使用远程服务

远程调用：

- **远程方法调用(Remote Method Invocation, RMI)**
- **Caucho的Hessian和Burlap**
- **Spring基于HTTP的远程服务**
- **使用JAX-RPC和JAX-WS的Web Service**

1 远程调用概述

RPC	适用场景
远程方法调用（RMI）	不考虑网络限制，访问/发布基于Java的服务
Hessian或Burlap	考虑网络限制，通过HTTP访问/发布基于Java的服务，Hessian是二进制协议，而Burlap是基于XML的
HTTP invoker	考虑网络限制。并希望使用基于XML或转悠的序列化机制实现Java序列化时，访问/发布基于spring的服务
JAX-RPC和JAX-WS	访问/发布平台独立的、基于SOAP的Web服务

服务作为Spring所管理的bean配置到我们的应用中共。通过一个代理工厂bean实现。这个bean能够把远程服务像本地对象一样爪给你配刀其他的bean属性中去。

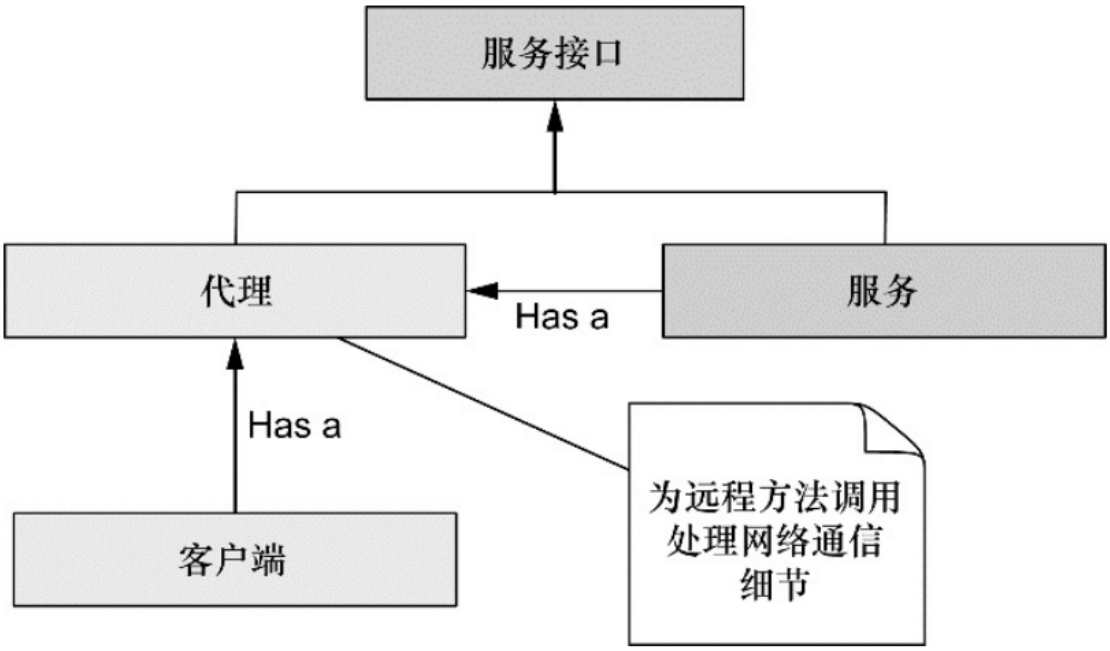


图15.2 在Spring中，远程服务被代理，所以它们能够像其他Spring bean一样被装配到客户端代码中

客户端向代理发起调用，就像代理提供了这些服务一样。代理代表客户端域远程服务进行通讯，由他处理连接的细节并向远程服务发起调用

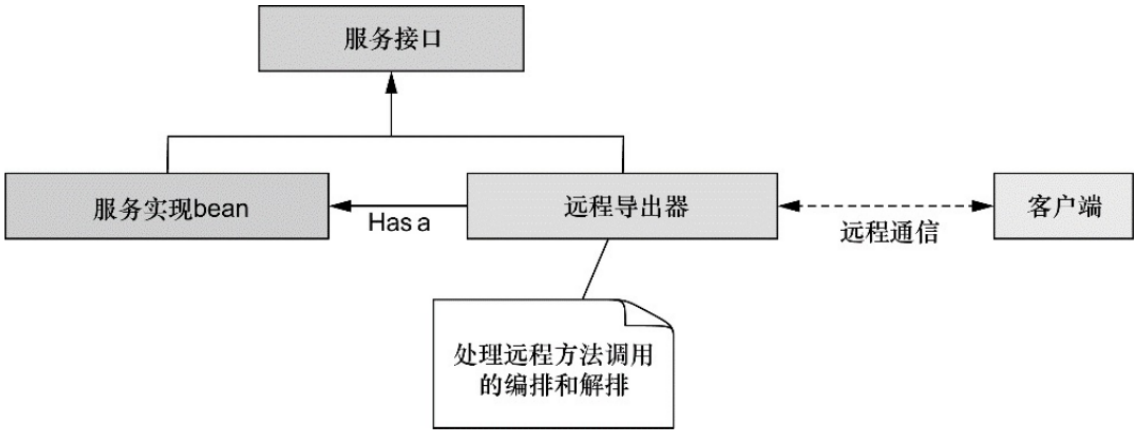


图15.3 使用远程导出器将Spring管理的bean发布为远程服务

无论开发法的是使用远程服务的代码，还是实现这些服务的代码，或者两者都有，在Spring助攻，使用远程服务纯粹是一个配置问题。不需要编写任何Java代码就可以支持远程调用。服务bean也不需要关心他们是否参与了一个RPC

2 使用RMI

导出RMI服务

- 1. 编写一个服务实现类，勒种的方法必须抛出java.rmi.RemoteException
- 2. 创建一个继承与java.rmi.Remote的服务接口
- 3. 运行RMI编译器，创建客户端stub类和服务端skeleton
- 4. 启动一个RMI注册表，以持有这些服务
- 5. 在RMI注册表中注册服务

2.1 在Spring中配置RMI服务

```
public interface SpitterService {
    List<Spittle> getRecentSpittles(int count);

    void saveSpittle(Spittle spittle);

    void saveSpitter(Spitter spitter);

    Spittle getSpittleById(long id);

    List<Spittle> getSpittlesForSpitter(Spitter spitter);

    List<Spittle> GetSpittlesForSpittler(String name);

    void deleteSpittle(long id);

    Spitter getSpitter(long id);

    Spitter getSpitter(String name);

    void startFollowing(Spitter follower, Spitter followee);

    List<Spitter> getAllSpitter();
}
```

RmiServiceExplorter可以吧Spring管理的bean发布为RMI服务，RmiServiceExplorter把bean包装在一个适配器类中没然后适配器类被绑定到RMI注册表中，并且代理到服务类的请求。配置方式

```
@Bean
public RmiServiceExporter rmiServiceExporter(SpitterService spitterService)
{
    RmiServiceExporter serviceExporter = new RmiServiceExporter();
    serviceExporter.setService(spitterService);
    serviceExporter.setServiceName("SpitterService");
    serviceExporter.setServiceInterface(SpitterService.class);
    serviceExporter.setRegistryHost("127.0.0.1");
    serviceExporter.setRegistryPort(1199);
    return serviceExporter;
}
```

默认情况下，RmiServiceExporter会尝试绑定到本地机器1099端口上的RMI注册表。如果在该端口没有发现RMI注册表，RmiServiceExporter将会启动一个注册表。可以通过registryPory和registryHost属性指定绑定的端口或主机。

2.2 装配RMI服务

Spring的RmiProxyFactoryBean是一个工厂bean，可以为RMI服务创建代理

```
@Bean
public RmiProxyFactoryBean spitterService() {
    RmiProxyFactoryBean proxyFactoryBean = new RmiProxyFactoryBean();
    proxyFactoryBean.setServiceUrl("rmi://localhost/SpitterService");
    proxyFactoryBean.setServiceInterface(SpitterService.class);
    return proxyFactoryBean;
}
```

服务的url通过setServiceUrl属性来设置。服务提供的接口由setServiceInterface方法指定

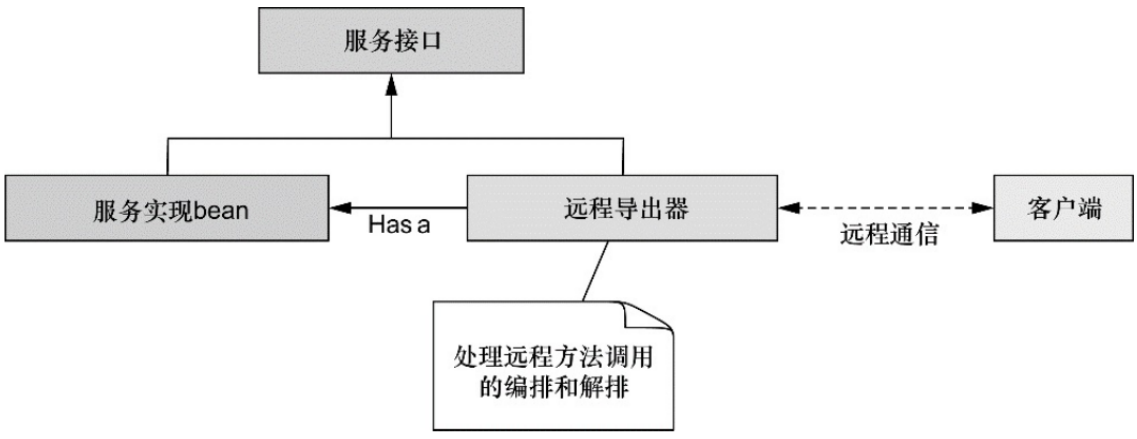


图15.3 使用远程导出器将Spring管理的bean发布为远程服务

RMI限制

- **RMI很难穿过防火墙，因为RMI使用任意端口来交互，防火墙不允许**
- **RMI是基于Java的。客户端和服务端必须都使用Java开发，因RMI使用了Java序列化机制**

3 使用Hessian和Burlap发布远程服务

Hessian和Burlap基于HTTP的轻量级远程服务解决方案。借助于京可能简单的API和通讯协议，都致力于简化Web服务。

- Hessian 使用二进制消息进行客户端和服务端的交互。他的二进制消息可以移植到其他非Java语言
- Burlap 基于XML的远程调用技术，可以很自然地一直到任何能过够计息XML的语言上

3.1 使用Hessian和Burlap导出bean

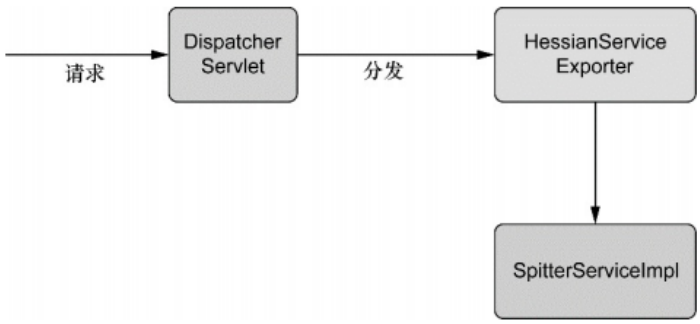
3.1.1到处Hessian服务

HessianServiceExporter对Hessian服务所执行的功能与RmiServiceExporter对RMI服务所执行的功能是相同的：他把PRJO的public方法发不成Hessian服务的方法

```
@Configuration
public class HessianConfig {

    @Bean
    public HessianServiceExporter hessianServiceExporter(SpitterService
spitterService) {
        HessianServiceExporter serviceExporter = new HessianServiceExporter();
        serviceExporter.setService(spitterService);
        serviceExporter.setServiceInterface(SpitterService.class);
        return serviceExporter;
    }

    @Bean
    public SpitterService spitterService() {
        return new SpitterServiceImpl();
    }
}
```



HessianServiceExporter是一个Spring MVC控制器，它可以接收Hessian请求，并把这些请求转换成对POJO的调用从而将POJO导出为一个Hessian服务

与RmiServiceExporter对RMI服务不同的是，不需要设置serviceName属性。因为Hessian没有注册表，没有必要对Hessian服务命名。

3.1.2 配置Hessian控制器

由于Hessian是基于HTTP的，所以HessianServiceExplorter实现为一个Spring MVC控制器。意味着为了到处的Hessian服务，需要额外的配置：

- 再web.xml中配置SpringDispatcherServlet，并把应用部署为Web应用
- 再Spring的配置文件中配置一个URL处理器，把Hessian服务的URL分发给对应的Hessian服务bean

.

.

由于Hessian和Burlap采用私有化的序列化机制，RMI使用了Java本身的序列化机制。

4 使用Spring的HttpInvoker

基于HTTP提供了RPC(像Hessian/Burlap)，同时又使用了Java的对象序列化机制(像RMI一样)

4.1 将bean导出为HTTP服务

- 将bean到处为RMI服务，需要使用RmiServiceExporter
- 将bean导出为Hessian服务，需要使用HessianServiceExplorter
- 将bean导出为Burlap服务，需要使用BurlapServiceExplorter
- 导出Http invoker服务，需要是使用HttpInvokerServiceExplorter

需要使用HttpInvokerServiceExporter

```
@Bean
public HttpInvokerServiceExporter serviceExporter(SpitterService spitterService)
{
    HttpInvokerServiceExporter serviceExporter = new
HttpInvokerServiceExporter();
    serviceExporter.setService(spitterService);
    serviceExporter.setServiceInterface(SpitterService.class);
    return serviceExporter;
}
```

HttpInvokerServiceExporter也是一个Spring的MVC控制器，通过DispatcherServlet接受来自于客户端的请求。并将这些请求转换成对实现服务的POJO的方法调用。同时需要建立一个URL处理器，映射HTTP URL到对应的服务

![(C:\Users\wanxin\AppData\Roaming\Typora\typora-user-images\image-20200702095017436.png)

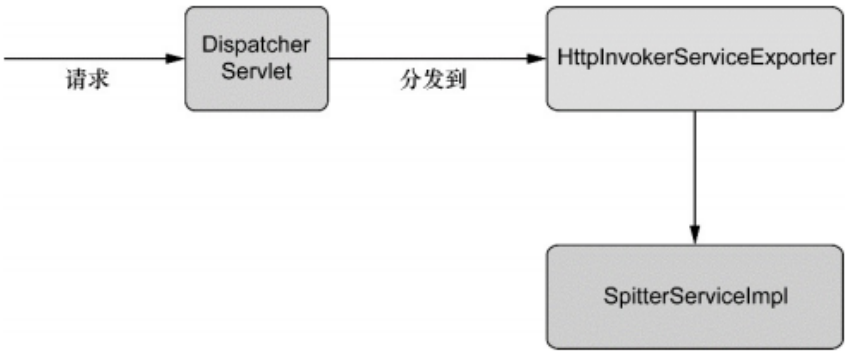


图15.8 HttpInvokerServiceExporter工作方式与Hessian和Burlap很相似，通过Spring MVC的DispatcherServlet接收请求，并将这些请求转换成对Spring bean的方法调用

为了把基于HTTp invoker的远程服务装配到客户端的应员工上下文中，需要将HttpInvokerProxyFactoryBean 配置为一个bean来代理它

```
public class HttpClientConfig {
    @Bean
    public HttpInvokerProxyFactoryBean spitterSrevice()
    {
        HttpInvokerProxyFactoryBean proxyFactoryBean = new
HttpInvokerProxyFactoryBean();
        proxyFactoryBean.setServiceInterface(SpitterService.class);

 proxyFactoryBean.setServiceUrl("http://localhost:8080/Spitter/spitter.service")
;
        return proxyFactoryBean;
    }
}
```

setServiceInterface用来标识Spitter服务所实现的接口，而serviceUrl属性任然用来标识远程服务的位置。因为HTTP invoker是基于HTTP的，如同Hessian和Burlap一样，serviceUrl可以包含与Hessian和Burlap版本中的bean一样的URL

Spirng的HTTP Invoker是作为两全其美的远程调用解决方案而出现的，把HTTP的简单性和Java内置的秀爱给你序列化机制融合在一起。使得HTTP invoker服务成为一个引人注目的替代RMI或者Hessian/Burlap的可选方案。

HTTP invoker 只是一个Spring框架所提供的远程调用解决方案，意味着客户端和服务端必须都是Spring应用

5 发布和使用Web服务

SOA(面向服务的架构)的核心理念是，应用程序可以并且影噶被设计成依赖于一组公共的核心服务，而不是每个应用都重新实际西安相同的功能。

5.1 创建基于Spirng的JAX-WS端点

JAX-WS(Java API for XML service)。在之前内容是一个弄Spring的服务到处器创建了远程服务。这些服务导出器将Spring配置的POJO转换成了远程服务。

Spring提供了JAX-WS服务导出器，SimpleJaxWsServiceExporter，但是它不是所有场景下的最好选择。SimpleJaxWsServiceExploter要求JAX-WS运行时支持将断电发布到指定地址上。

如果我们要部署的JAX-WS运行时不支持将其发布到指定地址上，就需要以传统的方式来编写JAX-WS端点。意味着端点的生命周期由JAX-WS运行时管理，而不是Spring。但这并不意味着他们不能装配Spring上下文中的bean。

5.1.1 在Spring中自动装配JAX-WS端点

JAX-WS编程模型是哟共注解将类和类方法声明为Web服务的操作。使用@WebService注解所标注的类被认为Web服务的端点，而使用@WebMethod注解所标注的方法被认为时操作

JAX-WS端点可能需要依赖注入，但是端点的生命周期由JAX-WS运行时管理而不是Spring来管理，这似乎不能把Spring管理额bean装配进JAX-WS管理的端点实例中。

装配JAX-WS端点的秘密在于继承SpringBeanAutowiringSupport。通过继承SpringBeanAutowiringSupport，则可以使用@Autowired注解标注端点的属性，一来就会自动注入。

```
@WebService(serviceName = "SpitterService")
public class SpitterBeanAutowiringSupport extends SpringBeanAutowiringSupport {

    @Autowired
```

```
    SpitterService spitterService;

    @WebMethod
    public void addSpittler(Spittle spittle) {
        spitterService.saveSpittle(spittle);
    }

    @WebMethod
    public void deleteSpittler(long spittleId) {
        spitterService.deleteSpittle(spittleId);
    }

    @WebMethod
    public List<Spittle> getRecentSpittles(int spittleCount) {
        return spitterService.getRecentSpittles(spittleCount);
    }

    @WebMethod
    public List<Spittle> getSpittlesForSpitter(Spitter spitter)
    {
        return spitterService.getSpittlesForSpitter(spitter);
    }
}
```

5.1.2 导出独立的JAX-WS端点

当对象的生命周期不是有Spring管理的，而对象的属性有需要注入Spring所管理的bean时，SpringBeanAutowiringSupport很有用。在何时的场景下，还是可以把Spring 管理的bean导出为JAX-WS端点的。

SpringSimpleJaxWsServiceExporter,把Spring管理的bean发布为JAX-WS运行时中的服务端点，与其他服务导出器不同，它不需要为他指定一个被导出bean的应员工，它会将使用JAX-WS注解所标注的所有bean发布为JAX-WS服务

```
@Bean
public SimpleJaxWsServiceExporter jaxWsServiceExporter() {
    return new SimpleJaxWsServiceExporter();
}
```

SimpleJaxWsServiceExporter不需要再做其他的事情就可以完成所有工作。启动的时候，他会搜索Spring应用上下文来查找所有使用@WebService注解的Bean。当找到符合的Bean时，SimpleJaxWsServiceExporter使用http://localhost:8080/地址将bean发布为JAX-WS端点。

```
@Component
@WebService(serviceName = "SpitterService")
public class SpitterServiceEndpoint {

    @Autowired
    SpitterService spitterService;

    @WebMethod
    public void addSpittler(Spittle spittle) {
        spitterService.saveSpittle(spittle);
    }

    @WebMethod
```

```
    public void deleteSpittler(long spittleId) {
        spitterService.deleteSpittle(spittleId);
    }

    @WebMethod
    public List<Spittle> getRecentSpittles(int spittleCount) {
        return spitterService.getRecentSpittles(spittleCount);
    }

    @WebMethod
    public List<Spittle> getSpittlesForSpitter(Spitter spitter) {
        return spitterService.getSpittlesForSpitter(spitter);
    }
}
```

SpitterServiceEndpoint的默认地址为http://localhost:8080/而SpitterServiceEndpoint使用了@WebService(serviceName = "SpitterService")，所以这两个bean形成的Web服务地址均为

http://localhost:8080/SpitterService。同时URL可以调整:

```
@Bean
public SimpleJaxWsServiceExporter jaxWsServiceExporter() {
    SimpleJaxWsServiceExporter exporter = new SimpleJaxWsServiceExporter();
    exporter.setBaseAddress("http://locolhost:8888/services/");
    return exporter;
}
```

5.2 在客户端代理JAX-WS服务

使用JaxWsProxyFactoryBean，可以在Spring装中配Spitter Web 服务，与其他bean一样，JaxWsProxyFactoryBean是Spring工厂bean，它能够生成一个知道如何与SOAPWeb服务江湖的代理。因此JaxWsProxyFactoryBean让装配和制员工一个远程Web服务变成也给你了可能，就像这个远程Web服务是本地POJO一样

```
@Bean
public JaxWsPortProxyFactoryBean spitterService() throws MalformedURLException {
    JaxWsPortProxyFactoryBean proxyFactoryBean = new
JaxWsPortProxyFactoryBean();
    proxyFactoryBean.setWsdlDocumentUrl(new
URL("http:localhost:8080/Services/SpitterService?wsdl"));
    proxyFactoryBean.setServiceName("spitterService");
    proxyFactoryBean.setPortName("spitterServiceHttpPort");
    proxyFactoryBean.setServiceInterface(SpitterService.class);
    proxyFactoryBean.setNamespaceUri("http:spitter.com");

    return proxyFactoryBean;
}
```

setWsdlDocumentUrl 标识了远程Web文件的位置。 JaxWsPortProxyFactoryBean 将使用这个位置上可用的WSDL来为服务创建代理。

```
<wsdl:definitions targetNamespace="http://spitter.com">
...
  <wsdl:service name="spitterService">
    <wsdl:port name="spitterServiceHttpPort"
            binding="tns:spitterServiceHttpBinding">
...
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

JaxWsPortProxyFactoryBean需要我们是哟共portName和ServiceName属性指定端口和服务名称。WSDL中wsdl:port和wsdl:service元素的name属性可以帮助我们识别出这些属性应该设置成什么

15. 使用远程服务 (Web 视图)