REST API con Java Spring

Java Spring

Java Spring è un framework open source per lo sviluppo di applicazioni Java. È progettato per semplificare la programmazione Java e facilitare lo sviluppo di applicazioni Java enterprise.

Una volta iniziato un progetto con Java Spring possiamo cominciare a creare delle API REST.

Per creare un REST API con Java Spring abbiamo già visto come utilizzare **spring initializr** start.spring.io.

Aggiungere API

Per aggiungere delle API REST al nostro progetto, dobbiamo creare una classe **Controller**.

```
@RestController
public class HelloController {
    ...
}
```

L'annotazione `@RestController` indica che la classe è un controller REST. Questo significa che i metodi della classe saranno esposti come API REST. Spring gestirà le richieste HTTP e le risposte JSON.

`@RestController` non ha bisogno di parametri.

Una volta creato il controller, possiamo aggiungere dei metodi per gestire le richieste HTTP.

```
@GetMapping("/hello")
public String hello() {
    return "Hello, World!";
}
```

L'annotazione `@GetMapping` indica che il metodo gestisce le richieste HTTP GET. Il valore dell'annotazione indica il percorso (o route) dell'API.

Il metodo `hello()` restituisce una stringa "Hello, World!", e viene eseguito quando si accede all'API `/hello`. La funzione poteva chiamarsi come volevamo, ma il nome `hello` è stato scelto per chiarezza.

Il tipo di ritorno del metodo è `String`, ma potrebbe essere qualsiasi tipo di dato. Spring convertirà automaticamente il valore di ritorno in JSON.

Oltre a `@GetMapping`, ci sono altre annotazioni che possiamo usare per gestire le richieste HTTP:

- `@PostMapping` per le richieste POST
- `@PutMapping` per le richieste PUT
- `@DeleteMapping` per le richieste DELETE

Queste annotazioni possono essere usate per creare API RESTful complete.

> Ripasso metodi HTTP:
>
> - GET: leggere dati
> - POST: creare dati
> - PUT: aggiornare dati
> - DELETE: eliminare dati

> Bonus:
>
> - PATCH: aggiornare dati parzialmente
> - OPTIONS: ottenere informazioni sul server
> - HEAD: ottenere informazioni sull'header della risposta
> - TRACE: testare la connessione

> Ripasso header HTTP request:
>
> - Accept: tipo di contenuto accettato nella risposta
> - Authorization: informazioni di autenticazione
> - Content-Length: lunghezza del corpo della richiesta
> - Content-Type: tipo di contenuto della richiesta
> - Cookie: informazioni sulle sessioni
> - User-Agent: informazioni sul client

> Ripasso header HTTP response:
>
> - Content-Length: lunghezza del corpo della risposta
> - Content-Type: tipo di contenuto della risposta
> - Set-Cookie: impostare un cookie
> - Location: reindirizzare il client
> - WWW-Authenticate: informazioni di autenticazione

Parametri

Per passare parametri alle API REST, possiamo usare l'annotazione `@RequestParam`.

```
@GetMapping("/hello")
public String hello(@RequestParam(value = "name", defaultValue = "World") String name) {
    return "Hello, " + name + "!";
}
```

In questo caso, il parametro `name` è opzionale e ha un valore di default `World`. Se non viene specificato un valore per il parametro `name`, verrà utilizzato il valore di default.

Se non vogliamo un valore di default, possiamo rimuovere il parametro `defaultValue`. `@RequestParam(value = "name") String name`. oppure `@RequestParam String name`.

Per passare un valore al parametro `name`, possiamo aggiungere il parametro alla richiesta HTTP. Ad esempio, `/hello?name=John`.

Record

In Java 14 è stata introdotta una nuova feature chiamata **record**. I record sono classi immutabili che contengono solo dati.

```
public record Greeting(long id, String content) {}
```

In questo caso, `Greeting` è un record con due campi: `id` e `content`. I record sono utili per creare classi con pochi campi e senza metodi.

I record sono immutabili, il che significa che i campi non possono essere modificati dopo la creazione dell'oggetto.

Possiamo utilizzare i record come tipo di ritorno per i metodi delle API REST.

```
private int counter = 0;

@GetMapping("/hello")
public Greeting hello(@RequestParam(value = "name", defaultValue = "World") String name) {
    return new Greeting(counter++, "Hello, " + name + "!");
}
```

In questo caso, il metodo `hello()` restituisce un oggetto `Greeting` con un `id` incrementale e un messaggio di saluto personalizzato. Il record `Greeting` verrà automaticamente convertito in JSON dalla libreria Jackson di Spring.

Body

Finora abbiamo visto come passare parametri tramite URL. Ma cosa succede se vogliamo passare un oggetto complesso, come un record, tramite una richiesta POST?

Per passare un oggetto complesso, possiamo utilizzare l'annotazione `@RequestBody`.

```
@PostMapping("/greeting")
public Greeting greeting(@RequestBody Greeting greeting) {
    return greeting;
}
```

In questo caso, il metodo `greeting()` accetta un oggetto `Greeting` come parametro. L'annotazione `@RequestBody` indica che l'oggetto `Greeting` verrà deserializzato dal corpo della richiesta HTTP.

Per fare ciò, dobbiamo inviare una richiesta POST con un corpo JSON che rappresenta un oggetto `Greeting`.

Conclusioni

In questo breve tutorial abbiamo visto come creare una REST API con Java Spring. Abbiamo creato un controller REST, aggiunto metodi per gestire le richieste HTTP, passato parametri e restituito record come risposta.

Java Spring è un framework potente e flessibile per lo sviluppo di applicazioni Java. Con Java Spring, è possibile creare API RESTful in modo semplice e veloce.

Domande

- Che cos'è Java Spring?
- Come si crea un controller REST con Java Spring?
- Come si gestiscono le richieste HTTP con Java Spring?
- Come si passano parametri alle API REST con Java Spring?
- Come si restituiscono record come risposta con Java Spring?