Spring For Gold

Table of Contents

Dependency Injection

When we create 2 classes, [Student, Level] in the level, we can initialise the Student class by using Student student = new Student(); but when we have 1000s of classes, then this process become tedious to do. That's is where inversion of controls comes into play: We give spring the authority to create the objects of the classes

One way of achieving this is through dependency injection, all the objects of the classes will be auto created as Beans into one spring container, so whenever you need a specific Bean, you will just call it from the container. No need to create any object, spring will do that for you

Spring Boot Intialiser

We need to initialise the application at https://start.spring.io/ then provide the configurations

Open the generated bootstrap code in intellij IDE

Spring Boot API

In the com.hydottech.Springboot.Tutorial, create a package and call it controller in the controller package, create a sample controller as a java class file, call it HelloController

```
package com.hydottech.Springboot.Tutorial.controller;
import org.springframework.web.bind.annotation.GetMapping;
import
org.springframework.web.bind.annotation.RestController;
@RestController

public class HelloController {

@GetMapping("/")
   public String helloWorld(){
      return "Hello Money";
   }


}
```

Very Simple. You can run from the terminal using the command mvn spring-boot: run

Spring Boot DevTools

This dependency will help to auto run the spring application when changes have been detected

add this dependency to pom.xml

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <scope>runtime</scope>
        <optional>true</optional>
</dependency>
```

Reload the maven project to add the dependency
Apply the changes in the IDEs

Spring Boot Database Connection(H2 MEMORY)

Add the following dependencies in your pom.xml

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

<dependency>
   <groupId>com.h2database</groupId>
   <artifactId>h2</artifactId>
   <scope>runtime</scope>
</dependency>
```

In the **application.properties**, add this configurations

```
spring.application.name=Spring-Boot-Tutorial
server.port=5000
spring.h2.console.enabled=true
spring.datasource.url=jdbc:h2:mem:dcbapp
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=password
spring.jpa.database-
platform=org.hibernate.dialect.H2Dialect
```

Inside the browser, access it through this url
[http://localhost:5000/h2-console]

Input your password to make a connection

Spring Boot Components

Inside the com.hydottech.Springboot.Tutorial, create multiple packages like entity, service, repository, controller
Entity => Models [Annotations: `@Entity`]
Service => [Annotations: `@Service`]
Repository => [Annotations: `@Repository`]
Controller => For Http Methods [Annotations: `@RestController`]

Entity

In the entity package create a new Department model class file

```
package com.hydottech.Springboot.Tutorial.entity;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;

@Entity
public class Department {
   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   private Long departmentId;
   private String departmentName;
   private String departmentAddress;
   private String departmentCode;
   //To generate the getters and setters, press command +
n, select getters and setters, highlight all the field we
will generate getters and setters for, press okay
//Use the same method for their constructors
//departmentId is a primary key and it will be auto
generated as a numeric value


   public Long getDepartmentId() {
      return departmentId;
   }
   public Department(Long departmentId, String
departmentName, String departmentAddress, String
departmentCode) {
      this.departmentId = departmentId;
      this.departmentName = departmentName;
      this.departmentAddress = departmentAddress;
      this.departmentCode = departmentCode;
   }
   public Department() {
   }
   @Override
   public String toString() {
      return "Department{" +
            "departmentId=" + departmentId +
            ", departmentName='" + departmentName + '\'' +
            ", departmentAddress='" + departmentAddress +
'\'' +
            ", departmentCode='" + departmentCode + '\'' +
            '}';
```

```
    }
    public void setDepartmentId(Long departmentId) {
        this.departmentId = departmentId;
    }
    public String getDepartmentName() {
        return departmentName;
    }
    public void setDepartmentName(String departmentName) {
        this.departmentName = departmentName;
    }
    public String getDepartmentAddress() {
        return departmentAddress;
    }
    public void setDepartmentAddress(String
departmentAddress) {
        this.departmentAddress = departmentAddress;
    }
    public String getDepartmentCode() {
        return departmentCode;
    }
    public void setDepartmentCode(String departmentCode) {
        this.departmentCode = departmentCode;
    }
}
```

Not defficult, command + N does the trick

Service

In the service create a standard approach, an interface and its implementation

create DepartmentService as an implementation as this

```
package com.hydottech.Springboot.Tutorial.service;
public interface DepartmentService {
}
```

Create DepartmentServiceImpl to implement the service as this

```
package com.hydottech.Springboot.Tutorial.service;
import org.springframework.stereotype.Service;
@Service
public class DepartmentServiceImpl implements
DepartmentService{
}
```

Repository

In the Repository folder, create a DepartmentRepository as an interface that extends all the functionalities in `JpaRepository`

```
package com.hydottech.Springboot.Tutorial.repository;
import com.hydottech.Springboot.Tutorial.entity.Department;
import
org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
@Repository
public interface DepartmentRepository extends
JpaRepository<Department,Long> {
}
```

Department is the model or entity
Long is the type of it primary key

CRUD Operation

Spring Boot Save Data

This is the application Flow

FrontEnd

Using FormData

departmentName Kitchen

departmentAddress Accra

departmentCode KT125

Entity

Department.java

```
package com.hydottech.Springboot.Tutorial.entity;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
@Entity
public class Department {
   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   private Long departmentId;
   private String departmentName;
   private String departmentAddress;
   private String departmentCode;
   //To generate the getters and setters, press command +
n, select getters and setters, highlight all the field we
will generate getters and setters for, press okay
//Use the same method for their constructors
//departmentId is a primary key and it will be auto
generated as a numeric value
   public Long getDepartmentId() {
      return departmentId;
   }
   public Department(Long departmentId, String
departmentName, String departmentAddress, String
departmentCode) {
      this.departmentId = departmentId;
      this.departmentName = departmentName;
      this.departmentAddress = departmentAddress;
      this.departmentCode = departmentCode;
   }
   public Department() {
   }
   @Override
   public String toString() {
      return "Department{" +
            "departmentId=" + departmentId +
            ", departmentName='" + departmentName + '\'' +
            ", departmentAddress='" + departmentAddress +
'\'' +
            ", departmentCode='" + departmentCode + '\'' +
            '}';
   }
   public void setDepartmentId(Long departmentId) {
      this.departmentId = departmentId;
```

```
    }
    public String getDepartmentName() {
        return departmentName;
    }
    public void setDepartmentName(String departmentName) {
        this.departmentName = departmentName;
    }
    public String getDepartmentAddress() {
        return departmentAddress;
    }
    public void setDepartmentAddress(String
departmentAddress) {
        this.departmentAddress = departmentAddress;
    }
    public String getDepartmentCode() {
        return departmentCode;
    }
    public void setDepartmentCode(String departmentCode) {
        this.departmentCode = departmentCode;
    }
}
```

Controller

DepartmentController.java

```
package com.hydottech.Springboot.Tutorial.controller;
import com.hydottech.Springboot.Tutorial.entity.Department;
import
com.hydottech.Springboot.Tutorial.service.DepartmentService
;
import
com.hydottech.Springboot.Tutorial.service.DepartmentService
Impl;
import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;
@RestController
public class DepartmentController {
    @Autowired
    private DepartmentService departmentService;
    @PostMapping("/departments")
    public Department saveDepartment(@ModelAttribute
Department department) {
        return departmentService.saveDepartment(department);
    }
/*
* Explanation
```

```
*
*  @Autowired
   private DepartmentService departmentService;
   This will bind the service with a global service stored
by spring
   *
  @PostMapping("/departments")
   public Department saveDepartment(@RequestBody Department
department){
      return departmentService.saveDepartment(department);
   };
1.  @PostMapping("/departments") this is a post method
2. public Department saveDepartment(@RequestBody Department
department),
 a public with the Entity type Department with a name,
saveDepartment and the JSON body of type Department
3.  return departmentService.saveDepartment(department);
inside the DepartmentService interface we are calling the
saveDepartment method in it,
When we highlight we can auto create the methods in the
interface as this
 public Department saveDepartment(Department department);
 This will generate an error in the DepartmentServiceImpl,
implementing the methods can auto generate like this
  @Override
   public Department saveDepartment(Department department)
{
      return null;
   }
*
*
*
*
*
* */
}
```

From the Controller the service method will be auto created

Service

DepartmentService.java as the interface

```
package com.hydottech.Springboot.Tutorial.service;
import com.hydottech.Springboot.Tutorial.entity.Department;
public interface DepartmentService {
  public Department saveDepartment(Department department);
}
```

From the DepartmentService.java the DepartmentServiceImpl.java method will be auto created

DepartmentServiceImpl.java as the implementation

```
package com.hydottech.Springboot.Tutorial.service;
import com.hydottech.Springboot.Tutorial.entity.Department;
import
com.hydottech.Springboot.Tutorial.repository.DepartmentRepo
sitory;
import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
@Service
public class DepartmentServiceImpl implements
DepartmentService{
    @Autowired
    private DepartmentRepository departmentRepository;
    @Override
    public Department saveDepartment(Department department)
{
        return departmentRepository.save(department);
    }
}
```

Repository

```
package com.hydottech.Springboot.Tutorial.repository;
import com.hydottech.Springboot.Tutorial.entity.Department;
import
org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
@Repository
public interface DepartmentRepository extends
JpaRepository<Department,Long> {
}
```

Spring Boot Get Data

Same Process, We will go from the controller to serviceInterface to serviceImplementation where the repository will be called

Controller

```
@GetMapping("/getDepartments")
public List<Department> fetchDepartments(){
   return departmentService.fetchDepartments();
}
```

Service Interface will be auto created as

```
public List<Department> fetchDepartments();
```

Service Implementation with data repository

```
@Override
public List<Department> fetchDepartments() {
   return departmentRepository.findAll();
}
```

Spring Boot Get Specific Data

From the Entity Model we defined the departmentId as the primaryKey, so it means that any findById will match with the departmentId.

Let start the process

Controller

```
@GetMapping("/getSpecificDepartment/{Id}")
public Department fetchOneDepartment(@PathVariable("Id")
long Id){
   return departmentService.fetchOneDepartments(Id);
}
```

Service Interface

```
public Department fetchOneDepartments(long id);
```

Service Implementation

```
@Override
public Department fetchOneDepartments(long id) {
   return departmentRepository.findById(id).get();
}
```

Spring Boot Delete Specific Data

Controller

```
@DeleteMapping("/departments/{Id}")
```

```
public String deleteOneDepartment(@PathVariable("Id") long
Id){
   departmentService.deleteOneDepartments(Id);
   return "Department deleted successfully";
}
```

Service Interface

```
public void deleteOneDepartments(long id);
```

Service Implementation

```
@Override
public void deleteOneDepartments(long id) {
   departmentRepository.deleteById(id);
}
```

Spring Boot Update Specific Data

For Put, Instead of manually Updating it one by one, there is a shortcut

Controller

```
@PutMapping("/departments/{Id}")
public Department updateDepartment(@PathVariable("Id") long
Id, @ModelAttribute Department department){
   return departmentService.updateDepartment(Id,
department);
}
```

Service Interface

```
public Department updateDepartment(long id, Department
department);
```

Service Implementation

```
@Override
public Department updateDepartment(long id, Department
department) {
   Department db =
departmentRepository.findById(id).orElseThrow(() -> new
RuntimeException("Department not found"));
   // Get all the fields of the Department class
   Field[] fields = Department.class.getDeclaredFields();
   for (Field field : fields) {
      field.setAccessible(true);
      try {
```

```
            // Get the value of the current field from the
input department
            Object value = field.get(department);
            // Check if the value is not null and not an empty
string
            if (Objects.nonNull(value)
&& !"".equals(value.toString().trim())) {
               // Set the value of the current field in the db
department
               field.set(db, value);
            }
         } catch (IllegalAccessException e) {
            e.printStackTrace(); // Handle exception as
appropriate
         }
      }
      return departmentRepository.save(db);
}
```

Generic Api's

To Get the specific name of the department, the data repository does the trick

Let start with the **controller**

```
@GetMapping("/departmentsName/{Name}")
public Department
fetchDepartmentByName(@PathVariable("Name") String Name){
   return departmentService.fetchDepartmentByName(Name);
}
```

Service Interface

```
public Department fetchDepartmentByName(String name);
```

Service Implementation

```
@Override
public Department fetchDepartmentByName(String name) {
   return departmentRepository.findByDepartmentName(name);
}
```

Repository

The field name is `departmentName` so in the datarepository, type findBy+the fieldName in camel case. That all

```
public Department findByDepartmentName(String
departmentName);
```

```
we can even execute sql queries directly
@Query(value = "SELECT * FROM DEPARTMENT WHERE
DEPARTMENT_NAME ILIKE :departmentName", nativeQuery = true)
public Department
findByDepartmentNameIgnoreCaseKofi(@Param("departmentName")
String departmentName);
```

External Links

For More Information Visit the Following url

Docs: https://docs.spring.io/spring-data/jpa/reference/#jpa.query-methods

Spring Data JPA: https://www.youtube.com/watch?v=XszpXoII9Sg&t=0s

Spring Security:https://www.youtube.com/watch?v=tWcqSIQr6Ks&list=PLhfxuQVMs-nzbKxB2Zb7F9kjXntJhcP5k&index=7

Microservices: https://www.youtube.com/watch?v=BnknNTN8icw&t=0s

Validation