



Spring Data Repositories

1. Understanding Spring Data repositories
2. Using a Spring Data repository

Annex:

- Accessing MongoDB

1. Understanding Spring Data Repositories

- Overview of Spring Data repositories
- Spring Data repository capabilities
- Paging and sorting
- Technology-specific repositories
- Domain-specific repositories

Overview of Spring Data Repositories

- Spring Data is a data-access abstraction mechanism
 - Makes it very easy to access a wide range of data stores
 - Using a familiar "repository" pattern
 - Create / Read / Update / Delete (CRUD)
- It provides template repositories for...
 - JPA
 - MongoDB, Cassandra, CouchBase
 - Etc.

Spring Data Repository Capabilities

- Spring Data defines a general-purpose repository interface:

```
public interface CrudRepository<T,ID> extends Repository<T,ID> {  
  
    long count();  
  
    void delete(T entity);  
  
    void deleteAll();  
  
    void deleteAll(Iterable<T> entities);  
  
    void deleteById(ID id);  
  
    boolean existsById(ID id);  
  
    Iterable<T> findAll();  
  
    Iterable<T> findAllById(Iterable<ID> ids);  
  
    Optional<T> findById(ID id);  
  
    T save(T entity);  
  
    Iterable<T> saveAll(Iterable<T> entities);  
  
}
```



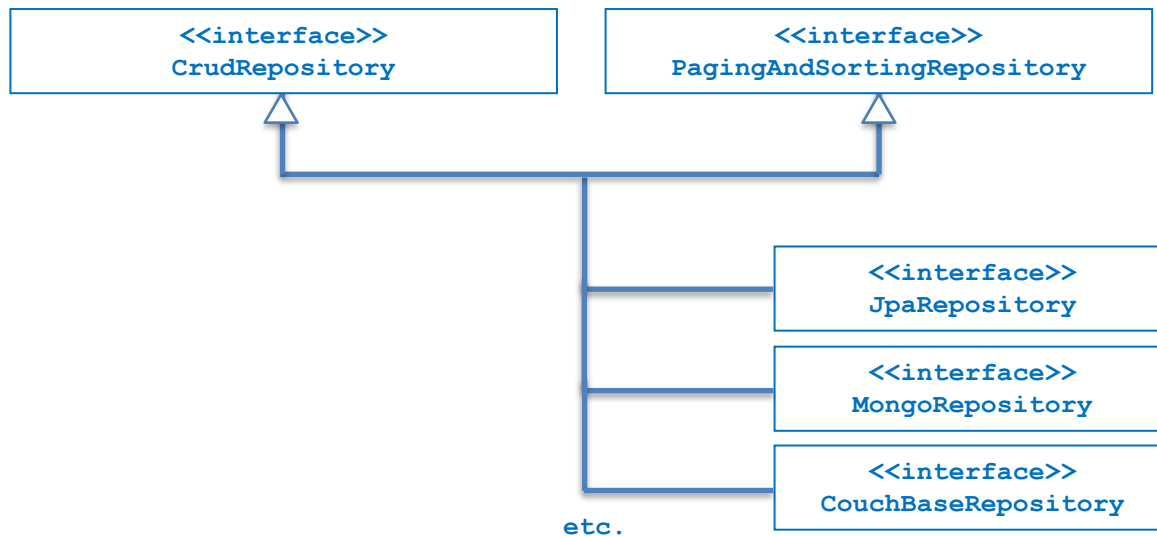
Paging and Sorting

- Support for paging and sorting is provided via this interface:

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

Technology-Specific Repositories

- Spring Data also provides technology-specific repositories
 - Provide technology-specific extensions



Domain-Specific Repositories

- You can define your own domain-specific interfaces
 - Extend `CrudRepository` (or sub-interface)
 - Specify the entity type and the PK type
- You can define specific query methods for your entities
 - Spring Data reflects on method names to create queries
 - You can provide an explicit query string for complex queries
- See next section for an example...

2. Using a Spring Data Repository

- Overview
- Defining a repository
- Locating Spring Data repositories
- Using a Spring Data repository

Overview

- In this section we'll see how to access a relational database by using a Spring Data repository
- Note the following key points in the demo first:
 - `pom.xml`
 - `application.properties`
 - `Employee.java`
 - `SeedDb.java`

Defining a Repository

- Here's an example of a domain-specific repository:

```
public interface EmployeeRepository extends CrudRepository<Employee, Long> {  
  
    List<Employee> findByRegion(String region);  
  
    @Query("select e from Employee e where e.dosh >= ?1 and e.dosh <= ?2")  
    List<Employee> findInSalaryRange(double from, double to);  
  
    Page<Employee> findByDoshGreaterThan(double salary, Pageable pageable);  
  
}
```

EmployeeRepository.java

- Note:
 - Entity type is `Employee`, PK type is `Long`
 - Also, we've defined some custom queries

Locating Spring Data Repositories

- A Spring Boot application scans for Spring Data JPA repository interfaces when it starts
 - It looks in the main application class package, plus sub-packages
- You can tell it to look elsewhere, if you like
 - Via `@EnableJpaRepositories`

```
import org.springframework.data.jpa.repository.config.EnableJpaRepositories;  
...  
  
@SpringBootApplication  
@EnableJpaRepositories({"repopackage1", "repopackage2"})  
public class Application {  
    ...  
}
```

Using a Spring Data Repository (1 of 2)

- Let's see how to use some standard repository methods:

```
@Component
public class EmployeeService {

    @Autowired
    private EmployeeRepository repository;

    public void useStandardRepoMethods() {

        // Insert an employee.
        Employee newEmp = new Employee(-1, "Simon Peter", 10000, "Israel");
        newEmp = repository.save(newEmp);
        System.out.printf("Inserted employee, id %d\n", newEmp.getEmployeeId());

        // Get count of all employees.
        System.out.printf("There are now %d employees\n", repository.count());

        // Get all employees.
        displayEmployees("All employees: ", repository.findAll());
    }
    ...
}
```

EmployeeService.java

Using a Spring Data Repository (2 of 2)

- Let's see how to use our custom queries in the repository:

```
@Component
public class EmployeeService {

    @Autowired
    private EmployeeRepository repository;

    public void useCustomQueryMethods() {

        // Get all employees by region.
        displayEmployees("All employees in London: ", repository.findByRegion("London"));

        // Get employees by salary range.
        List<Employee> emps = repository.findInSalaryRange(20000, 50000);
        displayEmployees("Employees earning 20k to 50k: ", emps);

        // Get a page of employees.
        Pageable pageable = PageRequest.of(1, 3, Direction.DESC, "dosh");
        Page<Employee> page = repository.findByDoshGreaterThan(50000, pageable);
        displayEmployees("Page 1 of employees more than 50k: ", page.getContent());
    }
}
```

EmployeeService.java

A large, light gray play button icon is positioned on the left side of the slide. It consists of a white right-pointing triangle centered within a series of concentric circles in varying shades of gray.

Summary

- Understanding Spring Data repositories
- Using a Spring Data repository



Annex: Accessing MongoDB

- What is MongoDB?
- Getting MongoDB
- Using MongoDB in Spring Boot
- Spring Boot APIs for MongoDB
- Defining a MongoDB entity class
- Defining a repository
- Seeding the MongoDB database
- Other useful info about the demo app

Overview of MongoDB (1 of 2)

- MongoDB is an open-source document database
 - In MongoDB, a document is a BSON object ("binary JSON")
 - A document contains fieldname/value pairs
 - Values can be simple types, arrays, or nested documents
- Here's an example of a MongoDB document:

```
{
  name: "Kari Nordmann",
  age: 25,
  skills: [ "Java", "Spring Boot", "Cobol" ],
  additionalInfo: {
    nationality: "Norsk",
    companyCar: {
      make: 'Bugatti',
      model: 'Divo'
    }
  }
}
```


Overview of MongoDB (2 of 2)

- High performance
 - Via indexes
- Rich query language for CRUD operations
 - Data aggregation
 - Text search and queries
- High availability
 - Via automatic failover and data redundancy
- Horizontal scalability across a cluster
 - Via sharding

Getting MongoDB for Mac (1 of 2)

- Open a Terminal window and type the following:

```
brew update  
brew install mongodb
```

- After downloading Mongo, create the "db" directory
 - This is where the Mongo data files will live
 - You can create the directory in the default location as follows:

```
mkdir -p /data/db
```

- Make sure the /data/db directory has correct permissions

```
> sudo chown -R `id -un` /data/db  
> # Enter your password
```

Getting MongoDB for Mac (2 of 2)

- Run the Mongo daemon as follows:

```
mongod
```

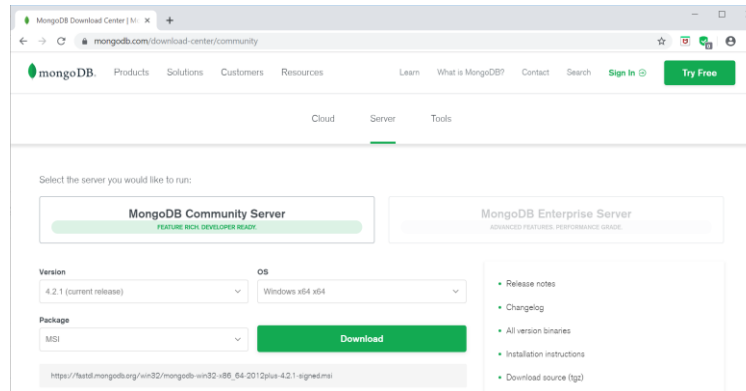
- Then open another Terminal window and run the Mongo shell as follows:

```
mongo
```

- You can then enter commands in the Mongo shell to create data, query data, etc.
- When you're ready to stop:
 - To exit the mongo shell, type `quit()`
 - To stop the Mongo daemon, hit `ctrl+c`

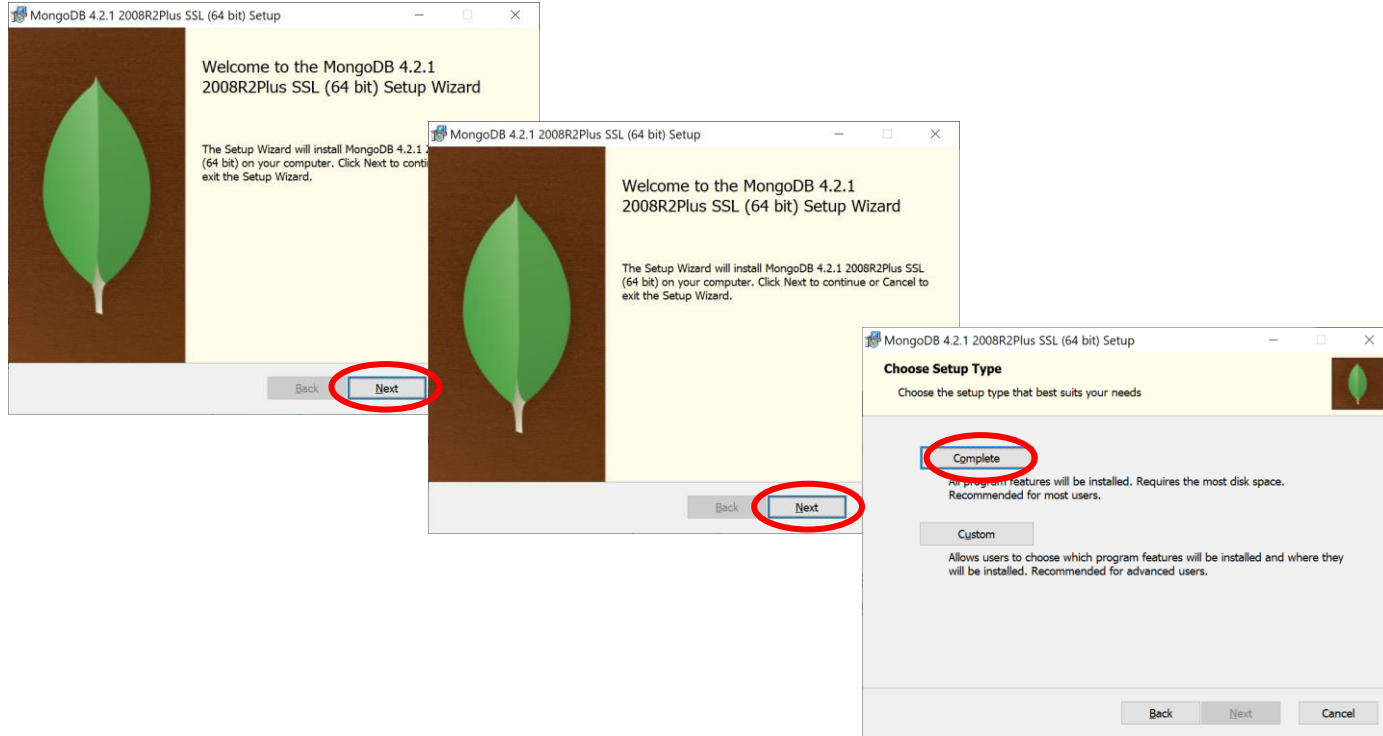
Getting MongoDB for Windows (1 of 5)

- This section shows how to install MongoDB Community Edition on Windows...
 - Requires Windows Server 2008 R2, Windows Vista, or later
- Go to the download page for MongoDB Community Edition
 - <https://www.mongodb.com/download-center#community>



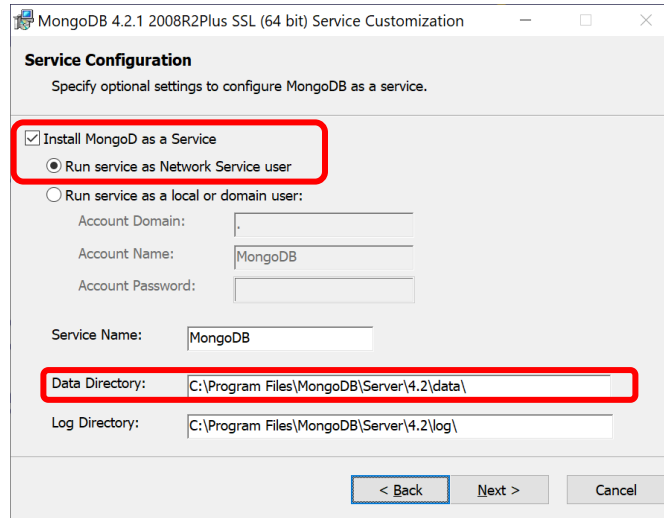
Getting MongoDB for Windows (2 of 5)

- When the MongoDB msi has downloaded, run it



Getting MongoDB for Windows (3 of 5)

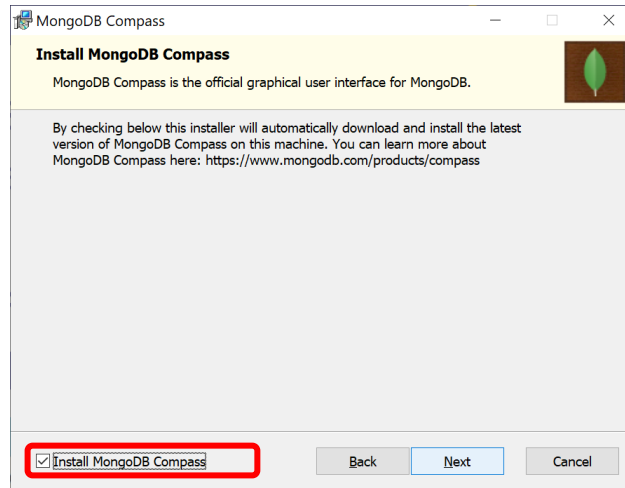
- It's possible to install MongoDB as a Windows Service
 - MongoDB starts automatically when the machine boots up
- MongoDB requires a data directory to store all data
 - You can accept the default location, or specify a different location



The screenshot shows the 'MongoDB 4.2.1 2008R2Plus SSL (64 bit) Service Customization' window. The 'Service Configuration' tab is active, with the instruction 'Specify optional settings to configure MongoDB as a service.' Below this, the 'Install MongoDB as a Service' checkbox is checked and highlighted with a red rectangle. Underneath, the radio button for 'Run service as Network Service user' is selected. The 'Run service as a local or domain user:' section contains fields for 'Account Domain:', 'Account Name:' (containing 'MongoDB'), and 'Account Password:'. The 'Service Name:' field contains 'MongoDB'. The 'Data Directory:' field, which contains 'C:\Program Files\MongoDB\Server\4.2\data\', is highlighted with a red rectangle. The 'Log Directory:' field contains 'C:\Program Files\MongoDB\Server\4.2\log\'. At the bottom, there are '< Back', 'Next >', and 'Cancel' buttons.

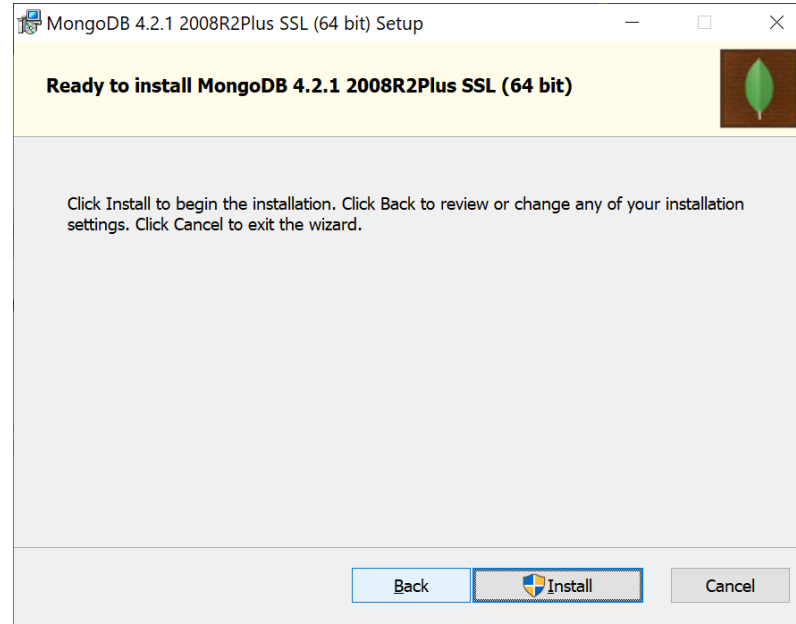
Getting MongoDB for Windows (4 of 5)

- You can choose to install MongoDB Compass
 - The official IDE for managing MongoDB
- Deselect this option if Compass is already installed, or if you want to install it later



Getting MongoDB for Windows (5 of 5)

- Proceed to begin the installation



Using MongoDB in Spring Boot

- Now let's see how to access MongoDB databases in a Spring Boot application
- You need the following Maven dependency:

```
<dependency>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-starter-data-mongodb</artifactId>  
</dependency>
```

pom.xml in demo project

- By default, Spring Boot assumes the following URI to connect to MongoDB (you can customize if necessary)

```
spring.data.mongodb.uri=mongodb://localhost:27017/test
```

application.properties

- Spring Boot has a `MongoTemplate` class
 - Allows you to perform low-level MongoDB operations
 - Similar purpose to `JdbcTemplate` (for low-level JDBC operations)
- Alternatively you can use Spring Data Repositories, similar to the JPA Data Repository earlier
 - Define an interface that extends `CrudRepository`
 - Declare method signatures, representing the queries you need
 - Spring Data implements the methods automatically, using `MongoTemplate` under the covers
 - We'll take this approach...

Defining a MongoDB Entity Class

- If you're using Spring Data Repositories to access MongoDB, you must define MongoDB entity classes
 - Similar to JPA entity classes, but note the differences highlighted:

```
import org.springframework.data.annotation.Id;  
import org.springframework.data.mongodb.core.mapping.Field;
```

```
public class Employee {
```

```
    @Id
```

```
    private long employeeId = -1;
```

```
    private String name;
```

```
    private String region;
```

```
    @Field("salary")
```

```
    private double dosh;
```

```
    ...
```

```
}
```

Employee.java

Defining a Repository

- A Spring Data repository for MongoDB is very similar to any other Spring Data repository
 - But note, the `@Query` annotation uses MongoDB query syntax!

```
import org.springframework.data.mongodb.repository.MongoRepository;
import org.springframework.data.mongodb.repository.Query;

public interface EmployeeRepository extends MongoRepository<Employee, Long> {

    List<Employee> findEmployeesByRegion(String region);

    @Query("{ 'dosh' : { $gte : ?0, $lte : ?1 } }")
    List<Employee> findEmployeesInSalaryRange(double from, double to);

    Page<Employee> findEmployeesByDoshGreaterThan(double salary, Pageable pageable);

}
```

EmployeeRepository.java

- Note:
 - We extended `MongoRepository`, for reasons explained here:
<https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#repositories.multiple-modules>

Seeding the Database

- We've defined a `SeedDb` component to create MongoDB documents at startup, and to delete them at the end

```
@Component
public class SeedDb {

    @Autowired
    private EmployeeRepository repository;

    @PostConstruct
    public void init() {
        repository.save(new Employee(1, "James", 21000, "London"));
        repository.save(new Employee(2, "Marie", 22000, "Edinburgh"));
        repository.save(new Employee(3, "Peter", 23000, "Belfast"));
        ...
    }

    @PreDestroy
    public void cleanup() {
        repository.deleteAll();
    }
}
```

SeedDb.java

Other Useful Info About the Demo App

- The MongoDB demo app is semantically equivalent to the JPA repository example earlier in the course
 - In fact, `EmployeeService.java` is identical to the JPA example
- About the demo app:
 - The demo creates `Employee` documents in the MongoDB database, manipulates them, and then deletes them at the end
 - All these tasks are achieved using the high-level Spring Data repository for MongoDB - nice 😊
- Note:
 - MongoDB must be running when you run the demo (!)

Exercise



- We've seen how to define a custom "select" method
 - Annotate a method with `@Query`
 - Specify a "select" JPQL string
- It's also possible to define a custom "modifying" method
 - Annotate with `@Query`, `@Modifying`, `@Transactional`
 - Specify an "insert", "update", or "delete" JPQL string

```
public interface EmployeeRepository extends CrudRepository<Employee, Long> {  
  
    @Query("delete from Employee e where e.dosh >= ?1 and e.dosh <= ?2")  
    @Modifying(clearAutomatically=true)  
    @Transactional  
    int deleteInSalaryRange(double from, double to);  
    ...  
}
```