Spring Data

RUJUAN XING

What is Spring Data?

- Spring Data offers a flexible abstraction for working with data access frameworks.
- Purpose is to **unify and ease the access** to different kinds of persistence stores.
- Both relational DB systems and NoSQL data stores.
- Addresses common difficulties developers face when working with databases in applications.
- Spring Data is an umbrella project that provides you with easy to use data access technologies for all kinds of relational and non-relational DBs.

Spring Data Modules

Modules supports:

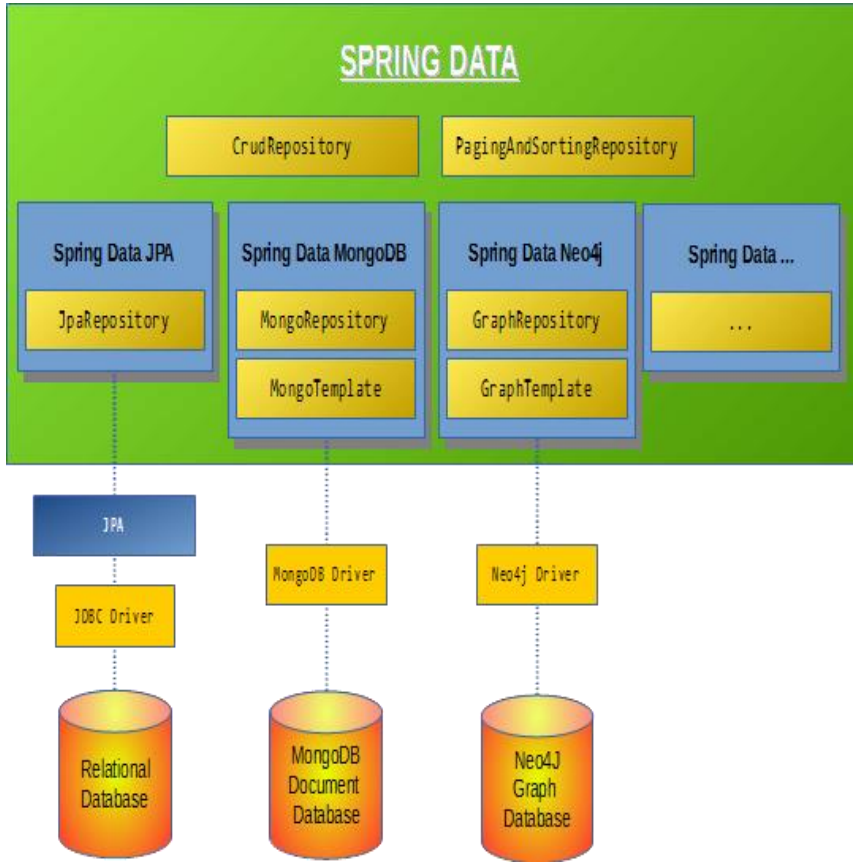
- Object/Datastore mapping
- Repository support

Every repository offers:

- CRUD operations
- Finder Methods
- Sorting and Pagination

Spring data module repositories provide generic interfaces:

- CrudRepository
- PagingAndSortingRepository



Mapping

JPA uses Object Relational Mapping

- Spring Data extends this concept to NoSQL datastores

All Spring Data modules provide an Object to data store mapping

- Because these data store structures can be so different, there is no common API
- Each type of data store has it's own set of annotations to map between objects and data store structures.

Mapping Examples

JPA	MongoDB	Neo4J

@Entity
@Table(name="USR")
public class User {
 	
@Id
private String id;
 	
@Column(name="fn")
private String name;
 	
private Date lastLogin;
 	
...
}
	@Document(
	collection="usr")
	public class User {
	
	@Id
	private String id;
	
	@Field("fn")
	private String name;
	
	private Date lastLogin;
	
	...
	}
		@NodeEntity
		public class User {
		
		@GraphId
		Long id;
		
		private String name;
		
		private Date lastLogin;
		
		...
		}

JPA Example

SPRING DATA

DAOs aka Repositories

DAOs (also known as repositories) are classes that implement CRUD operations and finder methods for a Entity

Spring Data generates DAOs which:

- Have the **same basic interface regardless of data store**
- Provide a common way of querying for data
- Provide a CRUD interface (if reasonable for DS)

CrudRepository Interface

- Spring Data JPA is instructed to scan package
- DAOs are generated for any **interface** that extends **Repository<T, id>**
  - JpaRepository extends PagingAndSortingRepostirory extends CrudRepository extends Repository

count()	saveAll(Iterable<S> entities)
exists(ID id)	save(S entity)
findAll()	delete(ID id)
findAllById(Iterable<ID> ids)	delete(T entity)
findById(id)	deleteAll(Iterable<? extends T> entities)
	deleteAll()

JPA Repository methods

Example Code

```
public interface ContactDao extends CrudRepository<Contact, Long> {
}
```

Use an interface to declare that there should be a ContactDao

Spring Data will automatically generate an implementation

```
@Service
@Transactional
public class ContactService {

   @Autowired
   private ContactDao contactDao;

        ...
}
```

Generated DAO can be used like any other DAO before this

Finder Methods

SPRING DATA

Finder Methods

- The basic provided methods may be enough for a small application
  - Real applications often need more specific methods
- You can **add a finder method**
  - by using the **name of the property**

```
public interface ContactDao extends JpaRepository<Contact, Long> {
        List<Contact> findByName(String name);
}
```

Contact Entity has a name property

Finder Method Prefixes

- The following prefixes start a finder method:
  - findBy, readBy, queryBy, getBy, and countBy
- They can then contain further expressions like:
  - Distinct, Top10, First5

```
public interface ContactDao extends JpaRepository<Contact, Long> {
        List<Contact> findDistinctContactByLastname(String lastname);
        List<Contact> findContactDistinctByLastname(String lastname);

        Page<Contact> queryFirst10ByLastname(String lastname, Pageable pageable);
        Slice<Contact> findTop3ByLastname(String lastname, Pageable pageable);
        List<Contact> findFirst10ByLastname(String lastname, Sort sort);
        List<Contact> findTop10ByLastname(String lastname, Pageable pageable);
}
```

Criteria

- The first **By** acts as a delimiter to indicate the start of the actual criteria

```
List<Contact> findDistinctContactByLastname(String lastname);
```

- You can combine criteria with **and** and **or**

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

- Typical operators are also supported (see next)

List of Keywords

And	After	Containing
Or	Before	OrderBy
Is,Equals	IsNull	Not
Between	IsNotNull, NotNull	In
LessThan	Like	NotIn
LessThanEqual	NotLike	True
GreaterThan	StartingWith	False
GreaterThanEqual	EndingWith	IgnoreCase (AllIgnoreCase)

Examples:

```
findByFirstname,findByFirstnameIs,findByFirstnameEquals // all the same
findByStartDateBetween // expects two parameters
findByAgeLessThanEqual
findByAgeIsNotNull, findByAgeNotNull // same
findByFirstnameLike // first parameter matched as Like (including %'s)
findByAgeIn(Collection<Age> ages) // or subclass of collection
```

IgnoreCase

- You can ignore case for a single property

```
List<Person> findByFirstNameAndLastnameIgnoreCase(String firstname, String lastname);
```

Only LastName is case insensitive

- Or ignore case for all properties

```
List<Person> findByLastnameAndFirstnameAllIgnoreCase(String lastname, String firstname);
```

PagingAndSortingRepository

- The PagingAndSortingRepository interface adds methods to sort and paginate entities
  - findAll(Pageable pageable), findAll(Sort sort)
- Enables you to use the orderBy keyword

```
public List<Student> findByLastName OrderByLastNameAsc(String lastName);
public List<Student> findByLastName OrderByLastNameDesc(String lastName);
```

- You can request a page using PageRequest

```
PageRequest pageRequest = new PageRequest(0, 10);
Page<Student> page = studentDao.findAll(pageRequest);
Slice<Student> slice = studentDao.findByFirstName("Lisa", pageRequest);
```

Difference Between Page and Slice

- Page extends Slice
  - Slice only knows if there is more
  - Not how much more
- Page executes a select count to find out

Limiting Results

- The keywords **first** and **top** will limit the amount of rows that will be returned
  - Optional numeric value can be added

```
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);
```

- If pagination or slicing is applied it is within the limited result

Advanced Methods

SPRING DATA

Property Expressions

- Similar to HQL you can traverse properties

```
List<Student> findByAddressZipCode(ZipCode zipCode);// x.address.zipCode
```

- What if a student has a AddressZipCode property?

```
List<Student> findByAddress_ZipCode(ZipCode zipCode);// x.address.zipCode
```

Optionally underscores can be used to separate properties

Java Methods should use CamelCase there should not be any conflicts caused by this

@Query

- What if you don't like typing long names or need to have a complicated query?

```
public interface UserRepository extends JpaRepository<User, Long>{
        @Query("Select u from User u where u.emailAddress = ?1")
        User findByEmailAddrss(String emailAddress);
}
```

- Or want to write SQL instead of HQL?

```
public interface UserRepository extends JpaRepository<User, Long>{
        @Query(value = "SELECT * FROM USERS WHERE EMAIL_ADDRESS = ?0", nativeQuery = true)
        User findByEmailAddrss(String emailAddress);
}
```

Summary

- Spring data allows you to generate DAOs independent of the type of data store you use
- The default methods of the generated DAOs are good for simple applications
- It’s easy to add additional Finder methods using nothing more than the method names
- Custom queries or complete method implementations can easily be added