Spring Data

Craig Walls
craig@habuma.com
Twitter: @habuma
http://github.com/habuma

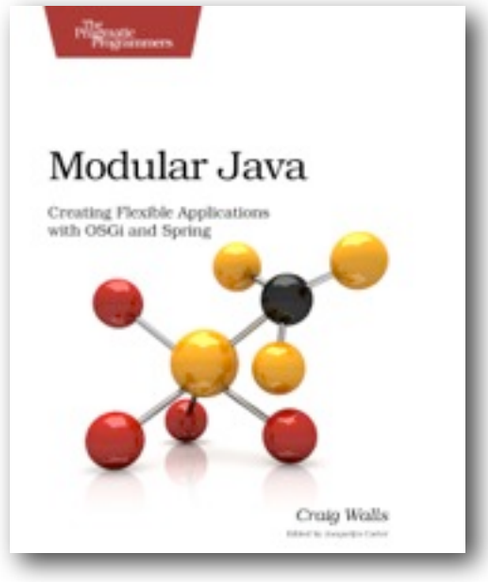
Who am I?

Java and Spring Fanatic

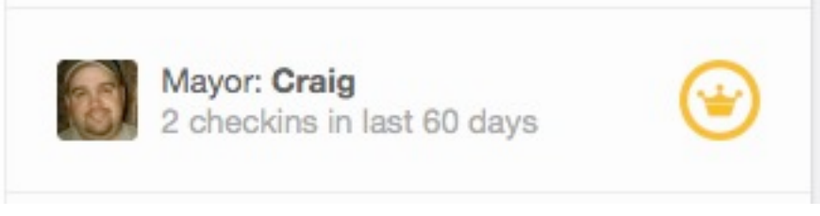
Senior Engineer with SpringSource

Spring Social Project Lead

Author



Mayor of "Post Office, Jal NM" on Foursquare



Email: craig@habuma.com Twitter: @habuma Blog: http://www.springinaction.com Sample Code: http://github.com/habuma

Spring Data Overview

Under the Spring Data Umbrella

- Relational
  - JPA, JDBC extensions
- Big Data
  - Hadoop
- Data Grid
  - GemFire
- Key Value Stores
  - Redis, Riak
- Document Stores
  - MongoDB, CouchDB*
- Graph Databases
  - Neo4j
- Column Stores
  - HBase*, Cassandra*
- Blob-Stores
  - such as Amazon S3
- JPA REST exporter

* Planned

Spring Data Features

Template-oriented data access

Auto-magic repositories

CRUD repositories

Paging and sorting repositories

GeoSpatial repositories

Custom repository methods

Object-to-store mapping annotations

Just Enough Spring Data for Today...

Relational DB

Spring Data JPA

Document Store

Spring Data MongoDB

Graph Database

Spring Data Neo4j

Key-Value

Spring Data Redis

Spring Data JPA

Spring Data JPA features

The JPA specification and Spring's existing JPA support already offer...

JpaTemplate

Mapping annotations

Therefore, all that's left for Spring Data JPA to provide is...

JPA repositories

JPA Repositories

```
public interface OrderRepository extends JpaRepository<Order, Long> {
   List<Order> findByCustomer(String customer);

   List<Order> findByCustomerLike(String customer);

   List<Order> findByCustomerAndType(String customer, String type);

   List<Order> findByCustomerLikeAndType(String customer, String type);

   @Query("select o from Order o where o.customer = 'Chuck Wagon' and o.type = ?1")
   List<Order> findChucksOrders(String type);
}
```

```
<jpa:repositories base-package="com.habuma.samples" />
```

Repository method naming

Supports these keywords:

And, Or, Between, LessThan, GreaterThan, IsNull, IsNotNull, NotNull, Like, NotLike, OrderBy, Not, In, NotIn

Examples:

```
findByEmailAddressAndLastname()

findByCompanyNameLike()

findByLastNameOrFirstName()

findByAddressZipCode()

findByAddress_ZipCode()
```

Using a JPA repository

```
Order savedOrder = orderRepository.save(order);
```

```
long orderCount = orderRepository.count();
```

```
Order order = orderRepository.findOne(orderId);
```

```
List<Order> chucksOrders = orderRepository.findByCustomer("Chuck Wagon");
```

```
List<Order> chucksOrders = orderRepository.findByCustomerLike("Chuck%");
```

```
List<Order> chucksWebOrders = orderRepository.findByCustomerAndType("Chuck Wagon", "WEB");
```

```
List<Order> chucksWebOrders = orderRepository.findByCustomerLikeAndType("Chuck%", "WEB");
```

```
List<Order> chucksWebOrders = orderRepository.findChucksOrders("WEB");
```

Spring Data MongoDB

Spring Data MongoDB Features

MongoTemplate

Mongo repositories

Annotation-based object-to-document mapping

Geo-Spatial queries

Cross-store persistence

Annotating the domain

```
@Document
public class Order {

    @Id
    private String id;

    private String customer;

    private String type;

    private Collection<Item> items = new LinkedHashSet<Item>();

...
}
```

Using MongoTemplate

```
@Autowired MongoOperations mongoOps;
```

```
mongoOps.save(order, "order");
```

```
List<Order> allOrders = mongoOps.findAll(Order.class);
```

```
Order order = mongoOps.findById(order.getId(), orderId);
```

```
List<Order> chucksOrders = mongoOps.find(
    Query.query(Criteria.where("customer").is("Chuck Wagon")), Order.class);
```

```
List<Order> chucksWebOrders = mongoOps.find(
    Query.query(Criteria.where("customer").is("Chuck Wagon")
                        .and("type").is("WEB")), Order.class);
```

```
mongoOps.remove(order);
```

Mongo repositories

```
public interface OrderRepository extends MongoRepository<Order, String> {
   List<Order> findByCustomer(String customer);

   List<Order> findByCustomerLike(String customer);

   List<Order> findByCustomerAndType(String customer, String type);

   @Query("{customer:'Chuck Wagon'}")
   List<Order> findChucksOrders();
}
```

```
<mongo:repositories base-package="com.habuma.samples" />
```

Using a Mongo repository

```
@Autowired OrderRepository orderRepository;
```

```
Order savedOrder = orderRepository.save(order);
```

```
long orderCount = orderRepository.count();
```

```
Order order = orderRepository.findOne(savedOrder.getId());
```

```
List<Order> chucksOrders = orderRepository.findByCustomer("Chuck Wagon");
```

```
List<Order> chuckLikeOrders = orderRepository.findByCustomerLike("Chuck");
```

```
List<Order> webOrders = orderRepository.findByCustomerAndType("Chuck Wagon", "WEB");
```

```
orderRepository.delete(savedOrder.getId());
```

```
orderRepository.deleteAll();
```

Spring Data Neo4j

Spring Data Neo4j Features

Neo4jTemplate

Neo4j repositories

Annotation-based object-to-graph mapping

Cross-store persistence

Annotating the domain

```
@NodeEntity
public class Place {
    @GraphId
    private Long graphId;

    @Indexed(indexType=IndexType.FULLTEXT, indexName = "placeSearch")
    private String name;

    @RelatedTo(type="IS_A_PLACE_IN")
    private Area area;

    private PlaceType type;
    private float latitude;
    private float longitude;
    private boolean active;
    private String foursquareId;
    private String facebookId;
...
}
```

Configuring Neo4j in Spring

```
<?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:neo4j="http://www.springframework.org/schema/data/neo4j"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/data/neo4j
        http://www.springframework.org/schema/data/neo4j/spring-neo4j-2.0.xsd">

    <neo4j:config storeDirectory="/Users/habuma/Projects/SampleData/DisneyData" />

    <neo4j:repositories base-package="com.mouseguests"  />

</beans>
```

Using Neo4jTemplate

```
@Inject Neo4jOperations neo4j;
```

```
long characterCount = neo4j.count(Character.class);
```

```
Iterable<Character> characters = neo4j.findAll(Character.class);
```

```
Character pinocchio = null;
Character savedCharacter = neo4j.save(pinocchio);
```

```
Appearance appearance = neo4j.createRelationshipBetween(
     character, place, Appearance.class, "WAS_SEEN_AT", false);
```

```
neo4j.delete(pinocchio);
```

Creating a Neo4j repository

```
public interface CharacterRepository extends GraphRepository<Character>,
                                             NamedIndexRepository<Character> {
}
```

```
public interface PlaceRepository extends GraphRepository<Place>,
NamedIndexRepository<Place>, SpatialRepository<Place> {

   @Query("START area=node({0}) MATCH place-[:IS_A_PLACE_IN]->area " +
          "RETURN place ORDER BY place.name")
   List<Place> findByArea(Long areaId);

   @Query("START place=node({0}) MATCH ()-[checkin]->place " +
          "RETURN checkin ORDER BY checkin.date DESC LIMIT 10")
   List<Checkin> findCheckinsForPlace(long placeId);

}
```

```
<neo4j:repositories base-package="com.mouseguests"  />
```

Using a Neo4j repository

```
Character character = new Character(name, knownFrom, tags);
characterRepository.save(character);
```

```
Character character = characterRepository.findOne(characterId);
```

```
Iterable<Character> characters = characterRepository.findAllByQuery(
        "characterSearch", "name", wildcardify(query));
```

```
Iterable<Place> places = placeRepository.findByArea(areaId);
```

GeoSpatial queries in the graph

```
@NodeEntity
public class Place {

...

    @SuppressWarnings("unused")
    @Indexed(indexName = "placeLocation", indexType = IndexType.POINT)
    private String wkt;

    public Place(String name, Area area, PlaceType type,
                 float latitude, float longitude,
                 boolean active, String foursquareId, String facebookId) {
         ...

        this.wkt = String.format("POINT( %.6f %.6f )",longitude, latitude);

         ...
    }

...
}
```

```
Iterable<Place> nearby = placeRepository.findWithinDistance(
                                    "placeLocation", longitude, latitude, range);
```

Spring Data Redis

Spring Data Redis Features

Redis connection factories

Jedis, JRedis, and RJS connector support

RedisTemplate and StringRedisTemplate

Redis-oriented Pub/Sub

Connection Factories

Jedis

```
<bean id="jedisConnectionFactory"
      class="org.springframework.data.redis.connection.jedis.JedisConnectionFactory"/>
```

```
<bean id="jedisConnectionFactory"
      class="org.springframework.data.redis.connection.jedis.JedisConnectionFactory"
      p:host-name="server"
      p:port="6379"/>
```

JRedis

```
<bean id="jredisConnectionFactory"
      class="org.springframework.data.redis.connection.jredis.JredisConnectionFactory"/>
```

RJC

```
<bean id="rjcConnectionFactory"
      class="org.springframework.data.redis.connection.rjc.RjcConnectionFactory"/>
```

RedisTemplate

```
<bean id="redisTemplate"
      class="org.springframework.data.redis.core.RedisTemplate"
      p:connection-factory-ref="jedisConnectionFactory"/>
```

```
<bean id="stringRedisTemplate"
      class="org.springframework.data.redis.core.StringRedisTemplate"
      p:connection-factory-ref="jedisConnectionFactory"/>
```

```
@Inject
private RedisTemplate<String, Product> redisOps;

public void addProduct(Product product) {
   redisOps.opsForValue().set(product.getSku(), product);
}
```

Redis PubSub: Sending messages

Low Level: Through RedisConnection

```
byte[] message = ...;
byte[] channel = ...;

redisConnection.publish(message, channel);
```

High Level: Via RedisTemplate

```
redisTemplate.convertAndSend("Hey there!", "greetingsChannel");
```

Redis PubSub: Receiving messages

Low Level: Via RedisConnection

Possible, but not fun.
Involves blocking and requires thread-management.
Let's not discuss it any further

High Level: With a listener container

```
public interface MessageHandler {
  void handleMessage(String message);
  void handleMessage(Map message);
  void handleMessage(byte[] message);
  void handleMessage(Serializable message);
}
```

```
<redis:listener-container>
  <redis:listener ref="listener" method="handleMessage" topic="greetingChannel" />
</redis:listener-container>

<bean id="listener" class="MyMessageHandler"/>
```

Q & A

Don’t forget to turn in your evals