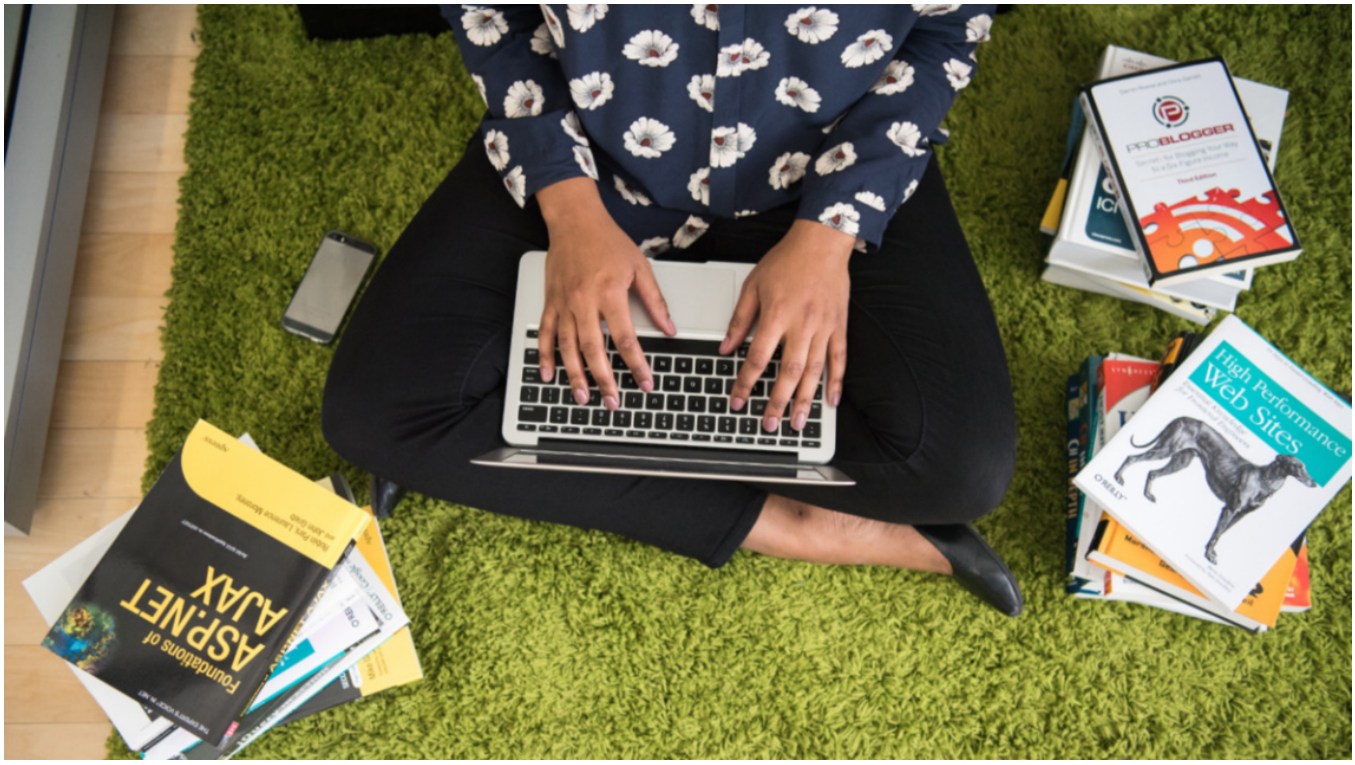
04 | 如何发布和引用服务?

2018-08-30 胡忠想



04 | 如何发布和引用服务?
朗读人：胡忠想　07′55″ | 3.63M

从这期开始，我将陆续给你讲解微服务各个基本组件的原理和实现方式。

今天我要与你分享的第一个组件是服务发布和引用。我在前面说过，想要构建微服务，首先要解决的问题是，服务提供者如何发布一个服务，服务消费者如何引用这个服务。具体来说，就是这个服务的接口名是什么？调用这个服务需要传递哪些参数？接口的返回值是什么类型？以及一些其他接口描述信息。

我前面说过，最常见的服务发布和引用的方式有三种：

- RESTful API
- XML 配置
- IDL 文件

下面我就结合具体的实例，逐个讲解每一种方式的具体使用方法以及各自的应用场景，以便你在选型时作参考。

RESTful API

首先来说说 RESTful API 的方式，主要被用作 HTTP 或者 HTTPS 协议的接口定义，即使在非微服务架构体系下，也被广泛采用。

下面是开源服务化框架 Motan（https://github.com/weibocom/motan）发布 RESTful API 的例子，它发布了三个 RESTful 格式的 API，接口声明如下：

```
@Path("/rest")
 public interface RestfulService {
     @GET
     @Produces(MediaType.APPLICATION_JSON)

     List<User> getUsers(@QueryParam("uid") int uid);


     @GET
     @Path("/primitive")
     @Produces(MediaType.TEXT_PLAIN)
     String testPrimitiveType();


     @POST
     @Consumes(MediaType.APPLICATION_FORM_URLENCODED)
     @Produces(MediaType.APPLICATION_JSON)
     Response add(@FormParam("id") int id, @FormParam("name") String name);
```

具体的服务实现如下：

```
public class RestfulServerDemo implements RestfulService {


     @Override
     public List<User> getUsers(@CookieParam("uid") int uid) {
         return Arrays.asList(new User(uid, "name" + uid));
     }


     @Override
```

```
    public String testPrimitiveType() {
        return "helloworld!";
    }

    @Override
    public Response add(@FormParam("id") int id, @FormParam("name") String name) {
        return Response.ok().cookie(new NewCookie("ck", String.valueOf(id))).entity(new User
    }
```

服务提供者这一端通过部署代码到 Tomcat 中，并配置 Tomcat 中如下的 web.xml，就可以通过 servlet 的方式对外提供 RESTful API。

```
<listener>
    <listener-class>com.weibo.api.motan.protocol.restful.support.servlet.RestfulServletConta
 </listener>

 <servlet>
    <servlet-name>dispatcher</servlet-name>
    <servlet-class>org.jboss.resteasy.plugins.server.servlet.HttpServletDispatcher</servlet-
    <load-on-startup>1</load-on-startup>
    <init-param>
        <param-name>resteasy.servlet.mapping.prefix</param-name>
        <param-value>/servlet</param-value>  <!-- 此处实际为 servlet-mapping 的 url-pattern，
    </init-param>
 </servlet>

 <servlet-mapping>
    <servlet-name>dispatcher</servlet-name>
    <url-pattern>/servlet/*</url-pattern>
 </servlet-mapping>
```

这样服务消费者就可以通过 HTTP 协议调用服务了，因为 HTTP 协议本身是一个公开的协议，对于服务消费者来说几乎没有学习成本，所以比较适合用作跨业务平台之间的服务协议。比如你有一个服务，不仅需要在业务部门内部提供服务，还需要向其他业务部门提供服务，甚至开放给外网提供服务，这时候采用 HTTP 协议就比较合适，也省去了沟通服务协议的成本。

XML 配置

接下来再来给你讲下 XML 配置方式，这种方式的服务发布和引用主要分三个步骤：

- 服务提供者定义接口，并实现接口。
- 服务提供者进程启动时，通过加载 server.xml 配置文件将接口暴露出去。
- 服务消费者进程启动时，通过加载 client.xml 配置文件来引入要调用的接口。

我继续以服务化框架 Motan 为例，它还支持以 XML 配置的方式来发布和引用服务。

首先，服务提供者定义接口。

```
public interface FooService {
    public String hello(String name);
}
```

然后服务提供者实现接口。

```
public class FooServiceImpl implements FooService {

    public String hello(String name) {
        System.out.println(name + " invoked rpc service");
        return "hello " + name;
    }
}
```

最后服务提供者进程启动时，加载 server.xml 配置文件，开启 8002 端口监听。

server.xml 配置如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:motan="http://api.weibo.com/schema/motan"
 xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.o
   http://api.weibo.com/schema/motan http://api.weibo.com/schema/motan.xsd">
```

```
    <!-- service implemention bean -->
    <bean id="serviceImpl" class="quickstart.FooServiceImpl" />
    <!-- exporting service by Motan -->
    <motan:service interface="quickstart.FooService" ref="serviceImpl" export="8002" />
</beans>
```

服务提供者加载 server.xml 的代码如下：

```
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Server {

    public static void main(String[] args) throws InterruptedException {
        ApplicationContext applicationContext = new ClassPathXmlApplicationContext("classpath
        System.out.println("server start...");
    }
}
```

服务消费者要想调用服务，就必须在进程启动时，加载配置 client.xml，引用接口定义，然后发起调用。

client.xml 配置如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:motan="http://api.weibo.com/schema/motan"
xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.or
   http://api.weibo.com/schema/motan http://api.weibo.com/schema/motan.xsd">

    <!-- reference to the remote service -->
    <motan:referer id="remoteService" interface="quickstart.FooService" directUrl="localhost:
</beans>
```

服务消费者启动时，加载 client.xml 的代码如下。

```
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;



public class Client {


    public static void main(String[] args) throws InterruptedException {
        ApplicationContext ctx = new ClassPathXmlApplicationContext("classpath:motan_client.x
        FooService service = (FooService) ctx.getBean("remoteService");
        System.out.println(service.hello("motan"));
    }
}
```

就这样，通过在服务提供者和服务消费者之间维持一份对等的 XML 配置文件，来保证服务消费者按照服务提供者的约定来进行服务调用。在这种方式下，如果服务提供者变更了接口定义，不仅需要更新服务提供者加载的接口描述文件 server.xml，还需要同时更新服务消费者加载的接口描述文件 client.xml。

一般是私有 RPC 框架会选择 XML 配置这种方式来描述接口，因为私有 RPC 协议的性能要比 HTTP 协议高，所以在对性能要求比较高的场景下，采用 XML 配置的方式比较合适。但这种方式对业务代码侵入性比较高，XML 配置有变更的时候，服务消费者和服务提供者都要更新，所以适合公司内部联系比较紧密的业务之间采用。如果要应用到跨部门之间的业务调用，一旦有 XML 配置变更，需要花费大量精力去协调不同部门做升级工作。在我经历的实际项目里，就遇到过一次底层服务的接口升级，需要所有相关的调用方都升级，为此花费了大量时间去协调沟通不同部门之间的升级工作，最后经历了大半年才最终完成。所以对于 XML 配置方式的服务描述，一旦应用到多个部门之间的接口格式约定，如果有变更，最好是新增接口，不到万不得已不要对原有的接口格式做变更。

IDL 文件

IDL 就是接口描述语言（interface description language）的缩写，通过一种中立的方式来描述接口，使得在不同的平台上运行的对象和不同语言编写的程序可以相互通信交流。比如你用 Java 语言实现提供的一个服务，也能被 PHP 语言调用。

也就是说 IDL 主要是用作跨语言平台的服务之间的调用，有两种最常用的 IDL：一个是 Facebook 开源的Thrift 协议，另一个是 Google 开源的gRPC 协议。无论是 Thrift 协议还是

gRPC 协议，它们的工作原理都是类似的。

接下来，我以 gRPC 协议为例，给你讲讲如何使用 IDL 文件方式来描述接口。

gRPC 协议使用 Protobuf 简称 proto 文件来定义接口名、调用参数以及返回值类型。

比如文件 helloword.proto 定义了一个接口 SayHello 方法，它的请求参数是 HelloRequest，它的返回值是 HelloReply。

```
// The greeter service definition.
service Greeter {
  // Sends a greeting
  rpc SayHello (HelloRequest) returns (HelloReply) {}
  rpc SayHelloAgain (HelloRequest) returns (HelloReply) {}

}

// The request message containing the user's name.
message HelloRequest {
  string name = 1;
}

// The response message containing the greetings
message HelloReply {
  string message = 1;
}
```

假如服务提供者使用的是 Java 语言，那么利用 protoc 插件即可自动生成 Server 端的 Java 代码。

```
private class GreeterImpl extends GreeterGrpc.GreeterImplBase {

  @Override
  public void sayHello(HelloRequest req, StreamObserver<HelloReply> responseObserver) {
    HelloReply reply = HelloReply.newBuilder().setMessage("Hello " + req.getName()).build();
    responseObserver.onNext(reply);
    responseObserver.onCompleted();
  }
```

```
  @Override
  public void sayHelloAgain(HelloRequest req, StreamObserver<HelloReply> responseObserver) {
    HelloReply reply = HelloReply.newBuilder().setMessage("Hello again " + req.getName()).bui
    responseObserver.onNext(reply);
    responseObserver.onCompleted();
  }
}
```

假如服务消费者使用的也是 Java 语言，那么利用 protoc 插件即可自动生成 Client 端的 Java 代码。

```
public void greet(String name) {
  logger.info("Will try to greet " + name + " ...");
  HelloRequest request = HelloRequest.newBuilder().setName(name).build();
  HelloReply response;
  try {
    response = blockingStub.sayHello(request);
  } catch (StatusRuntimeException e) {
    logger.log(Level.WARNING, "RPC failed: {0}", e.getStatus());
    return;
  }
  logger.info("Greeting: " + response.getMessage());
  try {
    response = blockingStub.sayHelloAgain(request);
  } catch (StatusRuntimeException e) {
    logger.log(Level.WARNING, "RPC failed: {0}", e.getStatus());
    return;
  }
  logger.info("Greeting: " + response.getMessage());
}
```

假如服务消费者使用的是 PHP 语言，那么利用 protoc 插件即可自动生成 Client 端的 PHP 代码。

```
    $request = new Helloworld\HelloRequest();
```

```
$request->setName($name);
list($reply, $status) = $client->SayHello($request)->wait();
$message = $reply->getMessage();
list($reply, $status) = $client->SayHelloAgain($request)->wait();
$message = $reply->getMessage();
```

由此可见，gRPC 协议的服务描述是通过 proto 文件来定义接口的，然后再使用 protoc 来生成不同语言平台的客户端和服务端代码，从而具备跨语言服务调用能力。

有一点特别需要注意的是，在描述接口定义时，IDL 文件需要对接口返回值进行详细定义。如果接口返回值的字段比较多，并且经常变化时，采用 IDL 文件方式的接口定义就不太合适了。一方面可能会造成 IDL 文件过大难以维护，另一方面只要 IDL 文件中定义的接口返回值有变更，都需要同步所有的服务消费者都更新，管理成本就太高了。

我在项目实践过程中，曾经考虑过采用 Protobuf 文件来描述微博内容接口，但微博内容返回的字段有几百个，并且有些字段不固定，返回什么字段是业务方自定义的，这种情况采用 Protobuf 文件来描述的话会十分麻烦，所以最终不得不放弃这种方式。

总结

今天我给你介绍了服务描述最常见的三种方式：RESTful API、XML 配置以及 IDL 文件。

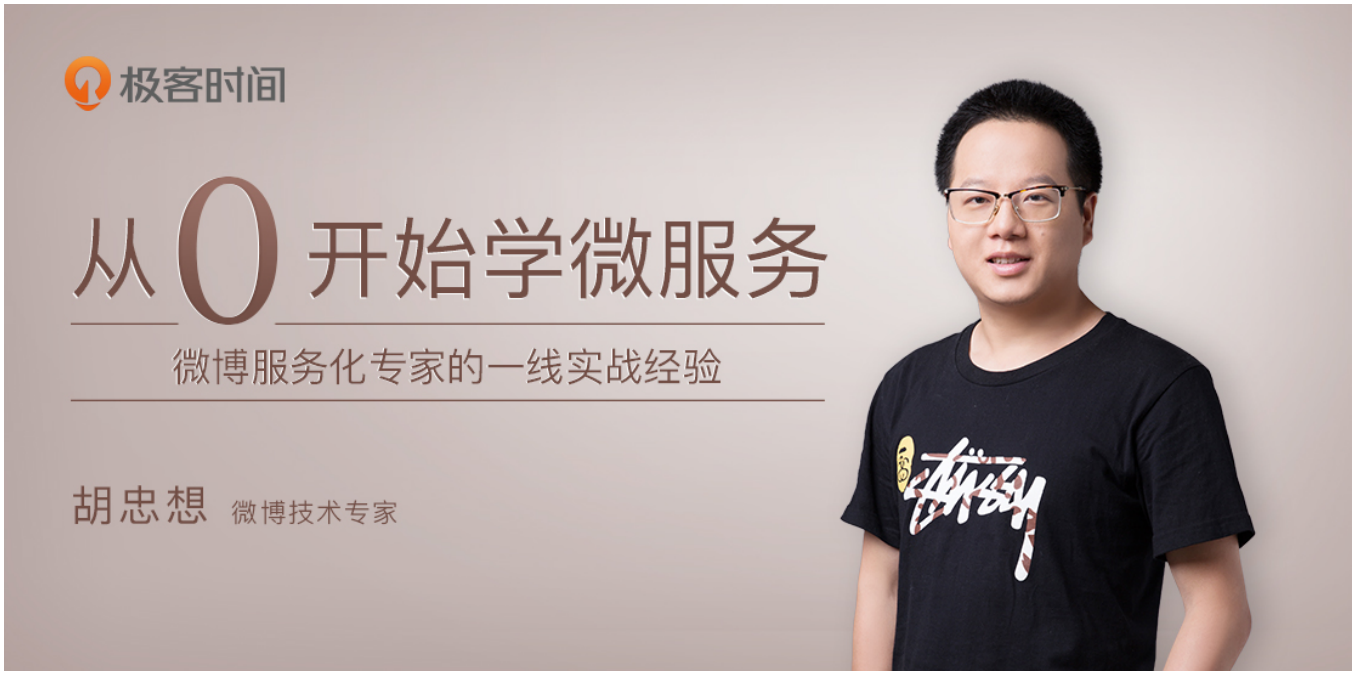
具体采用哪种服务描述方式是根据实际情况决定的，通常情况下，如果只是企业内部之间的服务调用，并且都是 Java 语言的话，选择 XML 配置方式是最简单的。如果企业内部存在多个服务，并且服务采用的是不同语言平台，建议使用 IDL 文件方式进行描述服务。如果还存在对外开放服务调用的情形的话，使用 RESTful API 方式则更加通用。

服务描述方式	使用场景	缺点
RESTful API	跨语言平台，组织内外皆可	使用了HTTP作为通信协议，相比TCP协议，性能较差
XML配置	Java平台，一般用作组织内部	不支持跨语言平台
IDL文件	跨语言平台，组织内外皆可	修改或者删除PB字段不能向前兼容

思考题

针对你的业务场景思考一下，假如要进行服务化，你觉得使用哪种服务描述最合适？为什么？

欢迎你在留言区写下自己的思考，与我一起讨论。



精选留言

陈华应 2

一般是两种结合，各个服务之间的调用通过xml的形式，对外暴露restful接口，如给前端调用等。dubbo，pigeon， springboot实现两种不同形式的服务接口定义与调用。目前都是JAVA，还没有涉及到多语言。

2018-08-30

小胖狗 1

我们这边是这样的。同一个服务，对内提供的话就使用RPC，对外提供的话，就走Restful API

2018-08-30

明天更美好 1

我觉得我们如果做的话，xml方式适合，我们对性能要求交高，对在提供能力要求1.5wtps，响应60ms以内，所以xml比较合适，但是我还想请教胡老师一个问题，就是我们目前还是单体应用，有23接口，就认证接口并发交高，有必要做微服务吗？目前光tomcat部署了30多个，感觉好烦，每次升级换包老半天。而且是没网部署。

2018-08-30

松花皮蛋me 0

还是推荐xml方式，可以优化为长连接

2018-08-30

王宏达达达 0

我们用RESTful API来提供服务信息，首先简单高效，其次企内不需要太规范的情况下增加开发效率，最重要的是国内用的这种只需两个post和get其他都不需要用好像。我还是想问下，分布式是否就包含soa和微服务

2018-08-30

不够 0

我司的微服务大多数都是内部调用，比较在意性能，采用的xml，依赖相同的api接口，有接口变更时，一般就升级接口

2018-08-30

钟悠 0

边看文章，边想dubbo

2018-08-30

一步 0

服务的注册和发现，不是应该需要一个服务注册中心呢?

2018-08-30

stamaimer 0

学习了

2018-08-30

stamaimer 0

1.代码太多了
2.为什么对性能要求较高的场景就需要使用 XML，可以使用 JSON 之类的吗?
3.IDL 是一种语言，为什么最常见的 IDL 是两种协议?
4.Restful 也能实现跨语言调用，那么 Restful 和 IDL 在跨语言调用方面各有啥优缺点?

2018-08-30

stamaimer 0

1.代码太多了
2.为什么对性能高就要使用 XML 描述接口，可以使用 JSON 吗?
3.IDL 是语言，为什么最常见的两种 IDL 是两种协议?
4.RESTful 也可以实现跨语言，RESTful 和 IDL 在跨语言调用方面各自的优缺点是什么?

2018-08-30

郁 0

对内grpc，对外http+json

2018-08-30

fish 0

采用protobuf如果遇到web前端是不是最佳实践还得在protobuf上面再封一层restful返回给页面

2018-08-30

Hurt

0

目前 微服务 还是以java为主吗 老师

2018-08-30