

Spring-Boot Lezione 2

Alessandro Vizzarro
alessandro.vizzarro@aesys.tech

2022 / 2023



Argomenti

- CONFIGURATION PROPERTIES
- PROFILES
- REST SERVICES
- ERRORS HANDLING
- APPLICATION CONTEXT
- INTERNATIONALIZATION
- VALIDATION FEATURE
- CLIENT REST
- ESERCITAZIONE

SPRING BOOT

External Configuration

- > E' possibile configurare l'applicazione rispettivamente ad un determinato environment.
- > Spring offre diverse sorgenti o file per configurare l'applicazione: properties file, yaml file, variabili d'ambiente e parametri passati da command line.
- > Puoi utilizzare le configurazioni (tipicamente chiave - valore) associando ad una variabile di classe tramite l'annotation @value, oppure utilizzando l'annotation @ConfigurationProperties per effettuare un bind di intere strutture ad oggetti a configurazioni partendo da una determinata chiave prefissa.
- > Le configurazioni seguono ordine prefissato, che consente l'override di valori.
- > E' possibile definire sorgenti esterne su file system che non siano contenute nel Classpath dell'applicazione.

```
import java.util.Arrays;

@RestController
@RequestMapping(path = "/")
@SpringBootApplication
@ConfigurationPropertiesScan({ "it.aesys.courses.springboot.lesson2.config" })
public class Lesson2Application {

    private static final Logger logger = LoggerFactory.getLogger(Lesson2Application.class);

    @Value("${app.lesson}")
    private String lessonNumber;

    @Value("${lesson.welcome-message}")
    private String lessonMessage;

    @Value("${lesson.author}")
    private String lessonAuthor;

    @Autowired
    private MyServiceImpl service;

    public static void main(String[] args) { SpringApplication.run(Lesson2Application.class, args); }

    @RequestMapping(method = RequestMethod.GET)
    public String lesson() {
```

```
return "${lesson}";
}

@Override
public void setMessage(String message) {
    this.lessonMessage = message;
}

@Override
public void setAuthor(String author) {
    this.lessonAuthor = author;
}

@Override
public void setLessonNumber(String lessonNumber) {
    this.lessonNumber = lessonNumber;
}
```

Spring Boot

Ordine Sorgenti proprietà



Definiamo di seguito alcune delle features di configurazione che vengono considerate nell'ordine seguente, sono state prelevate direttamente dalla documentazione (Ho mantenuto le features di configurazione più utilizzate riferirsi alla documentazione per completezza):

- 1 - Default properties (specified by setting `SpringApplication.setDefaultProperties`).
- 2 - `@PropertySource` annotations on your `@Configuration` classes. Please note that such property sources are not added to the Environment until the application context is being refreshed. This is too late to configure certain properties such as `logging.*` and `spring.main.*` which are read before refresh begins.
- 3 - Config data (such as `application.properties` files).
- 4 - OS environment variables.
- 5 - Java System properties (`System.getProperties()`).
- 6 - Command line arguments.

Config Data a sua volta legge i file di properties di default sulla root o sulla cartella "config" che sono presenti sul classpath, inoltre è possibile definire una proprietà che specifica la location del file sul file system e sovrascrivere le coppie chiave valore del file principale.

Altresì il file di properties può essere ulteriormente sovrascritto con il meccanismo dei Profile che vedremo più avanti.

Riferimenti su proprietà gestite direttamente dai Spring-boot e i moduli inclusi tramite dependencies:

<https://docs.spring.io/spring-boot/docs/current/reference/html/application-properties.html#appendix.application-properties.core>



Spring-boot Profiles

- > I profili consentono ad una applicazione spring boot di compartimentare componenti, file di configurazione definite da `@Component` (o sue estensioni), `@Configuration` e `@ConfigurationProperties` rispetto ad uno specifico Environment
- > Il profilo è attivabile inserendo una property: `spring.profiles.active = production` nel file `application.properties` o `application.yml`
- > In Alternativa è possibile passarla tramite linea di comando `–spring.profiles.active = production`
- > Può contenere anche diversi profili ex. `–spring.profiles.active = production-db, production-ws,...`
- > E' possibile inoltre impostare gruppi di profili per suddividere e delegare ad uno specifico profilo la responsabilità di alcune configurazioni

Ex. `spring.profiles.active = production`
`spring.profiles.group.production[0] = prod-db, prod-ws, prod-task,...`

Spring Boot

Profiles – file properties

- > Come già detto un profilo consente di selezionare oltre ad un file di default application.yml anche un file specifico di environment rispetto al profilo attivato.
- > I file specifici attivati dai profili effettuano override delle proprietà definite da entrambi i file

```
1 spring:
2   profiles:
3     active: devel
4
5 app:
6   lesson: LESSON 1
7
8 lesson:
9   author: alessandro.vizzarro@aesys.tech
10  welcome-message: Welcome to ${app.lesson} !!!
11
12 my:
```

Spring Boot

Profiles – file properties

- > Come già detto un profilo consente di selezionare oltre ad un file di default application.yml anche un file specifico di environment rispetto al profilo attivato.
- > I file specifici attivati dai profili effettuano override delle proprietà definite da entrambi i file

```
1 spring:
2   profiles:
3     active: devel
4
5 app:
6   lesson: LESSON 1
7
8 lesson:
9   author: alessandro.vizzarro@aesys.tech
10  welcome-message: Welcome to ${app.lesson} !!!
11
12 my:
```



SPRING BOOT REST API

- > Spring boot utilizza diverse implementazioni per i REST: Spring-MVC, JAX-RS, ci focalizzeremo su Spring-MVC
- > L'annotazione Principale è @RestController una estensione di @Controller utilizzata largamente su applicazioni web Spring framework, rende meno onerosa la configurazione di un endpoint Rest.
- > Esistono diverse Annotazioni che semplificano il mapping di metodi per le CRUD Operations: @GetMapping, @PostMapping, @DeleteMapping.
- > Spring Boot implementa diversi meccanismi per gestire le situazioni di errore e i codici di errore classici di una applicazione Rest con protocollo HTTP, 500, 404, 403, 502, 402...



Spring Boot

Rest api example

```
import java.util.List;

@RestController
@RequestMapping("/api/hero")
public class HeroCreatorController {

    @Autowired
    private HeroServiceImpl heroService;

    @PostMapping
    public Hero createHero(@RequestBody Hero hero) {
        return heroService.insert(hero);
    }

    @PutMapping("/{id}")
    public Hero saveHero(@RequestBody Hero hero, @PathVariable String id) {
        return heroService.update(hero, id);
    }

    @GetMapping("/{id}")
    public Hero find(@PathVariable String id) {
        return heroService.find(id);
    }

    @GetMapping
    public List<Hero> findAll() {
        return heroService.findAll();
    }
}
```



Spring-boot

Errors handling

- > Spring boot offre diversi meccanismi di gestione dell'errore.
- > Di default a secondo della chiamata da Browser o altro mostra una schermata riassuntiva di errore. Nel caso del Browser Spring Boot MVC utilizza un Internal Resolver di default che mostra una white Page, se non è stato configurato un ViewResolver ad hoc. E' possibile disabilitare questo comportamento tramite una property predefinita.
- > E' possibile Implementare un Resolver oppure ,configurare uno starter: spring-boot-starter-thymeleaf un java template engine, che auto-configura dei Resolver ad hoc in grado di effettuare un dispatch dell'elaborazione verso una parte di vista tramite file html localizzati, nel folder template nelle risorse.
- > Inoltre Spring-boot offre un meccanismo per gestire gli errori in modo centralizzato attraverso le seguenti annotations: @ControllerAdvice, @ExceptionHandler e @ResponseStatus



SPRING BOOT

Application Context

- > Spring boot come spring ha un Contesto fruibile attraverso una Specifica classe che implementa l'interfaccia ApplicationContext.
- > Il metodo SprinApplicationBootsrap.Run nel main method , ritorna una implementazione di ApplicationContext.
- > E' possibile effettuare un injection di un ApplicationContext di Spring all'interno delle proprie componenti bean.



Tempo di esercitazione



Implementare la versione Spring-boot degli esercizi proposti durante il corso Spring framework

Implementiamo esempio dei colori e l'esempio dell'orchestra o di un nuovo esercizio proposto durante corso Springframework (Compito per casa).

Ragionare un dominio di interesse che sia «semplice», proviamo a formalizzarne le specifiche UML ed implementiamo insieme il o i Microservizi necessari. Utilizzando i meccanismi che abbiamo finora imparato. Potete inviarmi le vostre esercitazioni personali per email e le supervisionerò per poi darvi dei feedback entro fine corso.

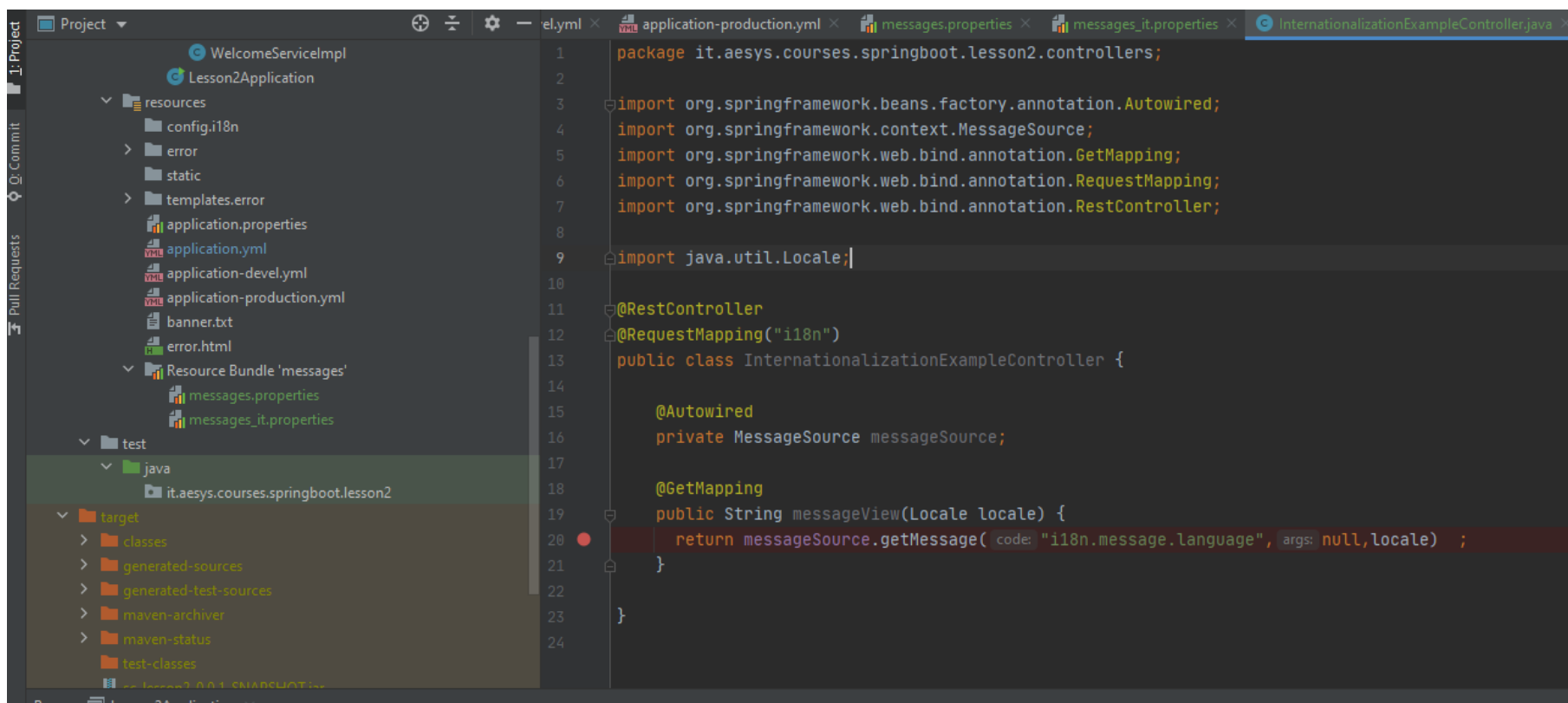


Spring-boot Internationalization II18N

- > Spring boot supporta meccanismi di internazionalizzazione dei messaggi applicativi
- > Come ogni modulo di spring anche L'internazionalizzazione è autoconfigurata e di default Spring effettua due operazioni:
 1. Verifica presenza la non presenza di un bean MessageSource
 2. legge un file testuale che contiene i messaggi applicativi nella solita forma chiave valore .
- > Se le condizioni sopra indicate sono rispettate , il modulo istanzia un bean MessageSource nel contesto utilizzabile per consultare il bundle dei messaggi.
- > Egualmente al meccanismo delle configuration properties è possibile avere più file messages per lingua
- > Le impostazioni possono essere indicate a spring-boot tramite delle properties `spring.messages.*`

Spring Boot

Internazionalizzazione - esempio



```
1 package it.aesys.courses.springboot.lesson2.controllers;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.context.MessageSource;
5 import org.springframework.web.bind.annotation.GetMapping;
6 import org.springframework.web.bind.annotation.RequestMapping;
7 import org.springframework.web.bind.annotation.RestController;
8
9 import java.util.Locale;
10
11 @RestController
12 @RequestMapping("i18n")
13 public class InternationalizationExampleController {
14
15     @Autowired
16     private MessageSource messageSource;
17
18     @GetMapping
19     public String messageView(Locale locale) {
20         return messageSource.getMessage( code: "i18n.message.language", args: null, locale) ;
21     }
22
23 }
24
```



SPRING BOOT

Validation module

- > Spring boot come spring fornisce un meccanismo di validatione dell'input per un dato Rest Endpoint.
- > Il modulo è autoconfigurabile a patto di includere la dipendenza spring-boot-starter-validation
- > E' possibile implementare la validazione annotando i nostri pojo di input con diverse tipologie di annotazioni : @NotNull, @NotBlank, @Min, @Max, @Pattern....secondo lo standard de Facto che segue le specifiche **JSR 380**
- > Per attivare la validazione di un bean è necessario annotare il parametro di input con l'annotazione @Valid.
- > La violazione di una o più regole di Validazione genererà una eccezione del tipo MethodArgumentNotValidException, di cui è possibile effettuare la gestione con le metodologie già affrontate.





SPRING BOOT

Rest Client

- > Un Rest client o più in generale un client di servizi web, può essere implementato con diverse tecnologie java e non, dipendentemente dai differenti contesti.
- > Una applicazione scritta in linguaggi tipici di front -end Angular, React, Javascript in generale, può essere implementato con librerie specifiche legate al tipo di linguaggio.
- > Librerie conosciute in ambito java: Retrofit(Molto usato Android Mobile), HTTP Client introdotto con java 11, RestTemplate incluso in Spring.....
- > Per i nostri fini proveremo a utilizzare RestTemplate e/o HttpClient
- > Esempio di RestTemplate:

```
RestTemplate restTemplate = new RestTemplate();
```

```
String fooResourceUrl = "http://localhost:8080/spring-rest/foo";
```

```
ResponseEntity<String> response = restTemplate.getForEntity(fooResourceUrl + "/1", String.class);
```

```
If (response.getStatusCode().equals(HttpStatus.OK)) return response.getBody();
```





Esercitazione

- > Implementare Microservizio Anagrafica Persone, che gestisce tutti i dati anagrafici di base di una persona fisica divisi per Aree: Anagrafici Base, Documenti validi, Black Book.
- > Disegnare L'UML tramite draw.io del modello dell'anagrafica.
- > Il microservizio deve esser profilato con i seguenti profili: Production, Devel; In Production profile cambiano le porte del servizio rispetto a quella di default 8080.
- > Introdurre i meccanismi di Validazione affrontati nel corso: tramite Validation Module Spring e/o programmatica con gestione delle BadRequest.
- > Gestire gli Errori e le eccezioni in maniera centralizzata tramite meccanismo di Error Handling affrontato nel corso.
- > Integrare Il Microservizio con il servizio Libreria Online già implementato tramite dei Client Rest, insieme capiremo dove e come integrare i vostri servizi con libreria Online.
- > L'esercizio verrà svolto in 3 Teams di 5 elementi, ogni elemento del team lavorerà su degli aspetti del Microservizio a vostra scelta ad esempio uno dei layer di cui abbiamo parlato. La complicità e collaborazione sono essenziali.



Esercitazione

- > Lo scopo dell'esercizio è far girare tutti i Microservizi insieme al progetto LibreriaOnline e verificare che tutto funzioni come da specifiche.
- > L'esercitazione potrà essere proseguita a casa.
- > Alla fine dell'esercitazione discuteremo con ogni team come è stato approcciato , realizzato il rispettivo progettino e le eventuali difficoltà incontrate prima dell'inizio dell'ultima parte di corso.



PESCARA
sede principale

via Conte di Ruvo 74 (PE)
65127
t. +39 085 812 3761
info[@aesystech.it](mailto:info@aesystech.it)

GRAZIE



www.aesystech.it