code URJC

Curso de Spring Framework

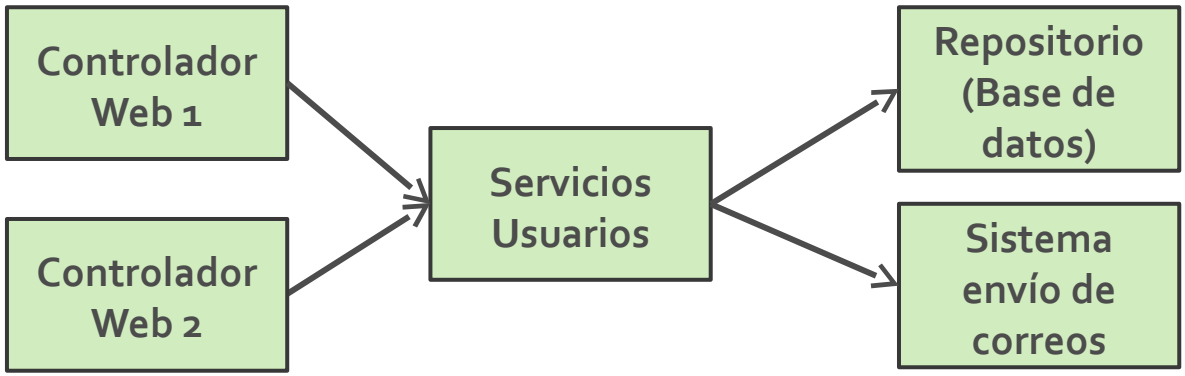
Inyección de dependencias

Micael Gallego
micael.gallego@gmail.com
@micael_gallego

Francisco Gortázar
patxi.gortazar@gmail.com
@fgortazar

Inyección de dependencias

- Las aplicaciones se suelen dividir en **módulos de alto nivel**
- Algunos **módulos** ofrecen servicios a otros módulos
- **Ejemplo:** Diseño modular de una aplicación web con SpringMVC

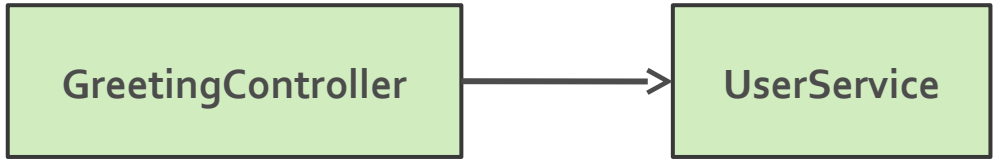


Inyección de dependencias

- ¿Cómo se **implementa un módulo**?
- ¿Cómo se **conecta un módulo** a otro módulo?
- La **inyección de dependencias** es una técnica que permite especificar un módulo y sus dependencias
- Cuando se inicia la aplicación, el framework crea todos los módulos e **inyecta las dependencias** en los módulos que las necesitan
- **Spring** dispone de un sistema de inyección de dependencias interno

Inyección de dependencias

ejem2




```
@Controller
public class GreetingController {

  @Autowired
  private UsersService usersService;

  @RequestMapping("/greeting")
  public String greeting(Model model) {

    model.addAttribute("name",
       usersService.getNumUsers()+" users");

    return "greeting_template";
  }
}
```

```
@Component
public class UsersService {

  public int getNumUsers(){
    return 5;
  }
}
```

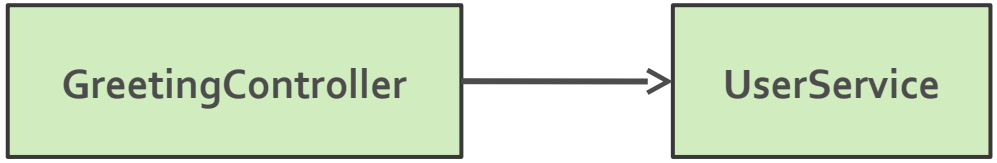
Inyección de dependencias

- A los módulos de la aplicación Spring se los denomina **beans** o **componentes**
- Para que una clase se considere un componente, tiene que anotarse con **@Component @Controller o @Service**
- Si un componente depende otro, puede poner la anotación **@Autowired** (auto enlazado) en un atributo, un constructor o un método setter

http://docs.spring.io/spring/docs/4.1.0.RELEASE/spring-framework-reference/htmlsingle/#beans

Inyección de dependencias

ejem2




```
@Controller
public class GreetingController {

  @Autowired
  private UsersService usersService;

  @RequestMapping("/greeting")
  public String greeting(Model model) {

    model.addAttribute("name",
       usersService.getNumUsers()+" users");

    return "greeting_template";
  }
}
```

```
@Component
public class UsersService {

  public int getNumUsers(){
    return 5;
  }
}
```

Inyección de dependencias

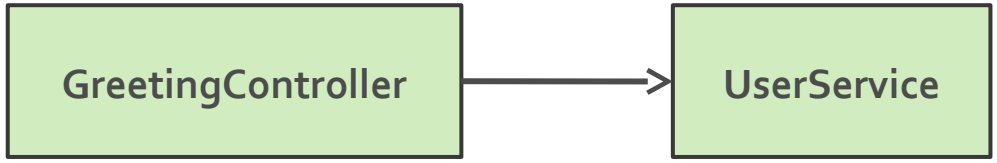
- Existen casos en los que los componentes de una aplicación vienen en librerías (no los podemos modificar) pero tienen que ser **configurados en la aplicación**
- Existen otros casos en los que existen **varias implementaciones** disponibles de un componente y la aplicación tiene que **seleccionar** la implementación concreta
- En estos casos, la aplicación puede **configurar los componentes** de la aplicación

Inyección de dependencias

- En las aplicaciones **SpringBoot**, la clase principal de la aplicación se utiliza para **configurar los componentes**
- Por cada componente que se quiera **configurar**:
  - Se **quita la anotación** @Component del componente
  - Se **añade un método** en la clase principal que devuelva un nuevo componente configurado

Inyección de dependencias

ejem3




```java
@Controller
public class GreetingController {

  @Autowired
  private UsersService usersService;

  @RequestMapping("/greeting")
  public String greeting(Model model) {

    model.addAttribute("name",
       usersService.getNumUsers()+" users");

    return "greeting_template";
  }
}
```

```java
public class UserService {

  private int numUsers;

  public UserService(int numUsers){
    this.numUsers = numUsers;
  }

  public int getNumUsers() {
    return numUsers;
  }
}
```

Inyección de dependencias

ejem3

En la clase de la aplicación se configura el componente

Se implementa un método anotado con **@Bean** que devuelve el componente ya configurado

```
@SpringBootApplication
public class Application {

    @Bean
    public UserService usersService(){
        return new UserService(10);
    }

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```