it.

- By extending the interface we get the most relevant CRUD methods for standard data access available in a standard DAO out of the box.

CrudRepository interface

```
public interface CrudRepository<T, ID extends Serializable>
    extends Repository<T, ID> {
    <S extends T> S save(S entity);  1
    T findOne(ID primaryKey);        2
    Iterable<T> findAll();           3
    Long count();                    4
    void delete(T entity);           5
    boolean exists(ID primaryKey);   6
    // … more functionality omitted.
}
```

1. Saves the given entity.
2. Returns the entity identified by the given id.
3. Returns all entities.
4. Returns the number of entities.
5. Deletes the given entity.
6. Indicates whether an entity with the given id exists.

Cybage® Delivering Value Scientifically

JPA Repository

Spring Data repository support

- Spring Data JPA provides a repository programming model that starts with an interface per managed domain object:

```
public interface AccountRepository extends JpaRepository<Account, Long>
{ … }
```

Custom Access Method and Queries

To define more specific access methods, Spring JPA supports :

- simply define a new method in the interface
- provide the actual JPQ query by using the *@Query* annotation
- use the more advanced Specification and Querydsl support in Spring Data
- define custom queries via JPA Named Queries

Automatic Custom Queries

- When Spring Data creates a new *Repository* implementation, it analyses all the methods defined by the interfaces and tries to **automatically generate queries from the method names**.
- While this has some limitations, it is a very powerful and elegant way of defining new custom access methods with very little effort.

```
public interface IFooDAO extends JpaRepository< Foo, Long >{

   Foo findByName( String name );

}
```

@Query Annotation

- The @*Query* annotation can be used to create queries by using the JPA query language and to bind these queries directly to the methods of your repository interface.

- When the query method is called, Spring Data JPA will execute the query specified by the @*Query* annotation .

Manual Custom Queries

We will define via the *@Query* annotation

```
@Query("SELECT f FROM Foo f WHERE LOWER(f.name) = LOWER(:name)")
    Foo retrieveByName(@Param("name") String name);
```

Spring Data Configuration

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans
  xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:jpa="http://www.springframework.org/schema/data/jpa"
  xsi:schemaLocation="
     http://www.springframework.org/schema/beans
     http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
     http://www.springframework.org/schema/data/jpa
     http://www.springframework.org/schema/data/jpa/spring-jpa.xsd">

   <jpa:repositories base-package="org.rest.dao.spring" />
 </beans>
```

The Spring Java or XML Configuration

- Annotation based repository configuation:

```
@Configuration
@EnableTransactionManagement
@ImportResource( "classpath*:*springDataConfig.xml" )
public class PersistenceJPAConfig{
   ...
}
```

Finally all steps together to build Spring Data

- Declare an interface extending Repository or one of its subinterfaces and type it to the domain class and ID type that it will handle.

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

- Declare query methods on the interface.

  interface PersonRepository extends Repository<Person, Long> {
  List<Person> findByLastname(String lastname); }

- Set up Spring to create proxy instances for those interfaces.
- Either via JavaConfig or XML Configuration

```
import
org.springframework.data.jpa.repository.config.EnableJpaRepositories;
@EnableJpaRepositories
class Config {}
```

Pagination and Sorting

- When we perform bulk operations, like finding all Person's from the database or finding everyone based on a country, we often do the paging so that we can present a small data chunk to the end user and, in the next request, we fetch the next data chunk.

Advantages

- This method enhances readability for end users. The users will need to scroll to find a particular person, which is bad UI design.

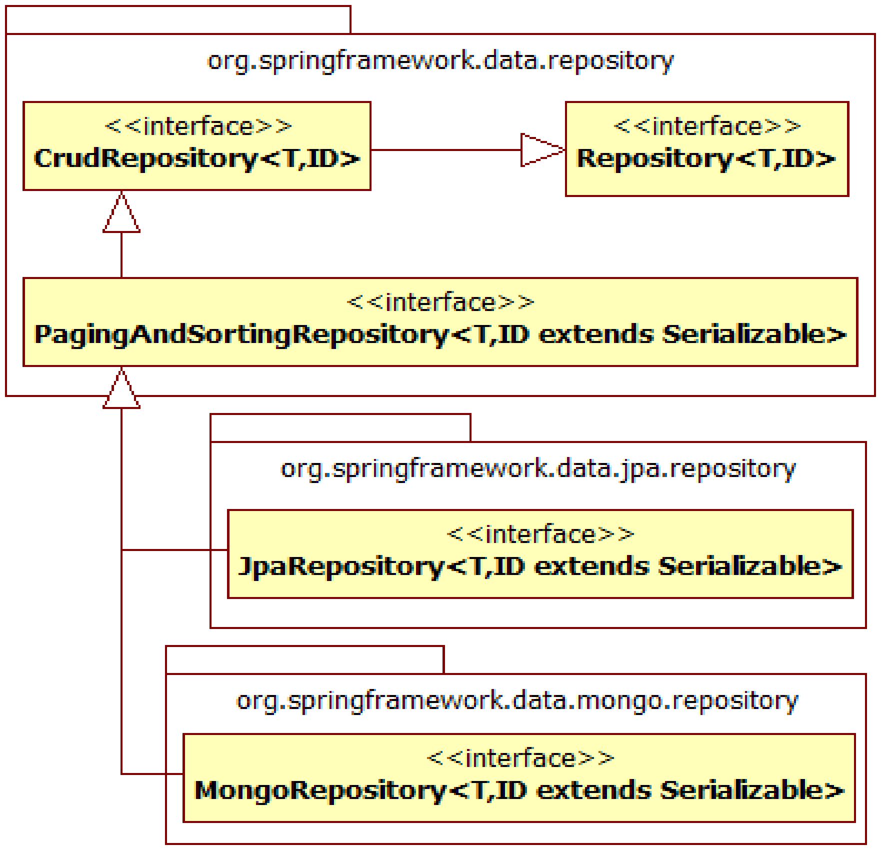
- It also reduces the query time and enhances performance as, in spite of fetching all the data, we only fetch a small chunk of data — amounting to less query time. A lot of UI technology supports client-side paging, which fetches all the data, and, based on the request, shows paginated data. But that doesn't reduce the query time, it only provides the first advantage.

Disadvantages

- To request every data chunk, a server trip is required. You can optimize it through caching, but when an end user is in the process of fetching data, another Person can be added to the system. If that situation, it might show as the last entry of one page and the first entry of another.

- But we are always using paging to fetch a small chunk of data, rather than all of it.

Interface Hierarchy


org.springframework.data.repository
<<interface>>
CrudRepository<T,ID>
<<interface>>
Repository<T,ID>
<<interface>>
PagingAndSortingRepository<T,ID extends Serializable>
org.springframework.data.jpa.repository
<<interface>>
JpaRepository<T,ID extends Serializable>
org.springframework.data.mongo.repository
<<interface>>
MongoRepository<T,ID extends Serializable>

Steps for Pagination

1. In your custom repository, extend **PagingAndSortingRepository**.

2. Create a **PageRequest** object, which is an implementation of the **Pageable** interface
   This PageRequest object takes the page number, the page size, and sorts direction and sort field.

3. By passing requested page number and page limit you can get the data for this page.
   If you pass a wrong page number Spring data will take care of that and not return any data.

References

For Relationships scenarios you can refer the below link:

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

2. http://www.javainterviewpoint.com/spring-data-jpa-one-to-many/

ExcelShore® Operational Excellence

Cybage® Delivering Value Scientifically

Thank You