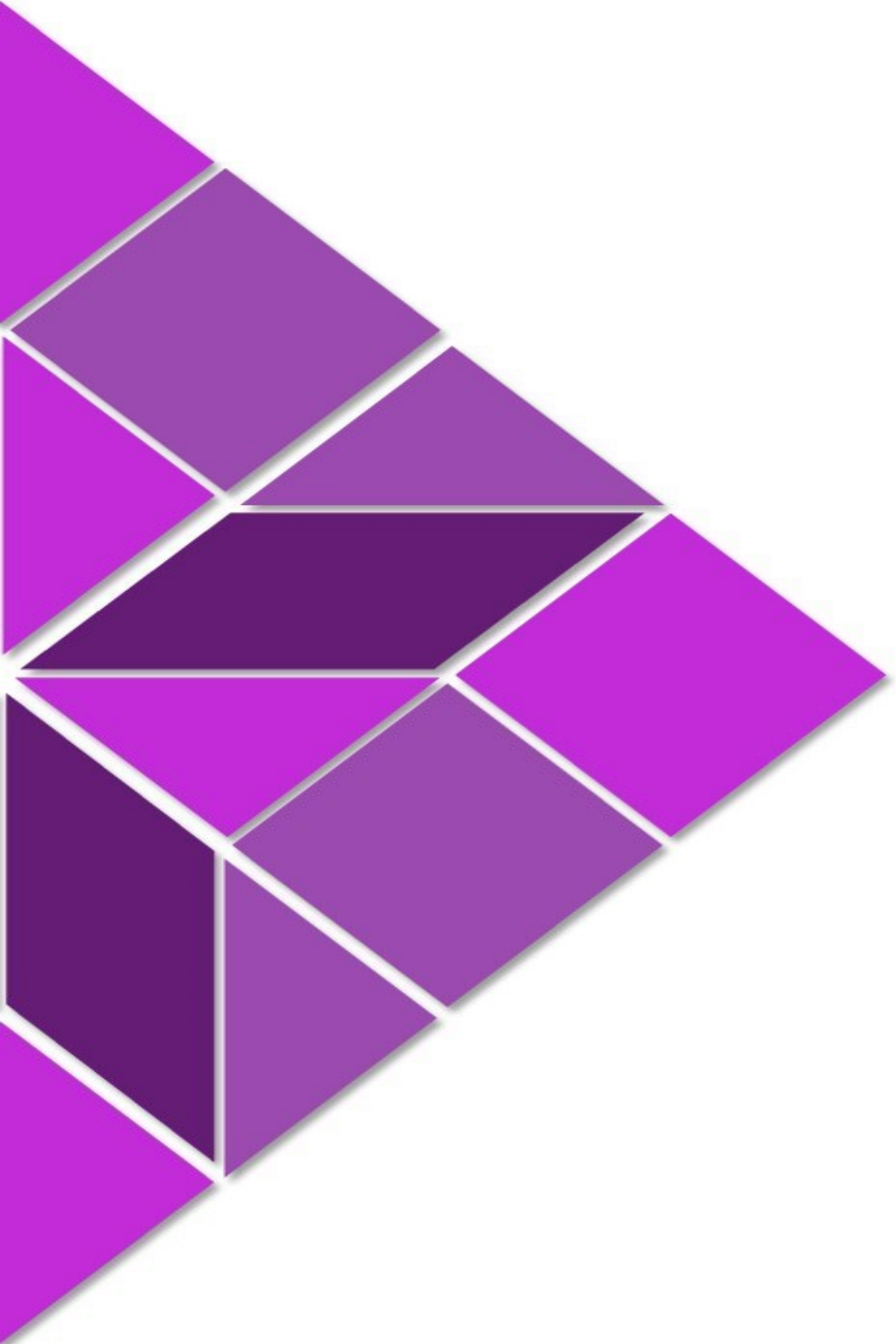

Spring Data

Presented by Hosein Zare
Twitter: @zare88r

Spring Data

- هدف spring data فراهم آوردن یک رویه واحد برای کار با دیتاها میباشد
- این ماژول با تکنولوژی های مختلفی از جمله پایگاه داده های رابطه ای ، ORM ها ، No SQL ها و ... کار میکند

  jpa , jdbc , gemfire , ldap , mongodb , rest , redis , ...

Spring data JDBC

- Spring JDBCچندین روش مختلف جهت اتصال به پایگاه داده و همچنین کلاس های لازم جهت ارتباط با آن را فراهم نموده است
- کلاس JdbcTemplate، یک ابزار بسیار ساده برای کار با api jdbc است و متودهای زیادی جهت سادگی در واکشی ،انجام dml و صدا زدن procedure store ها و ... فراهم کرده است

Spring data JDBC

- برای استفاده از این موضوع باید bean به نام datasource تعریف شود .
در Spring Boot کافی است که property های زیر را تعریف کنید :

```
spring.datasource.username=fod
spring.datasource.password=fod
spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver
spring.datasource.url=jdbc:oracle:thin:@127.0.0.1:1521/XE
```

- Spring Boot به طور پیش فرض از datasource HikariDataSource استفاده میکند که یک connection pool محسوب میشود

چند نمونه

```
int rowCount = jdbcTemplate.queryForObject("select count(*) from Student" , Integer.class);

int age = jdbcTemplate.queryForObject( "select age from Student where id = ?" , Integer.class, 10);

Student student = jdbcTemplate.queryForObject("select * from Student where id = ?",
        new StudentMapper() , 10);

List<Student> students = jdbcTemplate.query("select * from Student", new StudentMapper());

public class StudentMapper implements RowMapper<Student> {

    public Student mapRow(ResultSet rs, int rowNum) throws SQLException {
        Student student = new Student();
        student.setID(rs.getInt("id"));
        student.setName(rs.getString("name"));
        student.setAge(rs.getInt("age"));
        return student;
    }
}
```

چند نمونه

```
jdbcTemplate.update("update Persons set first_name = ? where person_id = ?",
“Mike”, 1001L);

jdbcTemplate.update(“delete Persons where person_id = ?", 1001L);

String ddl = "CREATE TABLE Student( " +
        "ID   INT NOT NULL AUTO_INCREMENT, " +
        "NAME VARCHAR(20) NOT NULL, " +
        "AGE  INT NOT NULL, " +
        "PRIMARY KEY (ID));";
jdbcTemplate.execute( ddl );
```

Spring JPA

- Spring Boot به طور پیش فرض از فریم ورک hibernate پشتیبانی میکند ، با تنظیم کردن datasource همه چیز مهیای استفاده کردن میباشد

```
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
```

Spring JPA

```
@PersistenceContext
private EntityManager entityManager;

@Transactional(readOnly = true)
public List<AvailableLanguages> fetchAll() {
    return entityManager.createQuery("from AvailableLanguages")
       .getResultList();
}
```

Spring Data Repository

- هدف Spring data repository کم کردن حجم کدهای در دسترسی دیتا میباشد .
- برای این موضوع یک abstraction تعریف شده است .
- روش کار به این گونه است تمامی نیاز های دسترسی با دیتا ها را در قالب یک interface به شکل declarative صرفاً تعریف میکنیم

https://docs.spring.io/spring-data/jpa/docs/2.0.6.RELEASE/reference/html/

CrudRepository

- Interface CrudRepository یک سری امکانات برای عملیات CRUD در اختیار میگذارد

```
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);

}
```

PagingAndSortingRepository

- Interface PagingAndSortingRepository امکانات مرتب سازی و pagination میدهد

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

    Iterable<T> findAll(Sort sort);

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

تولید Query

- مکانیزم تولید Query ها با روشی به نام Query Method ها صورت میپذیرد
- مثال

```
List<User> findByEmailAddressAndLastname(String emailAddress
                                        , String lastname);
```

- بدون نیاز به پیاده سازی چنین Query تولید میشود

```
select u from User u where u.emailAddress = ?1 and u.lastname = ?2
```

Query Methods انواع

findBy**Lastname**And**Firstname**	findBy**Firstname**Like
findBy**Lastname**Or**Firstname**	findBy**Firstname**NotLike
findBy**Firstname** , findBy**Firstname**Is , findBy**Firstname**Equals	findBy**Firstname**StartingWith
findBy**StartDate**Between	findBy**Firstname**EndingWith
findBy**Age**LessThan	findBy**Firstname**Containing
findBy**Age**LessThanEqual	findBy**Age**OrderBy**Lastname**Desc
findBy**Age**GreaterThan	findBy**Lastname**Not
findBy**Age**GreaterThanEqual	findBy**Age**In(Collection ages)
findBy**StartDate**After	findBy**Age**NotIn(Collection ages)
findBy**StartDate**Before	findBy**Active**True()
findBy**Age**IsNull	findBy**Active**False()
findBy**Age**IsNotNull	findBy**Firstname**IgnoreCase

تولید Query

- میتوان از @Query برای مشخص کردن query مورد نظر استفاده کرد

```
@Query("select u from AvailableLanguages u where u.language like %?1 ")
List<AvailableLanguages> findAllByLanguage(String language);

@Query("select u from AvailableLanguages u where u.language like :PLanguage ")
List<AvailableLanguages> findAllByLanguageNamed(@Param("PLanguage") String language);

@Query(name = "findAllAvailableLanguage")
List<AvailableLanguages> findAllByNamedQuery();

@Query(value = "select * from Available_Languages u where u.language like '%'||?1 "  ,
nativeQuery = true)
List<AvailableLanguages> findAllByLanguageNative(String language);
```

Modifying Query

- میتوان از @Query برای عملیات تغییر دیتا نیز استفاده کرد

```
@Modifying
@Query("update User u set u.firstname = ?1 where u.lastname = ?2")
int setFixedFirstnameFor(String firstname, String lastname);


@Modifying
@Query("delete from User u where user.role.id = ?1")
void deleteInBulkByRoleId(long roleId);
```