Spring Framework

… EN DESARROLLO …

Overview

Spring Framework es un poderoso y ampliamente utilizado marco de desarrollo de software para aplicaciones empresariales en Java. Diseñado para simplificar y acelerar el desarrollo de aplicaciones, Spring ofrece un enfoque integral que abarca desde la configuración hasta la implementación, abordando varios aspectos del desarrollo de software como la inversión de control, la inyección de dependencias, la gestión de transacciones, la seguridad y mucho más.

Una de las características distintivas de Spring es su **enfoque modular y extensible**, permitiendo a los desarrolladores elegir los módulos específicos que necesitan para sus proyectos. Además, fomenta las mejores prácticas de programación y sigue el principio de diseño de "Programación Orientada a Aspectos" (AOP), que facilita la separación de preocupaciones y mejora la modularidad del código.

Spring Framework se utiliza comúnmente para construir aplicaciones empresariales robustas y escalables, facilitando la creación de servicios web, aplicaciones basadas en la arquitectura Modelo-Vista-Controlador (MVC), integración con bases de datos, gestión de transacciones y mucho más. Con una comunidad activa y un ecosistema de proyectos relacionados, Spring ha evolucionado para adaptarse a las cambiantes demandas del desarrollo de software, convirtiéndose en una opción popular entre los desarrolladores Java.

Spring Boot es una extensión del popular Spring Framework que se centra en simplificar drásticamente el proceso de desarrollo de aplicaciones Java, especialmente aplicaciones basadas en Spring. Su objetivo principal es facilitar la creación de aplicaciones autónomas, autocontenidas y listas para la producción con la menor cantidad de configuración posible.

La relación entre Spring Boot y Spring Framework es fundamental, ya que Spring Boot se construye sobre la base sólida proporcionada por Spring. Spring Boot utiliza las características clave de Spring, como la inversión de control (IoC) y la inyección de dependencias, pero agrega una capa de convenciones y configuraciones por defecto para acelerar el desarrollo.

Lo más notable de Spring Boot es su enfoque de "opinión sobre la configuración", lo que significa que proporciona configuraciones predeterminadas sensatas para la mayoría de los casos de uso, permitiendo a los desarrolladores empezar rápidamente con sus proyectos sin tener que configurar extensamente. No obstante, sigue siendo altamente personalizable, permitiendo a los desarrolladores anular las configuraciones por defecto según sea necesario.

Con Spring Boot, el proceso de desarrollo se simplifica mediante la inclusión de un servidor embebido, como Tomcat o Jetty, lo que elimina la necesidad de desplegar la aplicación en un servidor externo. También facilita la gestión de dependencias mediante el uso de la herramienta Spring Initializr para generar proyectos con las dependencias necesarias.

En resumen, Spring Boot es una extensión de Spring Framework diseñada para hacer que el desarrollo de aplicaciones Java sea más rápido, sencillo y eficiente al proporcionar configuraciones por defecto y convenciones inteligentes sin sacrificar la flexibilidad y la potencia que ofrece Spring Framework.

> Introducción generada por ChatGPT

Spring Boot Cheat Sheet Annotations

@Repository

- Class Level Annotation
- It can reach the database and do all the operations.
- It make the connection between the database and the business logic.
- DAO is a repository.
- It is a marker interface.

```
@Repository
public class TestRepo{
   public void add(){
      System.out.println("Added");
   }
}
```

@Service

- Class Level Annotation
- It is a marker interface.
- It is a business logic.
- It is a service layer.
- It used to create a service layer.

```
@Service
public class TestService{
   public void service1(){
      //business code (iş kodları)
   }
}
```

@Autowired

- Field Level Annotation
- It is used to inject the dependency.
- It is used to inject the object.
- It is used to inject the object reference.
- Dependency Injection is a design pattern.

```
public class Brand{
   private int id;
   private String name;

   @Autowired
   public Brand(int id, String name){
     this.id = id;
     this.name = name;
```

```
    }
}
```

@Controller

- Class Level Annotation
- It is a marker interface.
- It is a controller layer.
- It is used to create a controller layer.
- It use with @RequestMapping annotation.

```
@Controller
@RequestMapping("/api/brands")
public class BrandsController{
   @GetMapping("/getall")
   public Employee getAll(){
       return brandService.getAll();
   }
}
```

@RequestMapping

- Method Level Annotation
- It is used to map the HTTP request with specific method.

```
@Controller
@RequestMapping("/api/brands")
public class BrandsController{
   @GetMapping("/getall")
   public Employee getAll(){
       return brandService.getAll();
   }
}
```

@GetMapping

- Method Level Annotation
- It is used to map the HTTP GET request with specific method.
- It is used to get the data.
- It is used to read the data.

```
@GetMapping("/getall")
public Employee getAll(){
  return brandService.getAll();
}
```

@PostMapping

- Method Level Annotation
- It is used to map the HTTP POST request with specific method.
- It is used to add the data.
- It is used to create the data.

```
@PostMapping("/add")
public void add(@RequestBody Brand brand){
  brandService.add(brand);
}
```

@PutMapping

- Method Level Annotation
- It is used to map the HTTP PUT request with specific method.
- It is used to update the data.

```
@PutMapping("/update")
public void update(@RequestBody Brand brand){
  brandService.update(brand);
}
```

@DeleteMapping

- Method Level Annotation
- It is used to map the HTTP DELETE request with specific method.
- It is used to delete the data.

```
@DeleteMapping("/delete")
public void delete(@RequestBody Brand brand){
  brandService.delete(brand);
}
```

@PathVariable

- Method Level Annotation
- It is used to get the data from the URL.
- It is the most suitable for RESTful web service that contains a path variable.

```
@GetMapping("/getbyid/{id}")
public Brand getById(@PathVariable int id){
  return brandService.getById(id);
}
```

@RequestBody

- It is used to get the data from the request body.
- It is used to get the data from the HTTP request.
- It is used to get the data from the HTTP request body.

```
@PostMapping("/add")
public void add(@RequestBody Brand brand){
  brandService.add(brand);
}
```

@RequestParam

- It is used to get the data from the URL.
- It is used to get the data from the URL query parameters.
- It is also known as query parameter.

```
@GetMapping("/getbyid")
public Brand getById(@RequestParam int id){
    return brandService.getById(id);
}
```

@RestController

- Class Level Annotation
- It is a marker interface.
- It is a controller layer.
- It is used to create a controller layer.
- It use with @RequestMapping annotation.
- It is a combination of @Controller and @ResponseBody annotations.
- @RestController annotation is explained with @ResponseBody annotation.
- @ResponseBody eliminates the need to add a comment to every method.

```
@RestController
@RequestMapping("/api/brands")
public class BrandsController{
   @GetMapping("/getall")
   public Employee getAll(){
       return brandService.getAll();
   }
}
```

Enlaces de interés

- https://spring.io/
- https://dev.to/burakboduroglu/spring-boot-cheat-sheet-460c
- https://www.geeksforgeeks.org/spring-boot/

Licencia