(/)

An Intro to Spring Cloud Zookeeper

Last modified: November 5, 2018

by baeldung (/author/baeldung/)

Spring Cloud (/category/spring/spring-cloud/)

1. Introduction

In this article, we will get acquainted with Zookeeper and how it's used for Service Discovery which is used as a centralized knowledge about services in the cloud.

Spring Cloud Zookeeper provides Apache Zookeeper (https://zookeeper.apache.org/) integration for Spring Boot apps through autoconfiguration and binding to the Spring Environment.

2. Service Discovery Setup

We will create two apps:

- An app that will provide a service (referred to in this article as the ***Service Provider***)
- An app that will consume this service (called the ***Service Consumer***)

Apache Zookeeper will act as a coordinator in our service discovery setup. Apache Zookeeper installation instructions are available at the following link (https://zookeeper.apache.org/doc/current/zookeeperStarted.html).

3. Service Provider Registration

We will enable service registration by adding the *spring-cloud-starter-zookeeper-discovery* dependency and using the annotation *@EnableDiscoveryClient* in the main application.

Below, we will show this process step by step for the service that returns the "Hello World !" in a response to GET requests.

3.1. Maven Dependencies

First, let's add the required spring-cloud-starter-zookeeper-discovery (https://search.maven.org/classic/#search%7Cga%7C1%7Cspring-cloud-starter-zookeeper-discovery), spring-web (https://search.maven.org/classic/#search%7Cga%7C1%7Cg%3A%22org.springframework%22%20AND%20a%3A%22spring-web%22), spring-cloud-dependencies (https://search.maven.org/classic/#search%7Cga%7C1%7Cg%3A%22org.springframework.cloud%22%20%20AND%20a%3A%22spring-cloud-dependencies%22) and spring-boot-starter (https://search.maven.org/classic/#search%7Cga%7C1%7Ca%3A%22spring-boot-starter%22%20AND%20g%3A%22org.springframework.boot%22) dependencies to our *pom.xml* file:

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter</artifactId>
    <version>1.5.2.RELEASE</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
    <artifactId>spring-web</artifactId>
        <version>4.3.7.RELEASE</version>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-zookeeper-discovery</artifac
        <version>1.0.3.RELEASE</version>
     </dependency>
</dependencies>
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-dependencies</artifactId>
            <version>Brixton.SR7</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

3.2. Service Provider Annotations

Next, we will annotate our main class with *@EnableDiscoveryClient*. This will make the *HelloWorld* application discovery-aware:

```
@SpringBootApplication
@EnableDiscoveryClient
public class HelloWorldApplication {
    public static void main(String[] args) {
        SpringApplication.run(HelloWorldApplication.class, args);
    }
}
```

And a simple controller:

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

3.3. YAML Configurations

Now let us create a YAML *Application.yml* file that will be used for configuring the application log level and informing Zookeeper that the application is discovery-enabled.

The name of the application with which gets registered to Zookeeper is the most important. Later in the service consumer, a *feign* client will use this name during the service discovery:

```
spring:
  application:
    name: HelloWorld
  cloud:
    zookeeper:
      discovery:
        enabled: true
logging:
  level:
    org.apache.zookeeper.ClientCnxn: WARN
```

The spring boot application looks for zookeeper on default port 2181. If zookeeper is located somewhere else, the configuration needs to be added:

```
spring:
  cloud:
    zookeeper:
      connect-string: localhost:2181
```

4. Service Consumer

Now we will create a REST service consumer and registered it using *spring Netflix Feign Client*.

4.1. Maven Dependency

First, let's add the required spring-cloud-starter-zookeeper-discovery (https://search.maven.org/classic/#search%7Cga%7C1%7Cspring-cloud-starter-zookeeper-discovery), spring-web (https://search.maven.org/classic/#search%7Cga%7C1%7Cg%3A%22org.springframework%22%20AND%20a%3A%22spring-web%22), spring-cloud-dependencies (https://search.maven.org/classic/#search%7Cga%7C1%7Cg%3A%22org.springframework.cloud%22%20%20AND%20a%3A%22spring-cloud-dependencies%22), spring-boot-starter-actuator (https://search.maven.org/classic/#search%7Cga%7C1%7Ca%3A%22spring-boot-starter-actuator%22%20AND%20g%3A%22org.springframework.boot%22) and spring-cloud-starter-feign (https://search.maven.org/classic/#search%7Cga%7C1%7Cspring-cloud-starter-feign) dependencies to our *pom.xml* file:

```
<dependencies>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-zookeeper-discovery</artifac
        <version>1.0.3.RELEASE</version>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-actuator</artifactId>
        <version>1.5.2.RELEASE</version>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-feign</artifactId>
        <version>1.2.5.RELEASE</version>
    </dependency>
</dependencies>
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-dependencies</artifactId>
            <version>Brixton.SR7</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

4.2. Service Consumer Annotations

As with the service provider, we will annotate the main class with *@EnableDiscoveryClient* to make it discovery-aware:

```
@SpringBootApplication
@EnableDiscoveryClient
public class GreetingApplication {

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

4.3. Discover Service with Feign Client

We will use the *Spring Cloud Feign Integration*, a project by Netflix that lets you define a declarative REST Client. We declare how the URL looks like and feign takes care of connecting to the REST service.

The *Feign Client* is imported via the *spring-cloud-starter-feign (https://search.maven.org/classic/#search%7Cgav%7C1%7Cg%3A%22org.springframework.cloud%22%20AND%20a%3A%22spring-cloud-starter-feign%22)* package. We will annotate a *@Configuration* with *@EnableFeignClients* to make use of it within the application.

Finally, we annotate an interface with *@FeignClient("service-name")* and auto-wire it into our application for us to access this service programmatically.

Here in the annotation *@FeignClient(name = "HelloWorld")*, we refer to the *service-name* of the service producer we previously created.

```
@Configuration
@EnableFeignClients
@EnableDiscoveryClient
public class HelloWorldClient {

    @Autowired
    private TheClient theClient;

    @FeignClient(name = "HelloWorld")
    interface TheClient {

        @RequestMapping(path = "/helloworld", method = RequestMethod.
        @ResponseBody
    String helloWorld();
    }
    public String HelloWorld() {
        return theClient.HelloWorld();
    }
}
```

4.4. Controller Class

The following is the simple service controller class that will call the service provider function on our feign client class to consume the service (whose details are abstracted through service discovery) via the injected interface *helloWorldClient* object and displays it in response:

```
@RestController
public class GreetingController {

    @Autowired
    private HelloWorldClient helloWorldClient;

    @GetMapping("/get-greeting")
    public String greeting() {
        return helloWorldClient.helloWorld();
    }
}
```

4.5. YAML Configurations

Next, we create a YAML file *Application.yml* very similar to the one used before. That configures the application's log level:

```
logging:
  level:
    org.apache.zookeeper.ClientCnxn: WARN
```

The application looks for the Zookeeper on default port 2181. If Zookeeper is located somewhere else, the configuration needs to be added:

```
spring:
  cloud:
    zookeeper:
      connect-string: localhost:2181
```

5. Testing the Setup

The HelloWorld REST service registers itself with Zookeeper on deployment. Then the *Greeting* service acting as the service consumer calls the *HelloWorld* service using the Feign client.

Now we can build and run these two services.

Finally, we'll point our browser to *http://localhost:8083/get-greeting* (http://localhost:8080/get-greeting), and it should display:

```
Hello World!
```

6. Conclusion

In this article, we have seen how to implement service discovery using *Spring Cloud Zookeeper* and we registered a service called *HelloWorld* within Zookeeper server to be discovered and consumed by the *Greeting* service using a *Feign Client* without knowing its location details.

As always, the code for this article is available on the GitHub (https://github.com/eugenp/tutorials/tree/master/spring-cloud/spring-cloud-zookeeper).

CATEGORIES

SPRING (/CATEGORY/SPRING/)

REST (/CATEGORY/REST/)
JAVA (/CATEGORY/JAVA/)
SECURITY (/CATEGORY/SECURITY-2/)
PERSISTENCE (/CATEGORY/PERSISTENCE/)
JACKSON (/CATEGORY/JACKSON/)
HTTP CLIENT (/CATEGORY/HTTP/)
KOTLIN (/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)
CONSULTING WORK (/CONSULTING)
META BAELDUNG (HTTP://META.BAELDUNG.COM/)
THE FULL ARCHIVE (/FULL_ARCHIVE)
WRITE FOR BAELDUNG (/CONTRIBUTION-GUIDELINES)
CONTACT (/CONTACT)
EDITORS (/EDITORS)
ADVERTISE ON BAELDUNG (/ADVERTISE)

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