

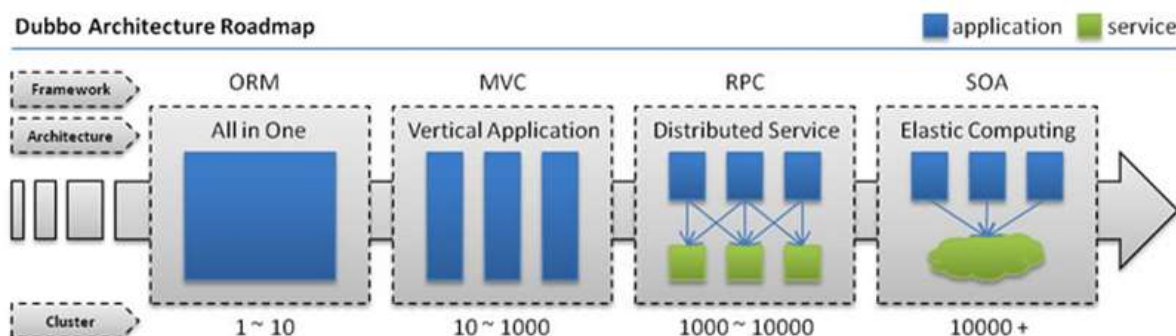
19、Spring Boot与分布式

2020年8月17日 16:14

- 一、分布式应用

在分布式系统中，国内常用zookeeper+dubbo组合，而Spring Boot推荐使用全栈的Spring，Spring Boot+Spring Cloud。

分布式系统：



- 单一应用架构

当网站流量很小时，只需一个应用，将所有功能都部署在一起，以减少部署节点和成本。此时，用于简化增删改查工作量的数据访问框架(ORM)是关键。

- 垂直应用架构

当访问量逐渐增大，单一应用增加机器带来的加速度越来越小，将应用拆成互不相干的几个应用，以提升效率。此时，用于加速前端页面开发的Web框架(MVC)是关键。

- 分布式服务架构

当垂直应用越来越多，应用之间交互不可避免，将核心业务抽取出来，作为独立的服务，逐渐形成稳定的服务中心，使前端应用能更快速的响应多变的市场需求。此时，用于提高业务复用及整合的分布式服务框架(RPC)是关键。

- 流动计算架构

当服务越来越多，容量的评估，小服务资源的浪费等问题逐渐显现，此时需增加一个调度中心基于访问压力实时管理集群容量，提高集群利用率。此时，用于提高机器利用率的资源调度和治理中心(SOA)是关键。

- 二、Zookeeper和Dubbo解决远程调用问题

- ZooKeeper

ZooKeeper 是一个分布式的，开放源码的分布式应用程序协调服务。它是一个为分布式应用提供一致性服务的软件，提供的功能包括：配置维护、域名服务、分布式同步、组服务等。

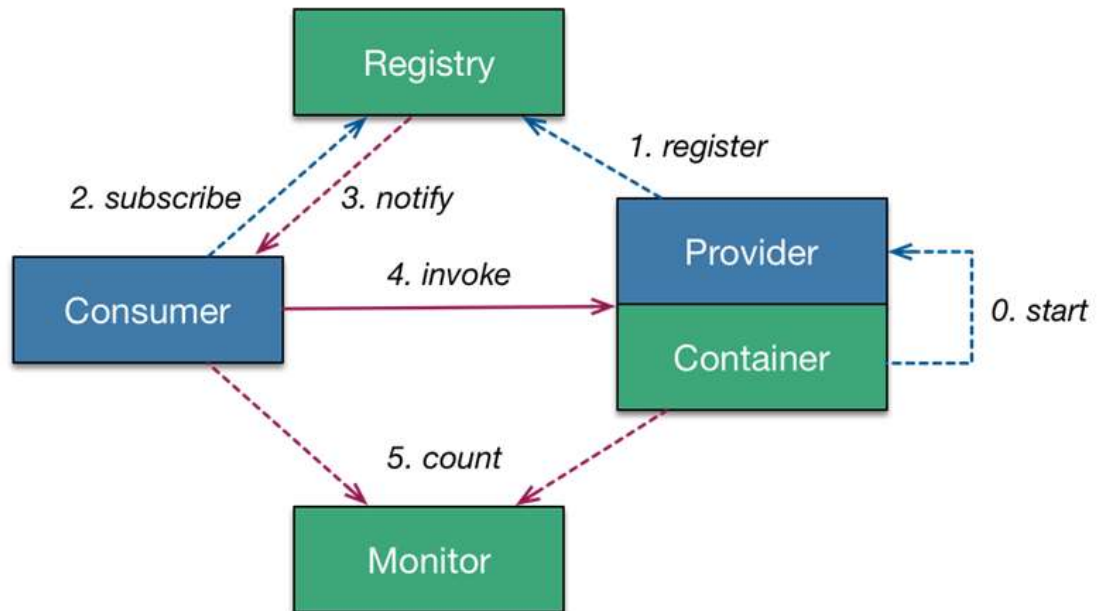
- Dubbo

Dubbo是Alibaba开源的分布式服务框架，它最大的特点是按照分层的方式来架

构，使用这种方式可以使各个层之间解耦合（或者最大限度地松耦合）。从服务模型的角度来看，Dubbo采用的是一种非常简单的模型，要么是提供方提供服务，要么是消费方消费服务，所以基于这一点可以抽象出服务提供方（Provider）和服务消费方（Consumer）两个角色。

Dubbo Architecture

-----> init -.-.-.-> async ———> sync



○ 1、安装zookeeper作为注册中心

1- 拉取zookeeper

`docker pull zookeeper`

2- 启动zookeeper

`docker run --name zk01 -p 2181:2181 --restart always -d zookeeper`

○ 2、引入依赖

`<dependency>`

`<groupId>org.apache.dubbo</groupId>`

`<artifactId>dubbo-spring-boot-starter</artifactId>`

`<version>2.7.3</version>`

`<exclusions>`

`<exclusion>`

`<groupId>org.slf4j</groupId>`

`<artifactId>slf4j-log4j12</artifactId>`

`</exclusion>`

`</exclusions>`

`</dependency>`

`<dependency>`

`<groupId>org.apache.dubbo</groupId>`

```

        <artifactId>dubbo</artifactId>
        <version>2.7.3</version>
    </dependency>
    <dependency>
        <groupId>org.apache.curator</groupId>
        <artifactId>curator-framework</artifactId>
        <version>4.0.1</version>
    </dependency>
    <dependency>
        <groupId>org.apache.curator</groupId>
        <artifactId>curator-recipes</artifactId>
        <version>2.8.0</version>
    </dependency>
    <dependency>
        <groupId>org.apache.zookeeper</groupId>
        <artifactId>zookeeper</artifactId>
        <version>3.4.13</version>
        <type>pom</type>
        <exclusions>
            <exclusion>
                <groupId>org.slf4j</groupId>
                <artifactId>slf4j-log4j12</artifactId>
            </exclusion>
        </exclusions>
    </dependency>
    <dependency>
        <groupId>com.101tec</groupId>
        <artifactId>zkclient</artifactId>
        <version>0.10</version>
    </dependency>

```

○ 3、配置文件编写

提供者

dubbo.application.name=provider-ticket

dubbo.registry.address=zookeeper://39.101.170.233:2181

dubbo.scan.base-packages=com.atlhq.ticket.service

消费者

dubbo.application.name=consumer-user

#注册中心地址

dubbo.registry.address=zookeeper://39.101.170.233:2181

○ 4、编写服务提供者

- Service层接口

```
package com.atlhq.ticket.service;

public interface TicketService {

    public String getTicket();

}
```

- Service层实现类

```
package com.atlhq.ticket.service;

import org.apache.dubbo.config.annotation.Service;
import org.springframework.stereotype.Component;

@Component
@Service//将服务发布出去

public class TicketServiceImpl implements TicketService {

    @Override

    public String getTicket() {

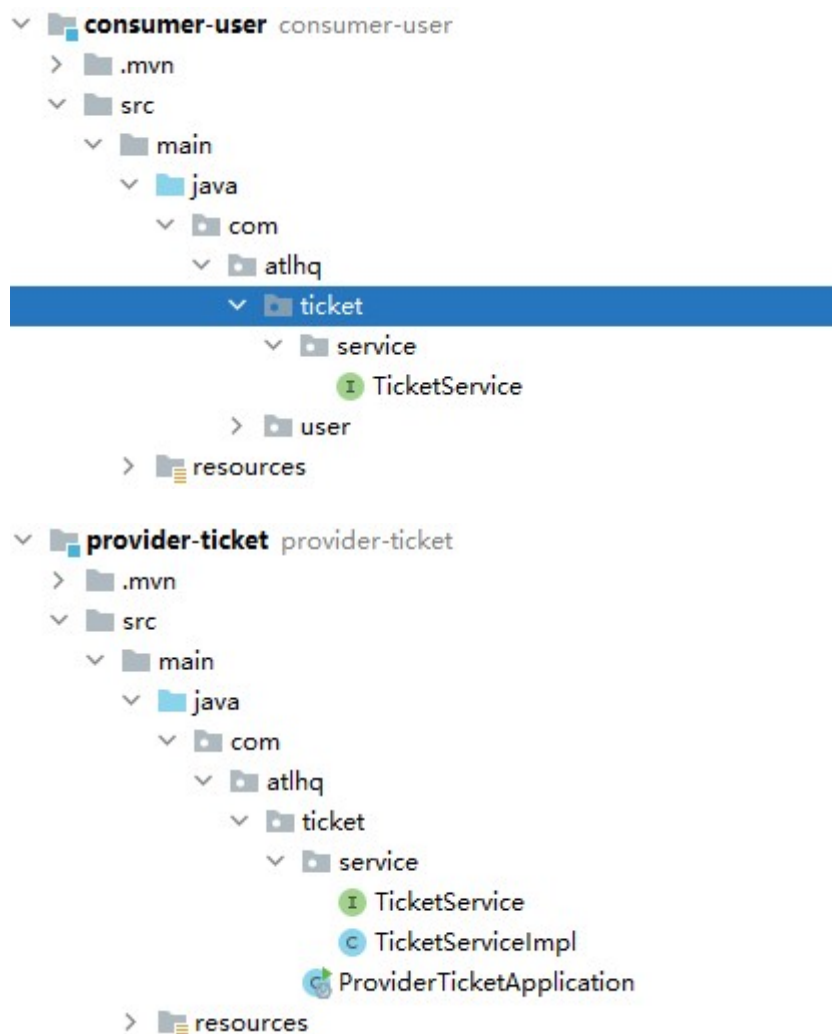
        return "《电影》";

    }

}
```

- 5、编写服务消费者

需要将提供者同一个包下的包与接口复制过来



编写Service层

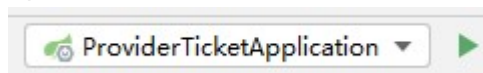
```

import org.apache.dubbo.config.annotation.Reference;
import org.springframework.stereotype.Service;
@Service
public class UserService {
    //远程调用注解，注意导正确的包
    @Reference
    TicketService ticketService;
    public void hello(){
        System.out.println("买到票了: "+ticketService.getTicket());
    }
}

```

○ 6、启动

1) 先启动提供者



2) 启动消费者



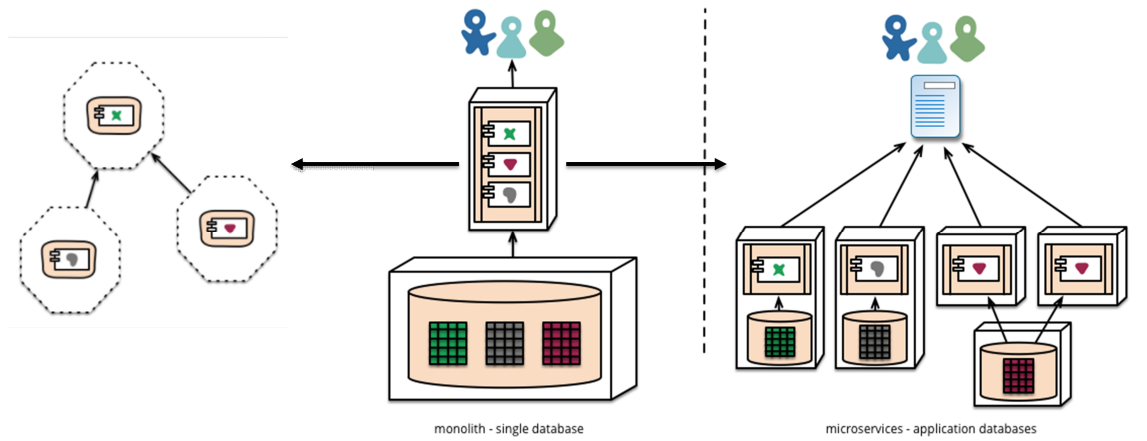
• 三、Spring Boot和Spring Cloud

- Spring Cloud是一个分布式的整体解决方案。Spring Cloud 为开发者提供了**在分布式系统（配置管理，服务发现，熔断，路由，微代理，控制总线，一次性token，全局锁，leader选举，分布式session，集群状态）中快速构建的工具**，使用Spring Cloud的开发者可以快速的启动服务或构建应用、同时能够快速和云平台资源进行对接。

○ SpringCloud分布式开发五大常用组件

- 服务发现——Netflix Eureka
- 客户端负载均衡——Netflix Ribbon
- 断路器——Netflix Hystrix
- 服务网关——Netflix Zuul
- 分布式配置——Spring Cloud Config

○ 微服务



○ 1、服务注册

- 创建一个空工程，添加三个模块

```
> consumer-user consumer-us
> eureka-server eureka-server
> provider-ticket provider-tick
```

- 编写eureka-server（注册中心）

- 配置Eureka信息（在application.yml中配置）

server:

port:8761

eureka:

instance:

hostname:eureka-server#eureka实例的主机名

client:

register-with-eureka:false#不把自己注册到eureka上

fetch-registry:false#不从eureka上来获取服务的注册信息

service-url:

defaultZone:http://localhost:8761/eureka/

- 在主启动器上添加@EnableEurekaServer

package com.atlhq.eurekaserver;

import org.springframework.boot.SpringApplication;

import

org.springframework.boot.autoconfigure.SpringBootApplication;

import

org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

/*

* 注册中心

* 1、配置Eureka信息

* 2、@EnableEurekaServer

* */

@EnableEurekaServer

@SpringBootApplication

```

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

```

- 编写provider-ticket

- application.yml

```

server:
port:8002
spring:
application:
name:provider-ticket
eureka:
instance:
prefer-ip-address:true#注册服务的时候使用服务的ip地址
client:
service-url:
defaultZone:http://localhost:8761/eureka/

```

- Service层

```

package com.atlhq.providerticket.service;
import org.springframework.stereotype.Service;
@Service
public class TicketService {
    public String getTicket(){
        System.out.println("8002");
        return "《姜子牙》";
    }
}

```

- Controller层



```

package com.atlhq.providerticket.controller;
import com.atlhq.providerticket.service.TicketService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
@RestController
public class TicketController {
    @Autowired
    TicketService ticketService;
    @GetMapping("/ticket")
    public String getTicket(){
        return ticketService.getTicket();
    }
}

```

```
}
}
```

- 将provider打包多个放一起（注意修改端口号）

	provider-ticket-0.0.1-SNAPSHOT-800...	2020/8/20 11:09	JAR 文件	42,787 KB
	provider-ticket-0.0.1-SNAPSHOT-800...	2020/8/20 11:13	JAR 文件	42,787 KB

- 注册中心先启动然后在cmd中使用java-jar命令启动两个provider
- 访问8761端口

Instances currently registered with Eureka			
Application	AMIs	Availability Zones	Status
PROVIDER-TICKET	n/a (2)	(2)	UP (2) - localhost:provider-ticket:8001 , localhost:provider-ticket:8002

○ 2、服务发现与消费

- 编写consumer-ticket

- application.yml

```
spring:
  application:
    name:consumer-user
  server:
    port:8200
  eureka:
    instance:
      prefer-ip-address:true #注册服务的时候使用服务的ip地址
    client:
      service-url:
        defaultZone:http://localhost:8761/eureka/
```

- 在主启动器上开启发现服务与负载均衡功能，并编写RestTemplate组件

```
package com.atlhq.consumeruser;
import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBootApplication;
import
org.springframework.cloud.client.discovery.EnableDiscoveryClient;
import
org.springframework.cloud.client.loadbalancer.LoadBalanced;
import org.springframework.context.annotation.Bean;
import org.springframework.web.client.RestTemplate;
@EnableDiscoveryClient//开启发现服务功能
@SpringBootApplication
```



```

public class ConsumerUserApplication {
    public static void main(String[] args) {
        SpringApplication.run(ConsumerUserApplication.class,
args);
    }
    @LoadBalanced//使用负载均衡机制
    @Bean
    public RestTemplate restTemplate(){
        return new RestTemplate();
    }
}

```

□ Controller

```

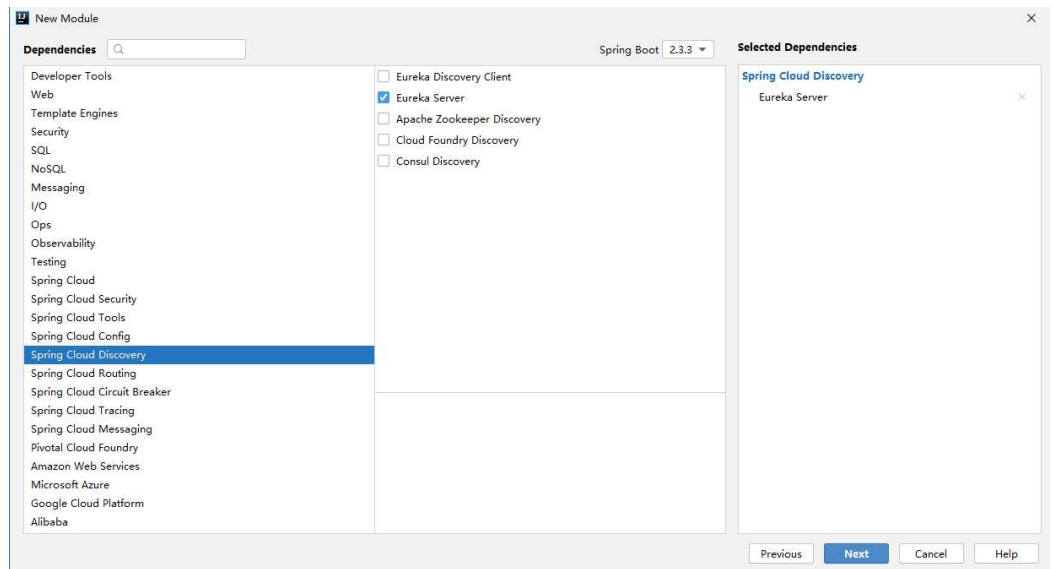
UserController
package com.atlhq.consumeruser.controller;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.client.RestTemplate;
@RestController
public class UserController {
    @Autowired
    RestTemplate restTemplate;
    @GetMapping("/buy")
    public String buyTicket(String name){
        String ticket =
restTemplate.getForObject("http://PROVIDER-TICKET/ticket",
String.class);
        return name + "购买了" + ticket;
    }
}

```

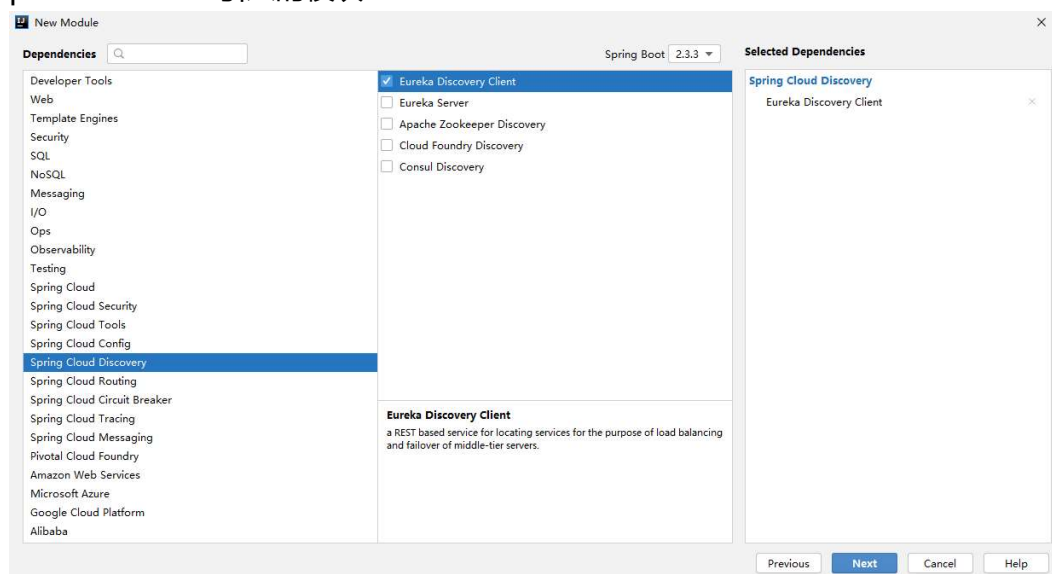
- 启动eureka-server (注册中心) ->启动多个provider-ticket->启动 consumer-ticket

○ Notes:

- eureka-server引入的模块



provider-ticket引入的模块



consumer-ticket引入的模块

