

Spring

Riccardo Cattaneo



Validated Annotation

Andiamo a vedere in questa lezione come andare a mettere dei controlli sui valori di input.

Prendiamo ad esempio nel nostro progetto il metodo `inserisciPersona`, che vuole in input un'oggetto di tipo `Persona`, potremmo voler non accettare campi null. Per come è impostato adesso il nostro progetto, se prova a fare una POST inserendo solo il nome, la mia applicazione funzionerebbe senza problemi...

POST

localhost:8080/inseriscipersona

Params

Authorization

Headers (9)

Body ●

Pre-request Script

Tests

Settings

☐ none

☐ form-data

☐ x-www-form-urlencoded

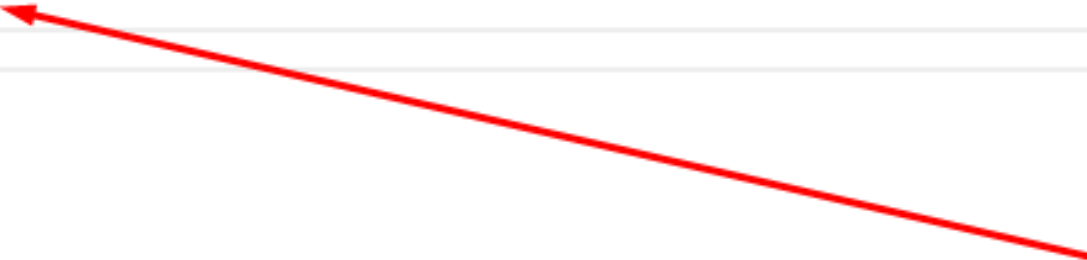
☒ raw

☐ binary

☐ GraphQL

JSON ▼

```
1 {  
2   "nome": "Lucia",  
3   "cognome": null  
4 }
```

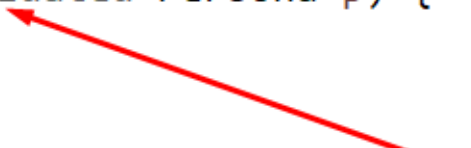


Possiamo quindi utilizzare l'annotazione **@validated** da inserire prima dell'oggetto di input. Questa annotazione sta dicendo a Spring che deve andare a validare questo oggetto.

Dove abbiamo costruito l'oggetto Persona possiamo adesso aggiungere dei controlli attraverso delle annotation. Se ad esempio voglio che il campo nome e cognome non siano null :

```
//@PostMapping("/inseriscipersona")
@RequestMapping(value = "/inseriscipersona", method = RequestMethod.POST)
public Persona inserisciPersona(@RequestBody @Validated Persona p) {

    return ser.inserisciPersonaService(p);
}
```



Mentre all'interno della classe Persona posso inserire l'annotazione per il controllo :

```
import javax.validation.constraints.NotNull;


public class Persona {

    @NotNull
    private String nome;

    @NotNull
    private String cognome;

    public Persona() {

    }
}
```



Assicuriamoci prima di provare il codice, di aver importato il modulo di Spring «Validation». Per farlo posizioniamoci all'interno del nostro IDE, tasto destro sul nostro progetto → Spring → Add Starters.

A questo punto si apre una finestra dove cerchiamo «Validation» e procediamo con l'importazione del modulo all'interno del nostro progetto :

Package Explorer

- demo [boot]
- demo-mvc [boot]
 - src/main/java
 - com.example.demo
 - Controller.java
 - DemoMvcApplication.java
 - Repository.java
 - Service.java
 - com.example.demo.entity
 - src/main/resources
 - src/test/java
 - JRE System Library [JavaSE-11]
 - Maven Dependencies
 - src
 - target
 - HELP.md
 - mvnw
 - mvnw.cmd
 - pom.xml
 - test-scope [boot]

Controller.java

```
1 package com.example.demo.entity;
2
3 import javax.validation.constraints.*;
4
5 public class Persona {
6
7     @NotNull
8     private String nome;
9
10    @NotNull
11    private String cognome;
12
13    public Persona() {
14    }
15
16
17    public String getNome() {
18        return nome;
19    }
20
21    public void setNome(String nome) {
22        this.nome = nome;
23    }
24
25    public String getCognome() {
26        return cognome;
27    }
28 }
```

Boot Dashboard

Type tags, projects, or working set names to m...

New Spring Starter Project Dependencies

Service URL:

Spring Boot Version:

Frequently Used:

☐ Spring Web ☒ Validation

Available:

Selected:

X Validation


I/O























☒ Validation

Make Default Clear Selection

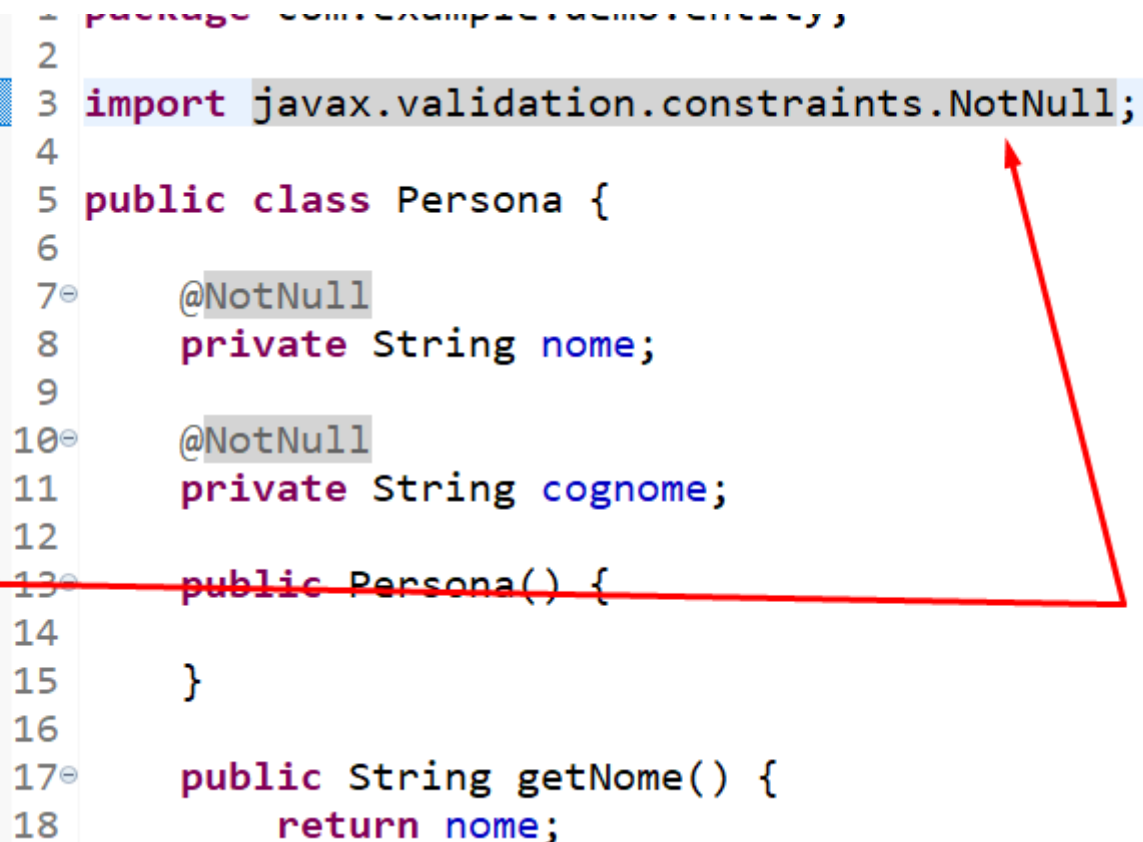
< Back Next > Finish Cancel

Per sapere quali e quante annotation ci sono per fare i controlli e sufficiente andare nel package appena importato quando abbiamo inserito l'annotation `NotNull` e vedere quelle disponibili :

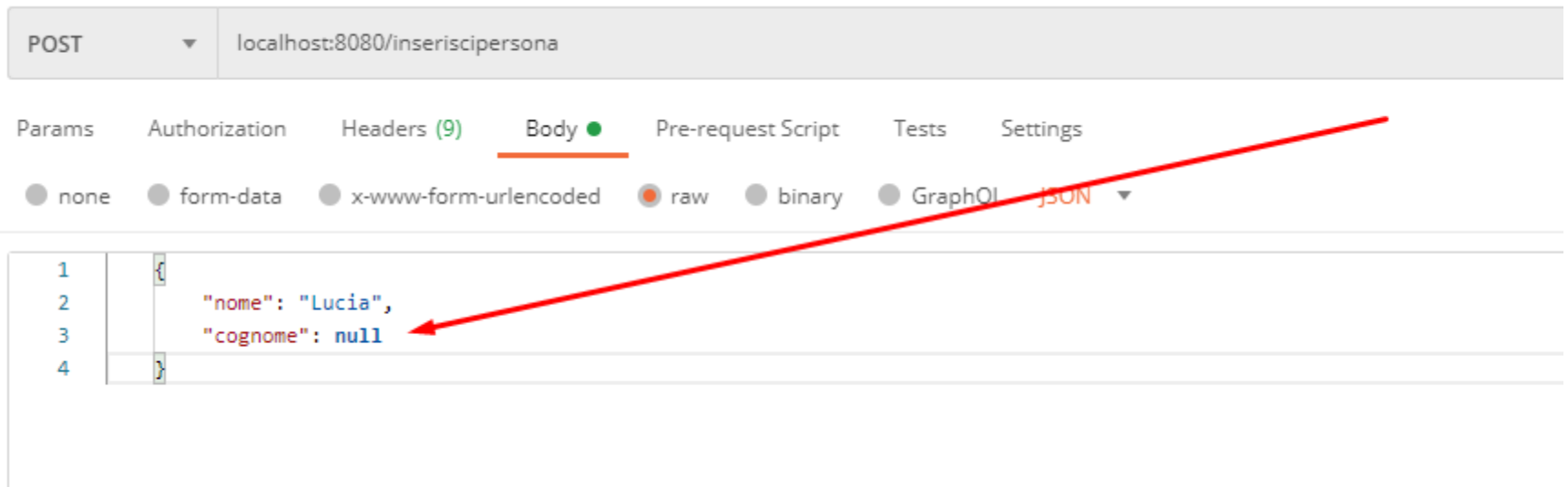
▼  javax.validation.constraints

- >  `AssertFalse.class`
- >  `AssertTrue.class`
- >  `DecimalMax.class`
- >  `DecimalMin.class`
- >  `Digits.class`
- >  `Email.class`
- >  `Future.class`
- >  `FutureOrPresent.class`
- >  `Max.class`
- >  `Min.class`
- >  `Negative.class`
- >  `NegativeOrZero.class`
- >  `NotBlank.class`
- >  `NotEmpty.class`
- >  `NotNull.class`
- >  `Null.class`
- >  `Past.class`
- >  `PastOrPresent.class`
- >  `Pattern.class`
- >  `Positive.class`
- >  `PositiveOrZero.class`
- >  `Size.class`

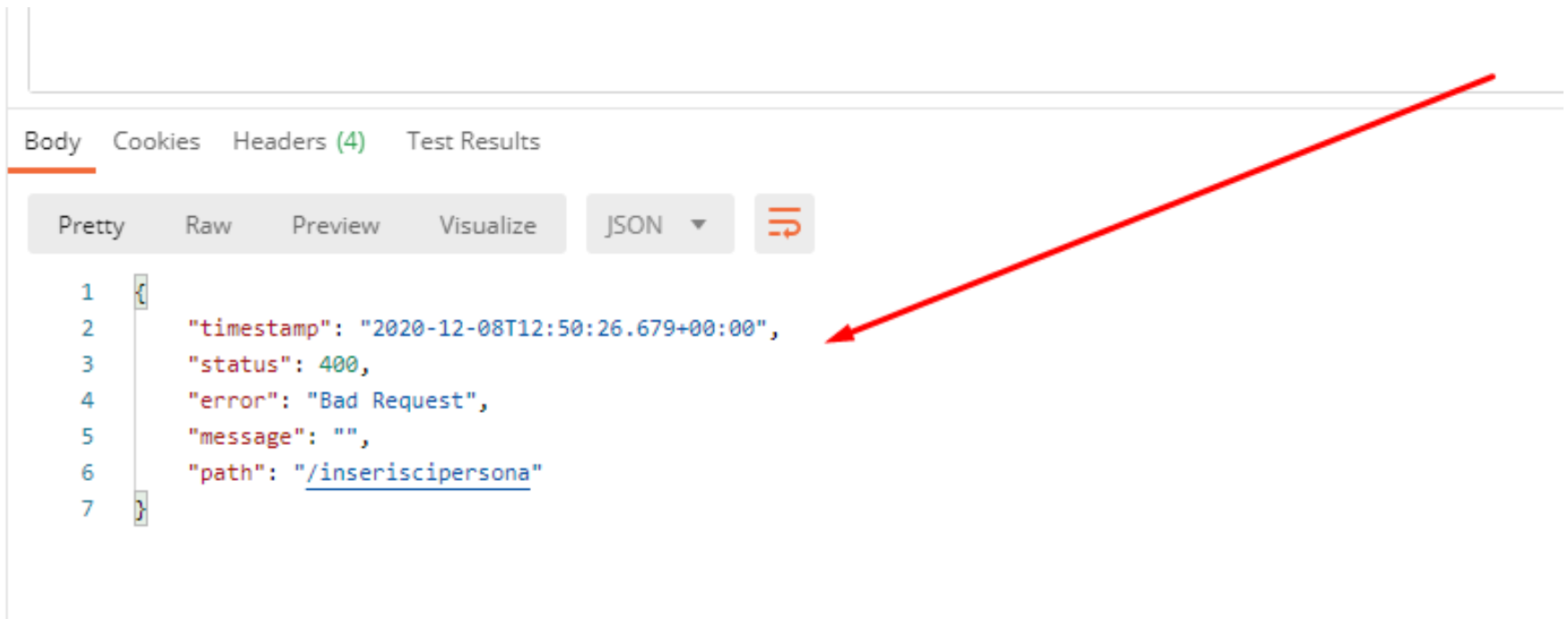
```
1 package com.example.demo.entities;  
2  
3 import javax.validation.constraints.NotNull;  
4  
5 public class Persona {  
6  
7     @NotNull  
8     private String nome;  
9  
10    @NotNull  
11    private String cognome;  
12  
13    public Persona() {  
14  
15    }  
16  
17    public String getNome() {  
18        return nome;  
19    }  
20 }
```



Finalmente possiamo provare la nostra validazione. Apriamo Postman e richiamiamo il servizio di inserimento Persona e mettere il cognome a null e inviamo la richiesta :



Come possiamo vedere ci torna un errore :




BindingResult

A questo punto abbiamo 2 possibilità, in base a quello che vogliamo fare : la prima possibilità è quella di lasciare andare in errore la chiamata, in quanto decidiamo di non gestire l'errore e lasciare che sia il chiamante ad effettuare la chiamata giusta.

La seconda possibilità è quella di aggiungere, subito dopo il parametro in ingresso al metodo, una variabile di tipo BindingResult in questo modo :

```
//@PostMapping("/inseriscipersona")
@RequestMapping(value = "/inseriscipersona", method = RequestMethod.POST)
public Persona inserisciPersona(@RequestBody @Validated Persona p, BindingResult result) {

    return ser.inserisciPersonaService(p);
}
```



Grazie a questo oggetto possiamo andare a gestire il risultato della chiamata, Spring in automatico quando vede tra i parametri di input un **BindingResult**, lo popola nel momento in cui prepara la risposta da inviare al chiamante, e noi possiamo così gestirla come meglio crediamo. Il vantaggio dell'utilizzo del BindingResult è che la nostra chiamata non andrà in errore ma andrà gestita... proviamo...

```
@PostMapping("/inseriscipersona")
public Persona inserisciPersona(@Validated @RequestBody Persona p, BindingResult result){

    Persona per = service.inserisciPersonaService(p);

    if(result.hasErrors()) {
        System.out.println("errore durante la fase di inserimento");
    }else{
        System.out.println("Tutto OK");
    }

    return per;
}
```

Launchpad

POST localhost:8080/inseriscipersona



Untitled Request

POST

localhost:8080/inseriscipersona

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON**

```
1 {  
2   "nome": "Lucia",  
3   "cognome": null  
4 }
```

Body Cookies Headers (5) Test Results

Pretty

Raw

Preview

Visualize

JSON



```
1 {  
2   "nome": "Lucia",  
3   "cognome": null  
4 }
```

Spring

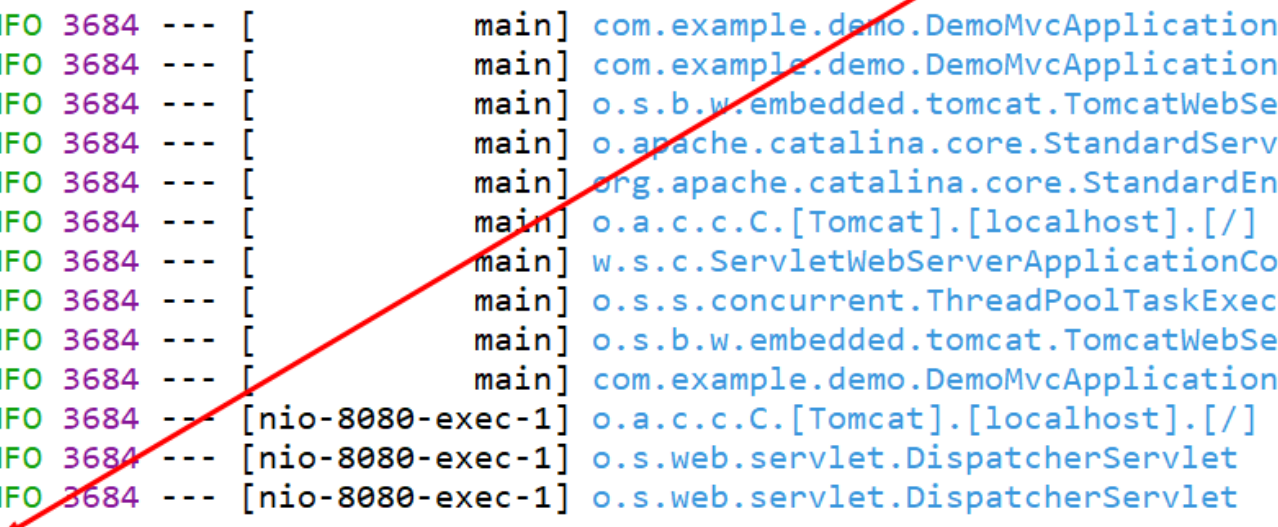
```

  ____
 /    \
(  )   \
 \    /
  ____
:: Spring Boot ::
                (v2.4.0)

```

```

2020-12-08 14:02:01.927 INFO 3684 --- [main] com.example.demo.DemoMvcApplication
2020-12-08 14:02:01.929 INFO 3684 --- [main] com.example.demo.DemoMvcApplication
2020-12-08 14:02:02.430 INFO 3684 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer
2020-12-08 14:02:02.437 INFO 3684 --- [main] o.apache.catalina.core.StandardService
2020-12-08 14:02:02.437 INFO 3684 --- [main] org.apache.catalina.core.StandardEngine
2020-12-08 14:02:02.485 INFO 3684 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/]
2020-12-08 14:02:02.485 INFO 3684 --- [main] w.s.c.ServletWebServerApplicationContext
2020-12-08 14:02:02.597 INFO 3684 --- [main] o.s.s.concurrent.ThreadPoolTaskExecutor
2020-12-08 14:02:02.715 INFO 3684 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer
2020-12-08 14:02:02.721 INFO 3684 --- [main] com.example.demo.DemoMvcApplication
2020-12-08 14:02:20.923 INFO 3684 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]
2020-12-08 14:02:20.923 INFO 3684 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet
2020-12-08 14:02:20.923 INFO 3684 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet
Errore durante la chiamata
```



A questo punto possiamo lanciare un'eccezione se non vogliamo che proceda nella insert oppure lasciare che la chiamata vada comunque a buon fine.

ATTENZIONE : Se inserisco tra i parametri il BindingResult, ma poi non lo gestisco all'interno del metodo, è come se non avessi messo il Validated, Spring non dà errori e la risposta torna al chiamante senza problemi.