Spring Data &
Spring Data REST

Agenda

- Type-safe queries
- Spring Data JPA
- Spring Data JPA & Java 8
- Other Spring Data projects
- Spring Data REST
- Practical Example

Type safe queries for JPA

- http://www.jinq.org/docs/

Spring Data

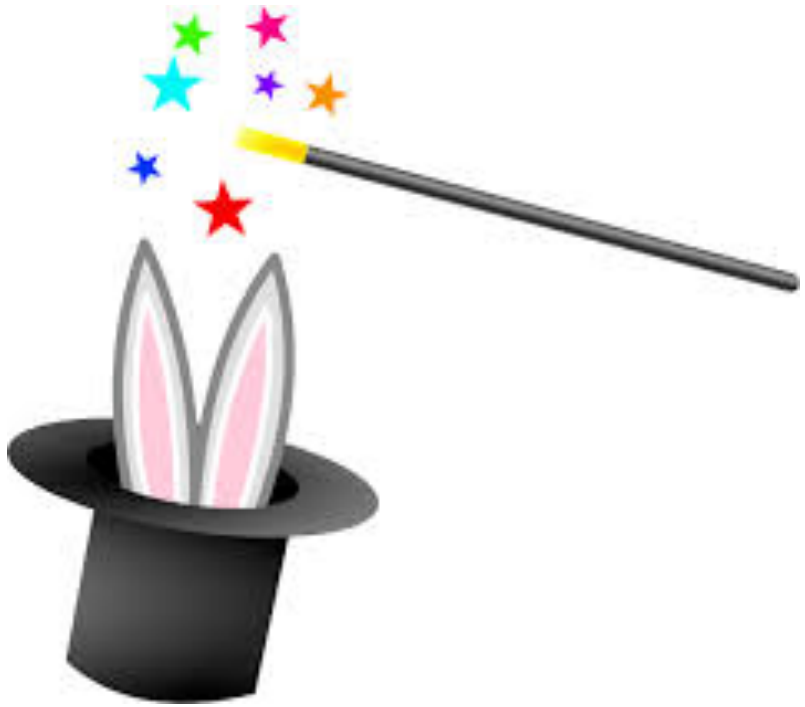
Spring Data Rest Dependency

```
dependencies {
  ...
  compile("org.springframework.boot:spring-boot-starter-data-rest")
  ...
}
```

Enable Spring Data JPA

- @EnableJpaRepositories(basePackages=…)

What is Spring Data



Magic

```
public interface UserRepository extends Repository<User, Long> {

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

Spring Data will create JPA query “select u from User u where u.emailAddress = ?1 and u.lastname = ?2”

More Magic

Keyword	Sample	JPQL snippet
And	`findByLastnameAndFirstname`	`… where x.lastname = ?1 and x.firstname = ?2`
Or	`findByLastnameOrFirstname`	`… where x.lastname = ?1 or x.firstname = ?2`
Is,Equals	`findByFirstname`, `findByFirstnameIs`, `findByFirstnameEquals`	`… where x.firstname = ?1`
Between	`findByStartDateBetween`	`… where x.startDate between ?1 and ?2`
LessThan	`findByAgeLessThan`	`… where x.age < ?1`
LessThanEqual	`findByAgeLessThanEqual`	`… where x.age <= ?1`
GreaterThan	`findByAgeGreaterThan`	`… where x.age > ?1`
GreaterThanEqual	`findByAgeGreaterThanEqual`	`… where x.age >= ?1`
After	`findByStartDateAfter`	`… where x.startDate > ?1`

- More magic at https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#jpa.query-methods

Manual queries

```
public interface UserRepository extends Repository<User, Long> {

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

Named Parameters

```
public interface UserRepository extends Repository<User, Long> {

  @Query("select u from User u where u.firstname = :firstname or u.lastname = :lastname")
  User findByLastnameOrFirstname(@Param("lastname") String lastname,
                                 @Param("firstname") String firstname);
}
```

Sorting

```
public interface UserRepository extends Repository<User, Long> {

  @Query("select u from User u where u.lastname like ?1%")
  List<User> findByAndSort(String lastname, Sort sort);


}
```

Pagination

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

```
Pageable pageable = new PageRequest(2, 10, Direction.ASC, "lastname",
"firstname");
Page<Customer> result = findByLastname("Matthews", pageable);
```

Defining Repositories

- **Repository** A plain marker interface to let the Spring Data infrastructure pick up user-defined repositories. Domain repositories extending this interface can selectively expose CRUD methods by simply declaring methods of the same signature as those declared in CrudRepository.
- **CrudRepository** Extends Repository and adds basic persistence methods like saving, finding, and deleting entities
- **PagingAndSortingRepositories** Extends CrudRepository and adds methods for accessing entities page by page and sorting them by given criteria

Defining Repository

When you extends CrudRepository or PagingAndSortingRepository, then you will have all methods defined there

```
public interface CrudRepository<T, ID extends Serializable> extends Repository<T, ID> {
        <S extends T> save(S entity);
        <S extends T> Iterable<S> save(Iterable<S> entities);
        T findOne(ID id);
        Iterable<T> findAll();
        void delete(ID id);
        void delete(T entity);
        void deleteAll();
}
```

```
public interface PagingAndSortingRepository<T, ID extends Serializable>
                                extends CrudRepository<T, ID> {
        Iterable<T> findAll(Sort sort);
        Page<T> findAll(Pageable pageable);
}
```

@NoRepositoryBean

- Define your own repository interface to inherit, Spring Data will skip it and will not create bean for it.

Manually implement repository methods

```
interface CustomerRepositoryCustom {
        Customer myCustomMethod(...);
}
class CustomerRepositoryCustomImpl implements
                CustomerRepositoryCustom {
        // Potentially wire dependencies
        public Customer myCustomMethod(...) {
        // custom implementation code goes here
        }
}
```

```
public interface CustomerRepository
        extends CrudRepository<Customer, Long>,
        CustomerRepositoryCustom { ... }
```

- How this magic works:
  - SpringData looks for repository interface suffixed by Impl and instantiates it

Transactions

- Use @Transactional over methods/interface

Spring Data & Java 8

https://github.com/spring-projects/spring-data-examples/tree/master/jpa/java8

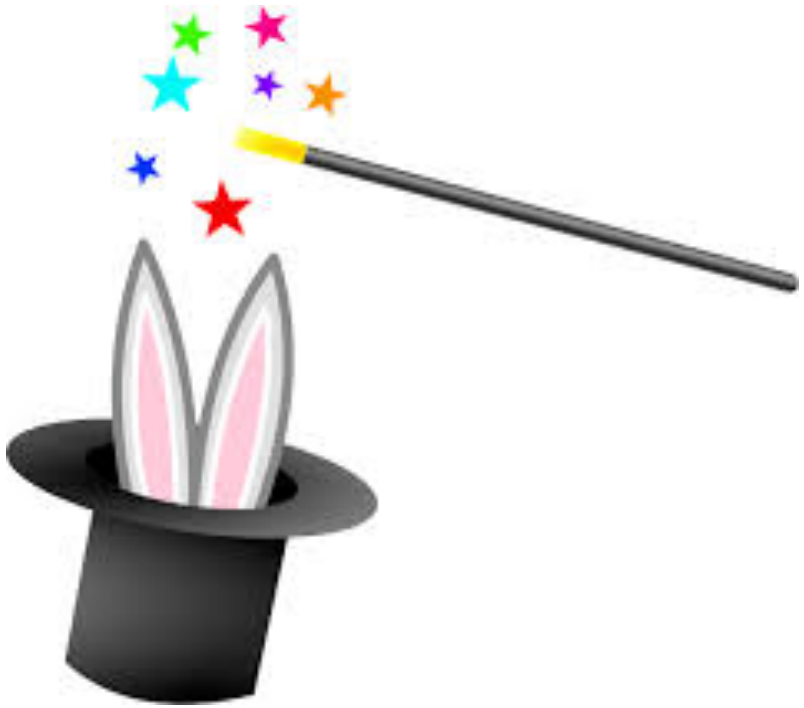
- Optional
- Stream
- CompletableFuture & Async
  - No transactional context in this case

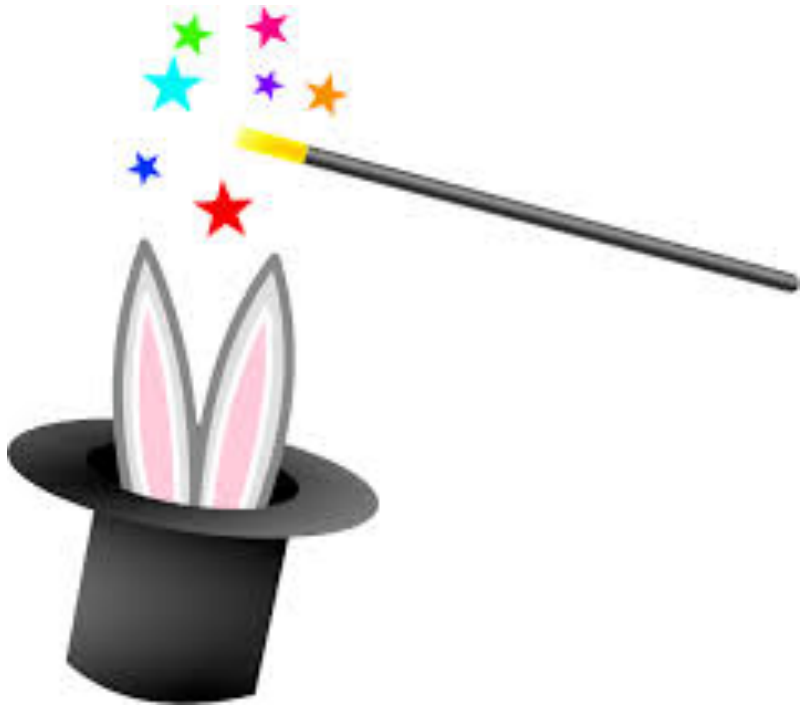
Other Spring Data Projects

- Spring Data JPA
- Spring Data LDAP
- Spring Data Mongo
- Spring Data Cassandra
- Spring Data Elasticsearch
- …

Spring Data REST

What is Spring Data REST





Dependency

```
dependencies {
   ...
   compile("org.springframework.boot:spring-boot-starter-data-rest")
   ...
}
```

Auto-configuration of Spring Data REST

- Done as we have Spring Boot (RepositoryRestMvcConfiguration is created)

Practical example

- Remove REST controller and run application

Practical Example

- Run practical example and see log
- Lets' figure out what are that links

```
Mapped "{[/ || ],methods=[OPTIONS],produces=[application/hal+json || application/json || appli
Mapped "{[/ || ],methods=[HEAD],produces=[application/hal+json || application/json || applicat
Mapped "{[/ || ],methods=[GET],produces=[application/hal+json || application/json || applicati
Mapped "{[/{repository}],methods=[OPTIONS],produces=[application/hal+json || application/json
Mapped "{[/{repository}],methods=[HEAD],produces=[application/hal+json || application/json ||
Mapped "{[/{repository}],methods=[GET],produces=[application/hal+json || application/json || a
Mapped "{[/{repository}],methods=[GET],produces=[application/x-spring-data-compact+json || tex
Mapped "{[/{repository}],methods=[POST],produces=[application/hal+json || application/json ||
Mapped "{[/{repository}/{id}],methods=[OPTIONS],produces=[application/hal+json || application/
Mapped "{[/{repository}/{id}],methods=[HEAD],produces=[application/hal+json || application/jso
Mapped "{[/{repository}/{id}],methods=[GET],produces=[application/hal+json || application/json
Mapped "{[/{repository}/{id}],methods=[PUT],produces=[application/hal+json || application/json
Mapped "{[/{repository}/{id}],methods=[PATCH],produces=[application/hal+json || application/js
Mapped "{[/{repository}/{id}],methods=[DELETE],produces=[application/hal+json || application/j
Mapped "{[/{repository}/{id}/{property}],methods=[GET],produces=[application/hal+json || appli
Mapped "{[/{repository}/{id}/{property}/{propertyId}],methods=[GET],produces=[application/hal+
Mapped "{[/{repository}/{id}/{property}],methods=[DELETE],produces=[application/hal+json || ap
Mapped "{[/{repository}/{id}/{property}],methods=[GET],produces=[application/x-spring-data-com
Mapped "{[/{repository}/{id}/{property}],methods=[PATCH || PUT || POST],consumes=[application/
Mapped "{[/{repository}/{id}/{property}/{propertyId}],methods=[DELETE],produces=[application/h
Mapped "{[/{repository}/search],methods=[HEAD],produces=[application/hal+json || application/j
Mapped "{[/{repository}/search],methods=[OPTIONS],produces=[application/hal+json || applicatio
Mapped "{[/{repository}/search],methods=[GET],produces=[application/hal+json || application/js
Mapped "{[/{repository}/search/{search}],methods=[GET],produces=[application/hal+json || appli
Mapped "{[/{repository}/search/{search}],methods=[GET],produces=[application/x-spring-data-com
Mapped "{[/{repository}/search/{search}],methods=[OPTIONS],produces=[application/hal+json || a
Mapped "{[/{repository}/search/{search}],methods=[HEAD],produces=[application/hal+json || appl
Mapped "{[/profile/{repository}],methods=[GET],produces=[application/alps+json || */*]}" onto
Mapped "{[/profile/{repository}],methods=[OPTIONS],produces=[application/alps+json]}" onto org
```

/{repository}

- repository = uncapitalized, pluralized, simple class name of the domain class being managed

- **public interface** UserDao **extends** Repository<User, Integer> { ..}

- /{repository} = /users

HATEOS & HAL

- A hypermedia-driven site provides information to navigate the site's REST interfaces dynamically by including hypermedia links with the responses
- HAL (JSON Hypertext Application Language) – standard link names
- HATEOS (Hypermedia as the Engine of Application State) - principle

Discover resources

- Discover repositories
  - http://localhost:8080/
- Discover other resources
  - http://localhost:8080/users/
  - http://localhost:8080/users/search

- There are
  - Collection resources
  - Item resources

For convenience add HAL browser to app

```
dependencies {
")
   compile ('org.springframework.data:spring-data-rest-hal-browser')
}
```

- http://localhost:8080/

Hiding repository methods

```
@RepositoryRestResource(exported = false)
interface PersonRepository extends CrudRepository<Person, Long> {}
```

```
interface PersonRepository extends CrudRepository<Person, Long> {

  @RestResource(exported = false)
  List<Person> findByName(String name);

  @Override
  @RestResource(exported = false)
  void delete(Long id);

}
```

Paging & sorting

- http://localhost:8080/users/search/findBySurnameContaining?surname=Sur&page=0&size=5&sort=age

Bibliography

- Spring Data Book (search Spring Data Book PDF + https://github.com/spring-projects/spring-data-book)
- Spring Data JPA – Reference Documentation https://docs.spring.io/spring-data/jpa/docs/current/reference/html/
- Spring Data REST – Reference Doc http://docs.spring.io/spring-data/rest/docs/current/reference/html/
- Spring Data Examples https://github.com/spring-projects/spring-data-examples

Домашечка

- Применить Spring Data и Spring Data REST в своем проекте