(/)

YAML to List of Objects in Spring Boot

Last modified: July 12, 2020

by baeldung (https://www.baeldung.com/author/baeldung/)

Spring Boot (https://www.baeldung.com/category/spring/spring-boot/)

1. Overview

In this short tutorial, we're going to have a closer look at how to map a YAML list into a *List* in Spring Boot.

We'll first start with some background on how to define lists in YAML. Then, we'll dig deeper to see how to bind YAML lists to *List*s of objects.

2. Quick Recap About Lists in YAML

In short, YAML (https://yaml.org/spec/1.2/spec.html) is a human-readable data serialization standard that provides a concise and clear way to write configuration files. The good thing about YAML is the fact that it supports multiple data types such as *List*s, *Map*s, and scalar types.

The elements in a YAML list are defined using the "-" character and they all share the same indentation level:

```
yamlconfig:
  list:
    - item1
    - item2
    - item3
    - item4
```

As a comparison, the properties-based equivalent uses indices:

```
yamlconfig.list[0]=item1
yamlconfig.list[1]=item2
yamlconfig.list[2]=item3
yamlconfig.list[3]=item4
```

For more examples, feel free to take a look at our article on how to define lists and maps using YAML and properties files (/spring-yaml-vs-properties#lists-and-maps).

As a matter of fact, the hierarchical nature of YAML significantly enhances readability compared to properties files. Another interesting feature of YAML is the possibility to define different properties for different Spring profiles (/spring-yaml#spring-yaml-file).

It's worth mentioning that Spring Boot provides out-of-box support for YAML configuration. By design, Spring Boot loads configuration properties from *application.yml* at startup without any extra work.

3. Binding a YAML List to a Simple *List* of Objects

Spring Boot provides the *@ConfigurationProperties (/configuration-properties-in-spring-boot)* annotation to simplify the logic of mapping external configuration data into an object model.

In this section, we'll be using *@ConfigurationProperties* to bind a YAML list into a *List<Object>*.

We start by defining a simple list in *application.yml*:

```
application:
  profiles:
    - dev
    - test
    - prod
    - 1
    - 2
```

Then, we'll create a simple *ApplicationProps* POJO (/java-pojo-class) to hold the logic of binding our YAML list to a *List* of objects:

```
@Component
@ConfigurationProperties(prefix = "application")
public class ApplicationProps {

    private List<Object> profiles;

    // getter and setter

}
```

The *ApplicationProps* class needs to be decorated with *@ConfigurationProperties* to express the intention of mapping all the YAML properties with the specified prefix to an object of *ApplicationProps*.

To bind the *profiles* list we just need to define a field of type *List* and the *@ConfigurationProperties* annotation will take care of the rest.

Notice that we register the *ApplicationProps* class as a normal Spring bean using *@Component*. As a result, we can inject it into other classes in the same way as any other Spring bean.

Finally, we inject the *ApplicationProps* bean into a test class and verify if our *profiles* YAML list is correctly injected as a *List<Object>*:

```
@ExtendWith(SpringExtension.class)
@ContextConfiguration(initializers = ConfigFileApplicationContextInitializer.class)
@EnableConfigurationProperties(value = ApplicationProps.class)
class YamlSimpleListUnitTest {

    @Autowired
    private ApplicationProps applicationProps;

    @Test
    public void whenYamlList_thenLoadSimpleList() {
        assertThat(applicationProps.getProfiles().get(0)).isEqualTo("dev");
        assertThat(applicationProps.getProfiles().get(4).getClass()).isEqualTo(Integer.class);
        assertThat(applicationProps.getProfiles().size()).isEqualTo(5);
    }
}
```

4. Binding YAML Lists to Complex Lists

Now, let's dive deeper and see how to inject nested YAML lists into complex structured *Lists*.

First, let's add some nested lists to *application.yml*:

```yaml
application:
  // ...
  props:
    -
      name: YamlList
      url: http://yamllist.dev
      description: Mapping list in Yaml to list of objects in Spring Boot
    -
      ip: 10.10.10.10
      port: 8091
    -
      email: support@yamllist.dev
      contact: http://yamllist.dev/contact
  users:
    -
      username: admin
      password: admin@10@
      roles:
        - READ
        - WRITE
        - VIEW
        - DELETE
    -
      username: guest
      password: guest@01
      roles:
        - VIEW
```

In this example, we're going to bind the *props* property to a *List<Map<String, Object>>*. Similarly, we'll map *users* into a *List* of *User* objects.

Since each element of the *props* entry holds different keys, then we can inject it as a *List* of *Maps*. Be sure to check out our article on how to inject a map from a YAML file in Spring Boot (/spring-yaml-inject-map).

However, in the case of *users*, all items share the same keys, so to simplify its mapping, we may need to create a dedicated *User* class to encapsulate the keys as fields:

```java
public class ApplicationProps {

    // ...

    private List<Map<String, Object>> props;
    private List<User> users;

    // getters and setters

    public static class User {

        private String username;
        private String password;
        private List<String> roles;

        // getters and setters

    }
}
```

Now we verify that our nested YAML lists are properly mapped:

```
@ExtendWith(SpringExtension.class)
@ContextConfiguration(initializers = ConfigFileApplicationContextInitializer.class)
@EnableConfigurationProperties(value = ApplicationProps.class)
class YamlComplexListsUnitTest {

    @Autowired
    private ApplicationProps applicationProps;

    @Test
    public void whenYamlNestedLists_thenLoadComplexLists() {
        assertThat(applicationProps.getUsers().get(0).getPassword()).isEqualTo("admin@10@");
        assertThat(applicationProps.getProps().get(0).get("name")).isEqualTo("YamlList");

assertThat(applicationProps.getProps().get(1).get("port").getClass()).isEqualTo(Integer.class);
    }

}
```

5. Conclusion

In this tutorial, we learned how to map YAML lists into Java *Lists*. We also checked how to bind complex lists to custom POJOs.

As always, the complete source code for this article is available over on GitHub (https://github.com/eugenp/tutorials/tree/master/spring-boot-modules/spring-boot-properties-2).

CATEGORIES

SPRING (HTTPS://WWW.BAELDUNG.COM/CATEGORY/SPRING/)
REST (HTTPS://WWW.BAELDUNG.COM/CATEGORY/REST/)
JAVA (HTTPS://WWW.BAELDUNG.COM/CATEGORY/JAVA/)
SECURITY (HTTPS://WWW.BAELDUNG.COM/CATEGORY/SECURITY-2/)
PERSISTENCE (HTTPS://WWW.BAELDUNG.COM/CATEGORY/PERSISTENCE/)
JACKSON (HTTPS://WWW.BAELDUNG.COM/CATEGORY/JSON/JACKSON/)
HTTP CLIENT-SIDE (HTTPS://WWW.BAELDUNG.COM/CATEGORY/HTTP/)
KOTLIN (HTTPS://WWW.BAELDUNG.COM/CATEGORY/KOTLIN/)

SERIES

JAVA "BACK TO BASICS" TUTORIAL (/JAVA-TUTORIAL)
JACKSON JSON TUTORIAL (/JACKSON)
HTTPCLIENT 4 TUTORIAL (/HTTPCLIENT-GUIDE)
REST WITH SPRING TUTORIAL (/REST-WITH-SPRING-SERIES)

SPRING PERSISTENCE TUTORIAL (/PERSISTENCE-WITH-SPRING-SERIES)
SECURITY WITH SPRING (/SECURITY-SPRING)

ABOUT

ABOUT BAELDUNG (/ABOUT)
THE COURSES (HTTPS://COURSES.BAELDUNG.COM)
JOBS (/TAG/ACTIVE-JOB/)
THE FULL ARCHIVE (/FULL_ARCHIVE)
WRITE FOR BAELDUNG (/CONTRIBUTION-GUIDELINES)
EDITORS (/EDITORS)
OUR PARTNERS (/PARTNERS)
ADVERTISE ON BAELDUNG (/ADVERTISE)

TERMS OF SERVICE (/TERMS-OF-SERVICE)
PRIVACY POLICY (/PRIVACY-POLICY)
COMPANY INFO (/BAELDUNG-COMPANY-INFO)
CONTACT (/CONTACT)