一、架构演进

为什么现在的系统不用单体架构，而用微服务架构。

1.单体架构

为了提高系统的吞吐量，单体架构中的垂直升级和水平扩展，存在以下几个问题

- 提升的性能是有限的
- 成本过高，没有办法针对某一个具体的模块做性能的提升，因为单体，所有模块都是在一起的。
- 更新、维护成本非常高，对于系统中要修改或增加的功能，整个发布的流程非常麻烦。
- 某一个模块出现了bug，就会影响整个系统。

2.垂直应用架构

根据业务的边界，对单体应用进行垂直拆分，将一个系统拆分成多个服务。比如一个管理系统，可以拆分成权限系统、业务系统、数据系统等。

垂直应用存在的问题：

每个独立部署的服务之间，公共的部分需要部署多份。那么对公共部分的修改、部署、更新都需要重复的操作，带来比较大的成本

为了解决这一问题，接下来就进入到分布式应用架构阶段。

3.分布式应用架构阶段

在这个阶段里，将服务内部公用的模块抽取出来，部署成一个独立的服务，那么需要解决服务之间的高效通信问题。

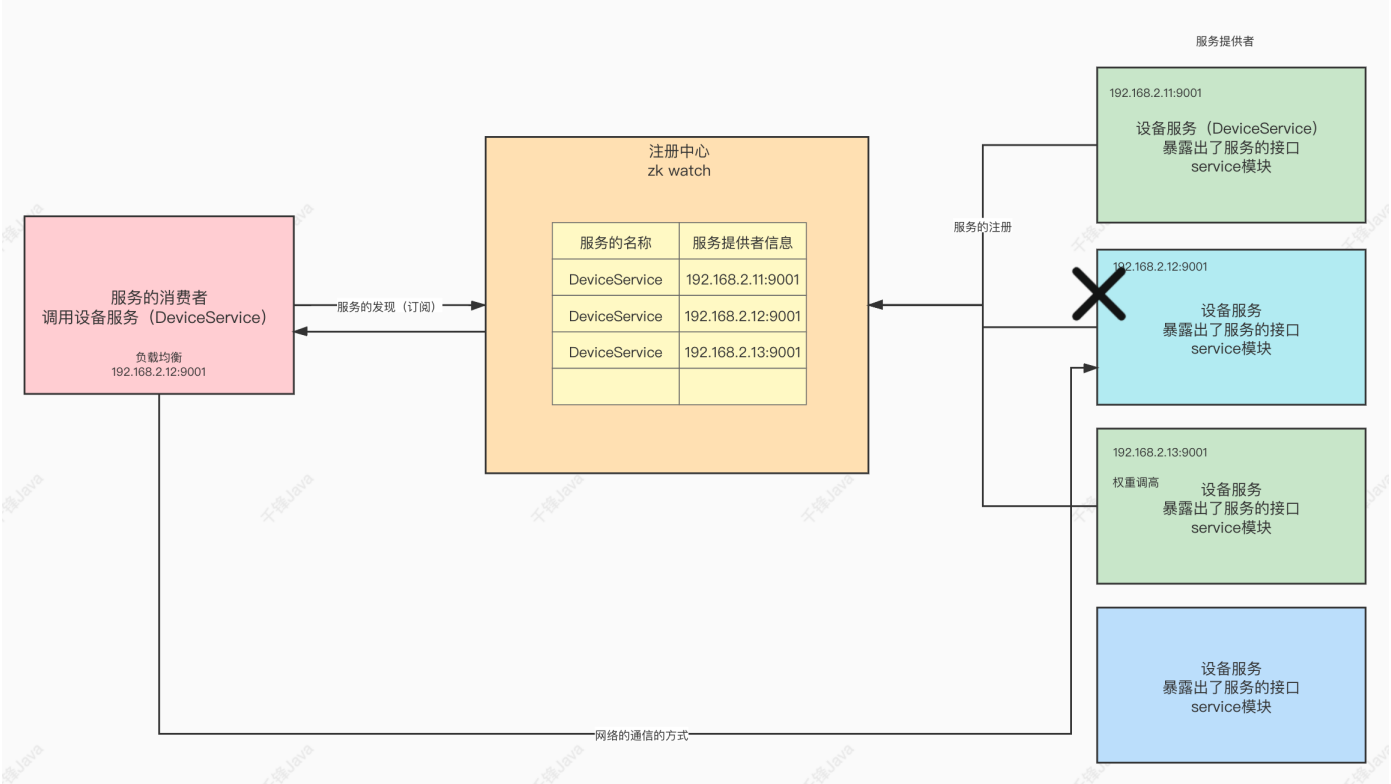
但是分布式应用架构阶段会存在新的问题：

- 服务越来越多，这么多服务如何被发现?
- 服务越来越多，服务如何被治理?
- 服务之间如何实现高效通信?

4.微服务架构阶段

微服务架构阶段主要解决的几个问题：

- 服务的注册和发现：这么多服务如何注册。这么多服务如何被发现
- 服务之间的高效调用：使用rpc或者http进行通信
- 服务治理：服务权重、负载均衡、服务熔断、限流等等一些服务治理方面的问题



二、注册中心

1.概述

注册中心的选择有很多：

- 自研：redis
- zookeeper（dubbo的推荐）：zk是一个分布式服务组件中的一个非常重要的组件，里面涉及到很多优秀的分布式设计思想，堪称鼻祖地位。
- nacos：nacos既可以作为注册中心使用，也可以作为分布式配置中心使用
- eureka：eureka是spring cloud netflix框架中著名的注册中心，里面的服务的续约、心跳等等的设计非常的经典。

2.搭建zookeeper注册中心

- 克隆一个虚拟机
- 安装jdk
- 解压zk的压缩包
- 进入到conf文件夹内，重命名：zoo_samples.cfg->zoo.cfg
- 进入到bin中，使用命令来操作zk

```
./zkServer.sh start # 启动zk
./zkServer.sh status # 查看zk状态，如果状态是：Mode: standalone 表示启动成功
./zkServer.sh stop # 关闭zk
```

三、RPC及Dubbo

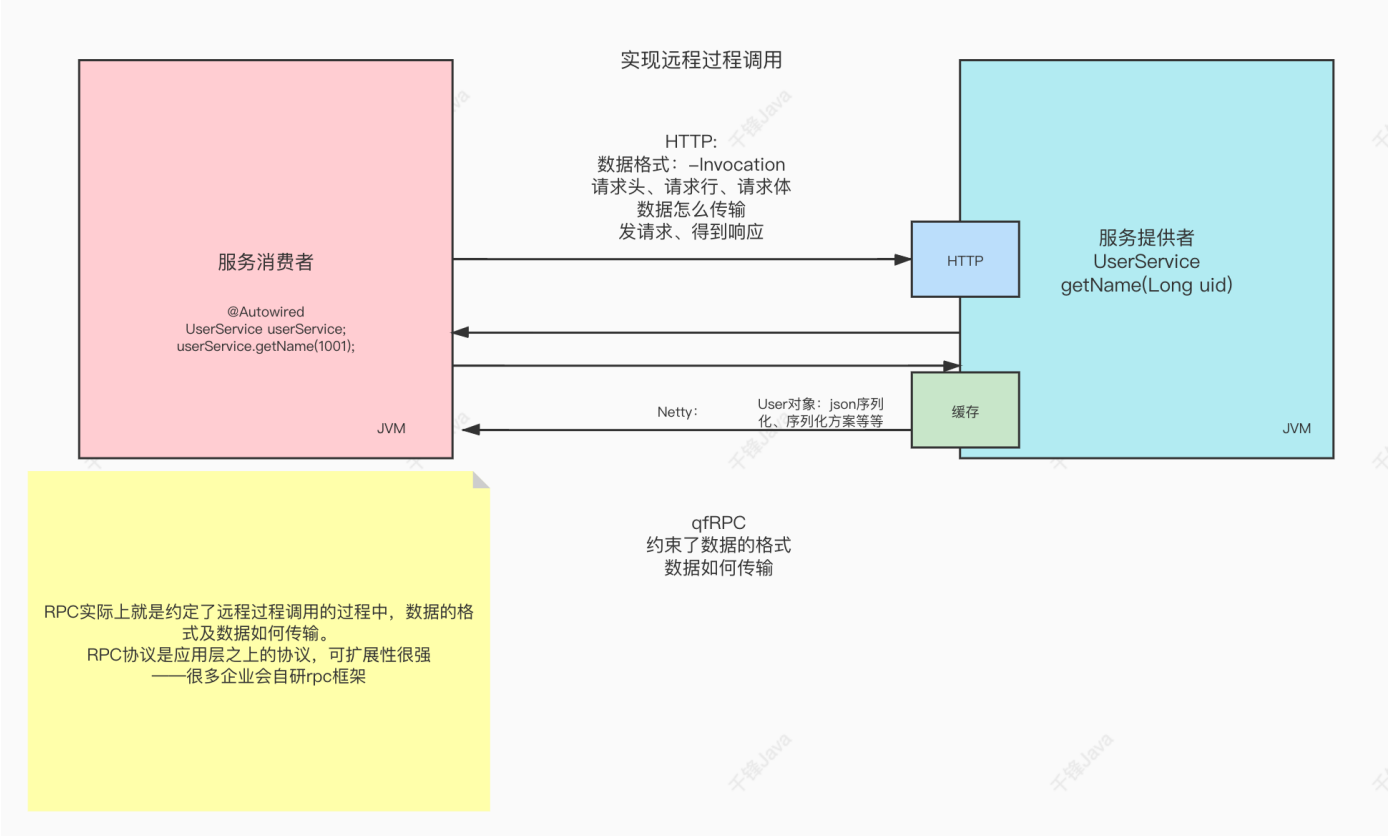
1.什么是RPC

dubbo是一款高性能的rpc框架。什么是rpc呢?

rpc是一种协议：是一种远程过程调用（remote procudure call）协议

rpc协议是在应用层之上的协议，规定了通信双方进行通信的数据格式是什么样的，及数据如何传输：

- 指明调用的类或接口
- 指明调用的方法及参数



2.手写RPC项目

参考《dubbo-init-demo》项目

3.什么是dubbo

Apache Dubbo 是一款高性能、轻量级的开源服务框架。

Apache Dubbo |ˈdʌbəʊ| 提供了六大核心能力：面向接口代理的高性能RPC调用，智能容错和负载均衡，服务自动注册和发现，高度可扩展能力，运行期流量调度，可视化的服务治理与运维。

4.dubbo怎么实现远程通信?

服务消费者去注册中心订阅到服务提供者的信息。然后通过dubbo进行远程调用。

5.dubbo初体验

1) 创建接口层项目

直接创建了一个项目。项目里有一个接口。接口中定义了一个服务的内容。

```
public interface SiteService {
  String getName(String name);
}

```

2）创建服务提供者

- 创建一个服务提供者项目。引入依赖

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <parent>
    <artifactId>spring-dubbo-demo</artifactId>
    <groupId>com.qf</groupId>
    <version>1.0-SNAPSHOT</version>
  </parent>
  <modelVersion>4.0.0</modelVersion>

  <artifactId>dubbo-demo-site-provider</artifactId>

  <properties>
    <maven.compiler.source>8</maven.compiler.source>
    <maven.compiler.target>8</maven.compiler.target>
  </properties>

  <dependencies>

```

```
21      <dependency>
22        <groupId>org.slf4j</groupId>
23        <artifactId>slf4j-api</artifactId>
24        <version>1.8.0-alpha2</version>
25      </dependency>
26
27      <!--dubbo-->
28      <dependency>
29        <groupId>org.apache.dubbo</groupId>
30        <artifactId>dubbo</artifactId>
31        <version>2.7.3</version>
32      </dependency>
33      <!--zk-->
34      <dependency>
35        <groupId>org.apache.curator</groupId>
36        <artifactId>curator-framework</artifactId>
37        <version>4.1.0</version>
38      </dependency>
39      <dependency>
40        <groupId>org.apache.curator</groupId>
41        <artifactId>curator-client</artifactId>
42        <version>4.1.0</version>
43      </dependency>
44      <dependency>
45        <groupId>org.apache.curator</groupId>
46        <artifactId>curator-recipes</artifactId>
47        <version>4.1.0</version>
48      </dependency>
49      <dependency>
50        <groupId>org.apache.zookeeper</groupId>
51        <artifactId>zookeeper</artifactId>
52        <version>3.4.13</version>
53      </dependency>
54      <!--接口层-->
55      <dependency>
56        <groupId>com.qf</groupId>
57        <artifactId>dubbo-demo-site-api</artifactId>
58        <version>1.0-SNAPSHOT</version>
59      </dependency>
60    </dependencies>
61
62  </project>
63
```

```
64
```

- 编写具体的提供服务的实现类

```
1  package com.qf.provider.service.impl;
2
3  import com.qf.api.SiteService;
4  //要把这个服务交给dubbo容器-》在项目中整合dubbo
5  public class SiteServiceImpl implements SiteService {
6    @Override
7    public String getName(String name) {
8      return "name:"+name;
9    }
10 }
11
```

- 编写bean配置文件，将dubbo和spring ioc整合，把服务提供到dubbo中

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <beans xmlns="http://www.springframework.org/schema/beans"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:dubbo="http://code.alibabatech.com/schema/dubbo"
4         xsi:schemaLocation="http://www.springframework.org/schema/beans
   http://www.springframework.org/schema/beans/spring-beans.xsd
   http://code.alibabatech.com/schema/dubbo
   http://code.alibabatech.com/schema/dubbo/dubbo.xsd">
5
6    <!--服务名称-->
7    <dubbo:application name="site-service"/>
8    <!--注册中心的信息：服务要注册到这个注册中心上-->
9    <dubbo:registry address="zookeeper://172.16.253.35:2181"/>
10   <!--配置当前这个服务在dubbo容器中的端口号，每个dubbo容器内部的服务的端口号必须是
   不一样的-->
11   <dubbo:protocol name="dubbo" port="20881"/>
12   <!--暴露出SiteService服务，指明该服务具体的实现bean是siteService-->
13   <dubbo:service interface="com.qf.api.SiteService" ref="siteService"/>
14   <!--将服务提供者的bean注入到ioc容器中-->
15   <bean id="siteService"
   class="com.qf.provider.service.impl.SiteServiceImpl"/>
16
17 </beans>
18
```

- 启动ioc容器，关联bean配置文件

```
public class Provider {

  public static void main(String[] args) throws IOException {
    ClassPathXmlApplicationContext context = new
ClassPathXmlApplicationContext(new String[]{"provider.xml"});
    context.start();
    System.in.read(); // 让当前服务一直在线，不会被关闭，按任意键退出
  }

}

```

3）创建服务消费者

- 引入依赖

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <parent>
    <artifactId>spring-dubbo-demo</artifactId>
    <groupId>com.qf</groupId>
    <version>1.0-SNAPSHOT</version>
  </parent>
  <modelVersion>4.0.0</modelVersion>

  <artifactId>dubbo-demo-site-consumer</artifactId>

  <properties>
    <maven.compiler.source>8</maven.compiler.source>
    <maven.compiler.target>8</maven.compiler.target>
  </properties>

  <dependencies>

    <dependency>
```

```
22        <groupId>org.slf4j</groupId>
23        <artifactId>slf4j-api</artifactId>
24        <version>1.8.0-alpha2</version>
25      </dependency>
26
27      <dependency>
28        <groupId>com.qf</groupId>
29        <artifactId>dubbo-demo-site-api</artifactId>
30        <version>1.0-SNAPSHOT</version>
31      </dependency>
32
33      <dependency>
34        <groupId>org.apache.dubbo</groupId>
35        <artifactId>dubbo</artifactId>
36        <version>2.7.3</version>
37      </dependency>
38
39      <!--zk-->
40      <dependency>
41        <groupId>org.apache.curator</groupId>
42        <artifactId>curator-framework</artifactId>
43        <version>4.1.0</version>
44      </dependency>
45      <dependency>
46        <groupId>org.apache.curator</groupId>
47        <artifactId>curator-client</artifactId>
48        <version>4.1.0</version>
49      </dependency>
50      <dependency>
51        <groupId>org.apache.curator</groupId>
52        <artifactId>curator-recipes</artifactId>
53        <version>4.1.0</version>
54      </dependency>
55      <dependency>
56        <groupId>org.apache.zookeeper</groupId>
57        <artifactId>zookeeper</artifactId>
58        <version>3.4.13</version>
59      </dependency>
60
61    </dependencies>
62
63  </project>
64
```

- 编写bean的配置文件

```
<?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:dubbo="http://code.alibabatech.com/schema/dubbo"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://code.alibabatech.com/schema/dubbo
http://code.alibabatech.com/schema/dubbo/dubbo.xsd">

  <dubbo:application name="site-consumer"/>

  <dubbo:registry address="zookeeper://172.16.253.35:2181"/>

  <!--在消费者中，需要调用的dubbo中的哪个服务，siteService-
>com.qf.api.SiteService-->
  <dubbo:reference interface="com.qf.api.SiteService"
id="siteService"/>
</beans>

```

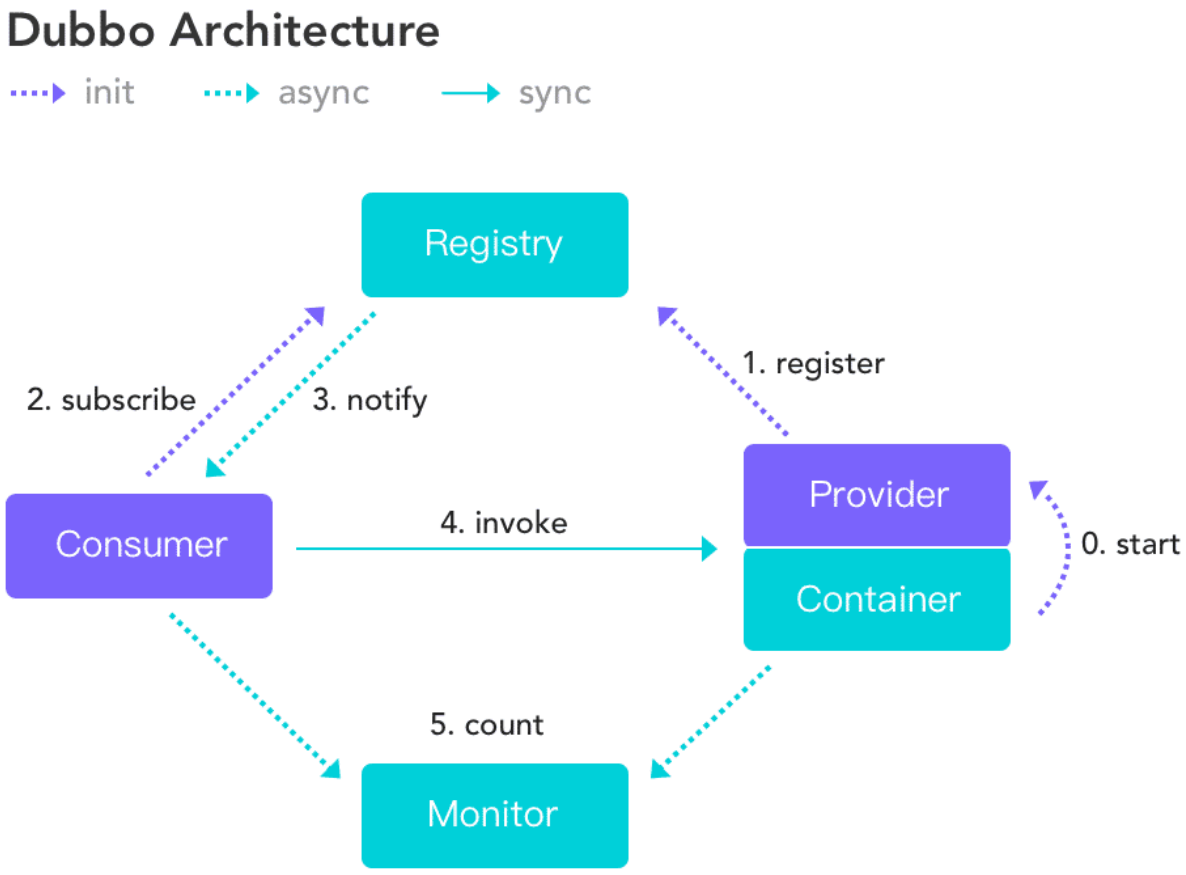
- 启动消费者，调用服务提供者

```
package com.qf.site.consumer;

import com.qf.api.SiteService;
import
org.springframework.context.support.ClassPathXmlApplicationContext;
/**
 * @author Thor
 * @公众号 Java架构栈
 */
public class Consumer {


  public static void main(String[] args) {
    ClassPathXmlApplicationContext context = new
ClassPathXmlApplicationContext(new String[] {"consumer.xml"});
    context.start();
    /*
```

```
16      下面这一整个过程。都是在执行远程过程调用—— rpc remote produce call 服务框架
17       */
18      //获取一个代理，代理服务提供者内提供的bean
19      SiteService service = (SiteService)context.getBean("siteService");
// 获取远程服务代理
20      //调用代理对象的getName方法。通过代理对象调到服务提供者内的bean
21      String result = service.getName("hellodubbo");
22      System.out.println(result);
23
24
25    }
26  }
27
```

服务代理的过程

6.dubbo内部结构



- dubbo提供了一个容器用来存放服务提供者（初始化）
- 服务提供者将服务名、及具体的服务地址、端口等信息注册到注册中心上（初始化）
- 服务消费者订阅需要的服务（初始化）
- 注册中心异步通知服务的变更情况
- 服务消费者同步的调用到服务提供者的服务
- 监控中心实时监控和治理当前的服务

注意：

- 同步：好比打电话，双方必须在线，才能完成
- 异步：好比发微信语音，上游发完就结束了，不需要等待对方执行完。

四、Springboot中使用dubbo

springboot中使用dubbo也是一样，需要建立接口层、服务提供者、服务消费者。

1.创建接口层

```java
package com.qf.api;

import com.qf.entity.Site;
/**
 * @author Thor
 * @公众号 Java架构栈
 */
public interface SiteService {
  Site getSiteById(Long id);
}

```

2.创建服务提供者

- 引入依赖

```xml
<!-- Dubbo Spring Boot Starter -->
    <dependency>
      <groupId>org.apache.dubbo</groupId>
      <artifactId>dubbo-spring-boot-starter</artifactId>
      <version>2.7.3</version>
    </dependency>
    <dependency>
      <groupId>org.apache.dubbo</groupId>
      <artifactId>dubbo-registry-zookeeper</artifactId>
      <version>2.7.3</version>
      <exclusions>
        <exclusion>
          <groupId>org.slf4j</groupId>
          <artifactId>slf4j-log4j12</artifactId>
        </exclusion>
      </exclusions>
    </dependency>
```

- 编写配置文件

```yaml
server:
  port: 9001
dubbo:
  application:
    name: site-service-provider
  registry:
    address: zookeeper://172.16.253.55:2181
  protocol:
    port: 20882

```

- 在服务提供者的实现类上打上注解来自于dubbo的@Service

```java
package com.qf.dubbo.site.provider.service.impl;


import com.qf.api.SiteService;
import com.qf.entity.Site;
import org.apache.dubbo.config.annotation.Service;
/**
 * @author Thor
 * @公众号 Java架构栈
 */
@Service
public class SiteServiceImpl implements SiteService {
  @Override
  public Site getSiteById(Long id) {
    Site site = new Site();
    site.setId(id);
    return site;
  }
}
```

- 在启动类上打上注解@EnableDubbo

3.服务消费者

- 引入依赖（与提供者相同）
- 编写配置文件

```yaml
server:
  port: 8001
dubbo:
  application:
    name: site-consumer
  registry:
    address: zookeeper://172.16.253.55:2181

```

- 使用@Reference注解订阅服务，注意这个注解来自于dubbo

```java
package com.qf.dubbo.site.consumer.controller;

import com.qf.api.SiteService;
import com.qf.entity.Site;
import org.apache.dubbo.config.annotation.Reference;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/site")
public class SiteController {

  @Reference
  private SiteService service;

  @GetMapping("/get/{id}")
  public Site getSiteById(@PathVariable Long id){
    return service.getSiteById(id);
  }
}

```

- 启动类上打上注解

```java
package com.qf.dubbo.site.consumer;

import org.apache.dubbo.config.spring.context.annotation.EnableDubbo;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
```

```
6
7  @SpringBootApplication
8  @EnableDubbo
9  public class DubboSiteConsumerApplication {
10
11   public static void main(String[] args) {
12     SpringApplication.run(DubboSiteConsumerApplication.class,args);
13   }
14
15 }
16
```

- 启动服务，访问接口

4.服务启动的源码剖析

六、dubbo用法示例

1.version版本号

版本号用处是对于同一个接口，具有不同的服务实现。

- 服务提供者1

```
1  @Service(version = "default")
2  public class DefaultSiteServiceImpl implements SiteService {
3      @Override
4      public String siteName(String name) {
5          return "default:"+name;
6      }
7  }
```

- 服务提供者2

```java
@Service(version = "async")
public class AsyncSiteServiceImpl implements SiteService {
    @Override
    public String siteName(String name) {
        return "async:" + name;
    }
}
```

- 服务消费者

```java
@Reference(id = "siteService",version = "async")
private SiteService siteService;
```

2.指定protocol协议

dubbo框架可以对协议进行扩展，比如使用：

- rest
- http
- dubbo
- 自定义协议

在配置文件中配置协议：

```properties
# 应用名称
spring.application.name=my-dubbo-provider
# 应用服务 WEB 访问端口
server.port=8080

# Base packages to scan Dubbo Component:
@org.apache.dubbo.config.annotation.Service==>@EnableDubbo
dubbo.scan.base-packages=com.qf.my.dubbo.provider
dubbo.application.name=${spring.application.name}


## Dubbo Registry
dubbo.registry.address=zookeeper://172.16.253.55:2181

# Dubbo Protocol
#dubbo.protocol.name=dubbo
```

```
#dubbo.protocol.port=20880

# @Path
#dubbo.protocol.name=rest
#dubbo.protocol.port=8083

dubbo.protocols.protocol1.id=rest
dubbo.protocols.protocol1.name=rest
dubbo.protocols.protocol1.port=8090
dubbo.protocols.protocol1.host=0.0.0.0

dubbo.protocols.protocol2.id=dubbo1
dubbo.protocols.protocol2.name=dubbo
dubbo.protocols.protocol2.port=20882
dubbo.protocols.protocol2.host=0.0.0.0

dubbo.protocols.protocol3.id=dubbo2
dubbo.protocols.protocol3.name=dubbo
dubbo.protocols.protocol3.port=20883
dubbo.protocols.protocol3.host=0.0.0.0
```

在暴露服务时指明要使用的协议:

```
@Service(version = "default",protocol = "protocol2")
public class DefaultSiteServiceImpl implements SiteService {
    @Override
    public String siteName(String name) {
        return "default:"+name;
    }
}
```

3.使用rest访问dubbo的服务

- 服务提供者暴露用rest协议制定的服务

```
package com.qf.my.dubbo.provider.impl;

import com.qf.site.SiteService;
```

```
import org.apache.dubbo.config.annotation.Service;
import org.apache.dubbo.rpc.protocol.rest.support.ContentType;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.QueryParam;

/**
 * @author Thor
 * @公众号 Java架构栈
 */
@Service(version = "rest", protocol = "protocol1")
@Path("site")
public class RestSiteService implements SiteService {
    @Override
    @GET
    @Path("name")
    @Produces({ContentType.APPLICATION_JSON_UTF_8,
ContentType.TEXT_PLAIN_UTF_8})
    public String siteName(@QueryParam("name") String name) {
        return "rest:" + name;
    }
}
```

- 在浏览器中使用restful调用服务

```
http://localhost:8090/site/name?name=abc
```

4.消费者通过url直连指定的服务提供者

- 配置文件中声明三个dubbo协议

```
dubbo.protocols.protocol1.id=dubbo1
dubbo.protocols.protocol1.name=dubbo
dubbo.protocols.protocol1.port=20881
dubbo.protocols.protocol1.host=0.0.0.0

dubbo.protocols.protocol2.id=dubbo2
dubbo.protocols.protocol2.name=dubbo
dubbo.protocols.protocol2.port=20882
dubbo.protocols.protocol2.host=0.0.0.0
```

```
10
11 dubbo.protocols.protocol3.id=dubbo3
12 dubbo.protocols.protocol3.name=dubbo
13 dubbo.protocols.protocol3.port=20883
14 dubbo.protocols.protocol3.host=0.0.0.0
```

- 服务提供者暴露服务，未指定协议，则会暴露三个服务，每个协议对应一个服务

```
1  /**
2   * @author Thor
3   * @公众号 Java架构栈
4   */
5  @Service(version = "default")
6  public class DefaultSiteServiceImpl implements SiteService {
7      @Override
8      public String siteName(String name) {
9          return "default:"+name;
10     }
11 }
```

- 消费者端通过url指定某一个服务

```
1  @Reference(id = "siteService",version = "default",url =
   "dubbo://127.0.0.1:20881/com.qf.site.SiteService:default")
2     private SiteService siteService;
```

5.服务超时

服务提供者和服务消费者都可以配置服务超时时间（默认时间为1秒）：

- 服务提供者的超时时间：执行该服务的超时时间。如果超时，则会打印超时日志（warn），但服务会正常执行完。

```
1  /**
2   * @author Thor
3   * @公众号 Java架构栈
4   */
5  @Service(version = "timeout", timeout = 4000)
6  public class TimeoutSiteServiceImpl implements SiteService {
7      @Override
```

```
    public String siteName(String name) {
        try {
            Thread.sleep(5000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("serving...");
        return "timeout site service:"+name;
    }
}
```

- 服务消费者的超时时间：从发起服务调用到收到服务响应的整个过程的时间。如果超时，则进行重试，重试失败抛异常

```
/**
 * @author Thor
 * @公众号 Java架构栈
 */
@EnableAutoConfiguration
public class TimeoutDubboConsumer {

    @Reference(version = "timeout", timeout = 3000)
    private SiteService siteService;


    public static void main(String[] args) {

        ConfigurableApplicationContext context =
SpringApplication.run(TimeoutDubboConsumer.class);
        SiteService siteService = (SiteService)
context.getBean(SiteService.class);
        String name = siteService.siteName("q-face");
        System.out.println(name);

    }

}

```

6.集群容错

dubbo为集群调用提供了容错方案:

- failover: (默认,推荐)

当出现失败时,会进行重试,默认重试2次,一共三次调用。但是会出现幂等性问题。

虽然会出现幂等性问题,但是依然推荐使用这种容错机制,在业务层面解决幂等性问题:

    - 方案一:把数据的业务id作为数据库的联合主键,此时业务id不能重复。

    - 方案二(推荐):使用分布式锁来解决重复消费问题。

- failfast:当出现失败时。立即报错,不进行重试。
- failsafe:失败不报错,记入日志。
- failback:失败就失败,开启定时任务 定时重发。
- forking:并行访问多个服务器,获取某一个结果既视为成功。

结论:如果使用dubbo,不推荐把重试关掉,而是在非幂等性操作的场景下,服务提供者方要做幂等性的解决方案(保证)。

7.服务降级

服务消费者通过Mock指定服务超时后执行的策略:

```
/**
 * @author Thor
 * @公众号 Java架构栈
 */
@EnableAutoConfiguration
public class MockDubboConsumer {

    @Reference(version = "timeout", timeout = 1000, mock = "fail:return timeout")
    private SiteService siteService;


    public static void main(String[] args) {
        ConfigurableApplicationContext context = SpringApplication.run(MockDubboConsumer.class);
```

```
        SiteService siteService = (SiteService)
context.getBean(SiteService.class);
        String name = siteService.siteName("q-face");
        System.out.println(name);
    }
}

```

- `mock=force:return+null` 表示消费方对该服务的方法调用都直接返回 null 值，不发起远程调用。用来屏蔽不重要服务不可用时对调用方的影响。
- 还可以改为 `mock=fail:return+null` 表示消费方对该服务的方法调用在失败后，再返回 null 值，不抛异常。用来容忍不重要服务不稳定时对调用方的影响。

8.本地存根

远程服务后，客户端通常只剩下接口，而实现全在服务器端，但提供方有些时候想在客户端也执行部分逻辑，比如：做 ThreadLocal 缓存，提前验证参数，调用失败后伪造容错数据等等，此时就需要在 API 中带上 Stub，客户端生成 Proxy 实例，会把 Proxy 通过构造函数传给 Stub 1，然后把 Stub 暴露给用户，Stub 可以决定要不要去调 Proxy。

- 服务提供者的接口包下创建：

```
package com.qf.site;

import com.qf.site.SiteService;

/**
 * @author Thor
 * @公众号 Java架构栈
 */
public class SiteServiceStub implements SiteService {

    private final SiteService siteService;

    public SiteServiceStub(SiteService siteService) {
        this.siteService = siteService;
    }

    @Override
    public String siteName(String name) {
```

```
        try {
            return siteService.siteName(name);
        } catch (Exception e) {
            return "stub:"+name;
        }
    }
}

```

- 服务消费者调用服务，开启本地存根

```
package com.qf.my.dubbo.consumer;

import com.qf.site.SiteService;
import org.apache.dubbo.config.annotation.Reference;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
import org.springframework.context.ConfigurableApplicationContext;

/**
 * @author Thor
 * @公众号 Java架构栈
 */
@EnableAutoConfiguration
public class StubDubboConsumer {

    @Reference(version = "timeout", timeout = 1000, stub = "true")
    private SiteService siteService;

    public static void main(String[] args) {
        ConfigurableApplicationContext context =
SpringApplication.run(StubDubboConsumer.class);
        SiteService siteService = (SiteService)
context.getBean(SiteService.class);
        String name = siteService.siteName("q-face");
        System.out.println(name);
    }
}

```

9.参数回调

参数回调方式与调用本地 callback 或 listener 相同，只需要在 Spring 的配置文件中声明哪个参数是 callback 类型即可。Dubbo 将基于长连接生成反向代理，这样就可以从服务器端调用客户端逻辑。

简而言之，就是服务端可以调用客户端的逻辑。

- 接口层

```
public interface SiteService {
    //同步调用方法
    String siteName(String name);

    //回调方法
    default String siteName(String name, String key, SiteServiceListener siteServiceListener){
        return null;
    }
}
```

- 服务提供者

```
package com.qf.my.dubbo.provider.impl;

import com.qf.site.SiteService;
import com.qf.site.SiteServiceListener;
import org.apache.dubbo.config.annotation.Argument;
import org.apache.dubbo.config.annotation.Method;
import org.apache.dubbo.config.annotation.Service;

/**
 * @author Thor
 * @公众号 Java架构栈
 */
@Service(version = "callback", methods = {@Method(name = "siteName", arguments = {@Argument(index = 2, callback = true)})}, callbacks = 3)
public class CallbackSiteServiceImpl implements SiteService {
    @Override
    public String siteName(String name) {
        return null;
    }
```

```
19
20      @Override
21      public String siteName(String name, String key, SiteServiceListener
    siteServiceListener) {
22          siteServiceListener.changed("provider data");
23          return "callback:"+name;
24      }
25  }
26
```

- 创建回调接口

```
1   package com.qf.site;
2
3   /**
4    * @author Thor
5    * @公众号 Java架构栈
6    */
7   public interface SiteServiceListener {
8       void changed(String data);
9   }
```

- 创建回调接口的实现类

```
1   package com.qf.site;
2
3   import com.qf.site.SiteServiceListener;
4
5   import java.io.Serializable;
6
7   /**
8    * @author Thor
9    * @公众号 Java架构栈
10   */
11  public class SiteServiceListenerImpl implements SiteServiceListener,
    Serializable {
12      @Override
13      public void changed(String data) {
14          System.out.println("changed:" + data);
15      }
16  }
17
```

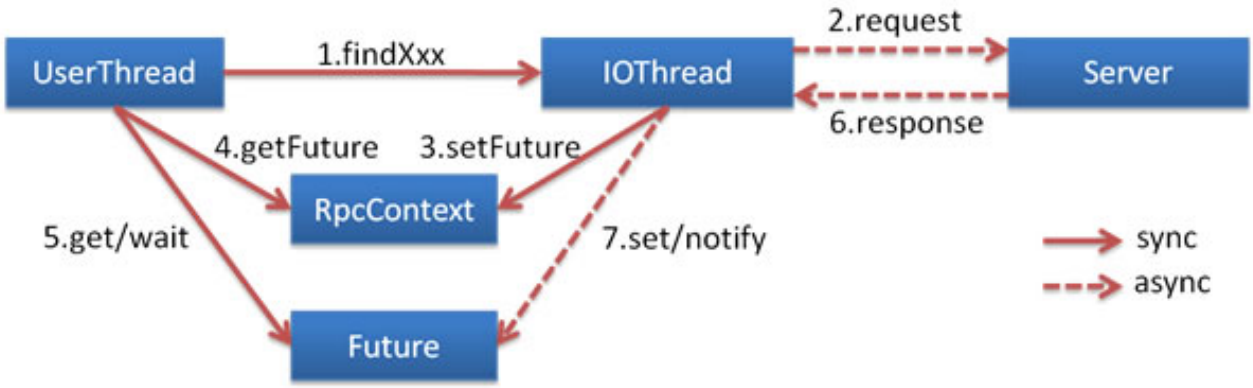
- 创建服务消费者

```
package com.qf.my.dubbo.consumer;

import com.qf.site.SiteService;
import com.qf.site.SiteServiceListenerImpl;
import org.apache.dubbo.config.annotation.Reference;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
import org.springframework.context.ConfigurableApplicationContext;

/**
 * @author Thor
 * @公众号 Java架构栈
 */
@EnableAutoConfiguration
public class CallbackDubboConsumer {

    @Reference(version = "callback")
    private SiteService siteService;


    public static void main(String[] args) {

        ConfigurableApplicationContext context =
SpringApplication.run(CallbackDubboConsumer.class);
        SiteService siteService = (SiteService)
context.getBean(SiteService.class);
        // key 目的是指明实现类在服务提供者和消费者之间保证是同一个
        System.out.println(siteService.siteName("q-face", "c1", new
SiteServiceListenerImpl()));
        System.out.println(siteService.siteName("q-face", "c2", new
SiteServiceListenerImpl()));
        System.out.println(siteService.siteName("q-face", "c3", new
SiteServiceListenerImpl()));

    }
}

```

10.异步调用

从 2.7.0 开始，Dubbo 的所有异步编程接口开始以 CompletableFuture 为基础

基于 NIO 的非阻塞实现并行调用，客户端不需要启动多线程即可完成并行调用多个远程服务，相对多线程开销较小。



简而言之，消费者通过异步调用，不用等待服务提供者返回结果就立即完成任务，待有结果后再执行之前设定好的监听逻辑。

- 接口层

```
package com.qf.site;

import java.util.concurrent.CompletableFuture;

/**
 * @author Thor
 * @公众号 Java架构栈
 */
public interface SiteService {
    //同步调用方法
    String siteName(String name);

    //回调方法
    default String siteName(String name, String key,
SiteServiceListener siteServiceListener){
        return null;
    }

    //异步调用方法
    default CompletableFuture<String> siteNameAsync(String name){
```

```
        return null;
    }
}

```

- 服务提供者

```
package com.qf.my.dubbo.provider.impl;

import com.qf.site.SiteService;
import org.apache.dubbo.config.annotation.Service;

import java.util.concurrent.CompletableFuture;

/**
 * @author Thor
 * @公众号 Java架构栈
 */
@Service(version = "async")
public class AsyncSiteServiceImpl implements SiteService {

    @Override
    public String siteName(String name) {
        return "async:" + name;
    }

    @Override
    public CompletableFuture<String> siteNameAsync(String name) {
        System.out.println("异步调用: " + name);
        return CompletableFuture.supplyAsync(() -> {
            return siteName(name);
        });
    }
}

```

- 服务消费者

```
package com.qf.my.dubbo.consumer;

import com.qf.site.SiteService;
import org.apache.dubbo.config.annotation.Reference;

```

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
import org.springframework.context.ConfigurableApplicationContext;

import java.util.concurrent.CompletableFuture;

/**
 * @author Thor
 * @公众号 Java架构栈
 */
@EnableAutoConfiguration
public class AsyncDubboConsumer {

    @Reference(version = "async")
    private SiteService siteService;


    public static void main(String[] args) {

        ConfigurableApplicationContext context = SpringApplication.run(AsyncDubboConsumer.class);
        SiteService siteService = (SiteService) context.getBean(SiteService.class);
        //调用异步方法
        CompletableFuture<String> future = siteService.siteNameAsync("q-face");
        //设置监听，非阻塞
        future.whenComplete((v, e) -> {
            if (e != null) {
                e.printStackTrace();
            } else {
                System.out.println("result:" + v);
            }
        });
        System.out.println("异步调用结束");

    }
}

```

七、dubbo的负载均衡策略

1.负载均衡策略

在上一章节中，已经涉及到dubbo的负载均衡概念：一个服务接口具有三个服务提供者。

dubbo的负载均衡是发生在服务提供者端，负载均衡策略一共有以下四种：

- 随机（默认的策略）:random
- 轮询: roundrobin
- 最小活跃调用数：leastactive
- 一致性hash: consistenthash

2.dubbo中如何配置负载均衡策略

- 服务提供者：

```
@Service(version = "default",loadbalance = "roundrobin")
public class DefaultSiteServiceImpl implements SiteService {
    @Override
    public String siteName(String name) {
        return "default:"+name;
    }
}
```

- 服务消费者：如果两边都配置了负载均衡策略，则以消费者端为准。

```

@EnableAutoConfiguration
public class DefaultDubboConsumer {

    @Reference(version = "default", loadbalance = "roundrobin")
    private SiteService siteService;


    public static void main(String[] args) {

        ConfigurableApplicationContext context =
SpringApplication.run(DefaultDubboConsumer.class);
```

```
        SiteService siteService = (SiteService) context.getBean(SiteService.class);
        String name = siteService.siteName("q-face");
        System.out.println(name);

    }
}

```

3.一致性hash的实现

- 服务端的实现

```
@Service(version = "default",loadbalance = "roundrobin")
public class DefaultSiteServiceImpl implements SiteService {
    @Override
    public String siteName(String name) {
        URL url = RpcContext.getContext().getUrl();
        return String.format("%s: %s, Hello, %s", url.getProtocol(), url.getPort(), name);
    }
}
```

- 消费端的实现

```
@EnableAutoConfiguration
public class LoadBalanceDubboConsumer {

    @Reference(version = "default", loadbalance = "consistenthash")
    private SiteService siteService;

    public static void main(String[] args) {

        ConfigurableApplicationContext context = SpringApplication.run(DefaultDubboConsumer.class);
        SiteService siteService = (SiteService) context.getBean(SiteService.class);
        for (int i = 0; i < 100; i++) {
            String name = siteService.siteName("q-face"+i%6);
            System.out.println(name);
```

```
        }

    }
}
```

4.最少活跃调用数的实现

最少活跃调用数：相同活跃数的随机，活跃数指调用前后计数差。使慢的提供者收到更少请求，因为越慢的提供者的调用前后计数差会越大。

在服务消费者端记录当前服务器目前被调用的数量（消费者自己维护着这个数据）。具体的执行过程如下：

- 消费者在本地缓存所有服务提供者
- 消费者在调用某一个服务时，会选择本地的所有服务提供者中，属性active值最小的那个服务提供者。
- 选定该服务提供者后，并对其active属性+1
- 开始调用该服务
- 完成调用后，对该服务提供者的active属性-1

整个过程，如果active的值越大，说明该服务提供者的响应性能越差，因此越少调用。

八、安装Dubbo admin监管平台

1.使用docker安装

```
docker run -d \
-p 8080:8080 \
-e dubbo.registry.address=zookeeper://172.16.253.55:2181 \
-e dubbo.admin.root.password=root \
-e dubbo.admin.guest.password=guest \
chenchuxin/dubbo-admin
```

2.访问

```
http://127.0.0.1:8080
```

九、Dubbo的SPI可扩展机制

1.Java的SPI（Service Provider Interface）机制

Java中提供了DriverManager、Connection、Statement接口，来约定了JDBC规范。但针对于MySQL或Oracle数据库来说，需要指明具体的驱动包，比如MySQL：

mysql-connector-java-5.7.25.jar 包中的META-INF/services下的java.sql.Driver文件，文件中指明了具体的驱动类

```
1 com.mysql.cj.jdbc.Driver
```

这样Java会读取该jar包下的该文件，那java怎么找到该文件呢？因为java程序需要该类：java.sql.Driver，所以找文件名是java.sql.Driver的文件。——相当于是加载了Driver接口的具体的实现类

2.SPI机制的缺点

文件中的所有类都会被加载且被实例化。没有办法指定某一个类来加载和实力化。此时dubbo的SPI可以解决

3.dubbo的SPI机制

dubbo自己实现了一套SPI机制来解决Java的SPI机制存在的问题。

dubbo源码中有很多的项目，每个项目被打成一个jar包。比如代码中通过@Service注解的属性protocol="c1"找到application.properties的c1协议是rest，那么就会去rest项目中找该项目中的META-INF中对应的文件，再找到指定的类来加载。

```
1 rest=org.apache.dubbo.rpc.protocol.rest.RestProtocol
```

通过这种机制，在项目中新增一个协议也非常方便：

- 项目中新增协议jar
- 在application.properties中加入协议名称

- 在新增的项目的META-INF/services文件中加入配置

1）dubbo的spi机制简单实现

META-INF/dubbo/com.qf.cat

```
black=com.qf.BlackCat
```

TestSPI:

```
ExtensionLoader<Cat> extensionLoader =
ExtensionLoader.getExtensionLoader(Cat.class);
Cat cat = extensionLoader.getExtension("black");
System.out.println(cat.getName());
```

2）体会spi的AOP效果

- 创建CatWrapper

```
package com.qf;

import javax.swing.text.Caret;

/**
 * @author Thor
 * @公众号 Java架构栈
 */
public class CatWrapper implements Cat {

    private Cat cat;

    public CatWrapper(Cat cat) {
        this.cat = cat;
    }

    @Override
    public String getName() {
        //切面效果
        System.out.println("cat wrapper");
        return cat.getName();
    }
```

```
23 }
24
```

- **META-INF/dubbo/com.qf.cat**中加入配置

```
1 com.qf.CatWrapper
```

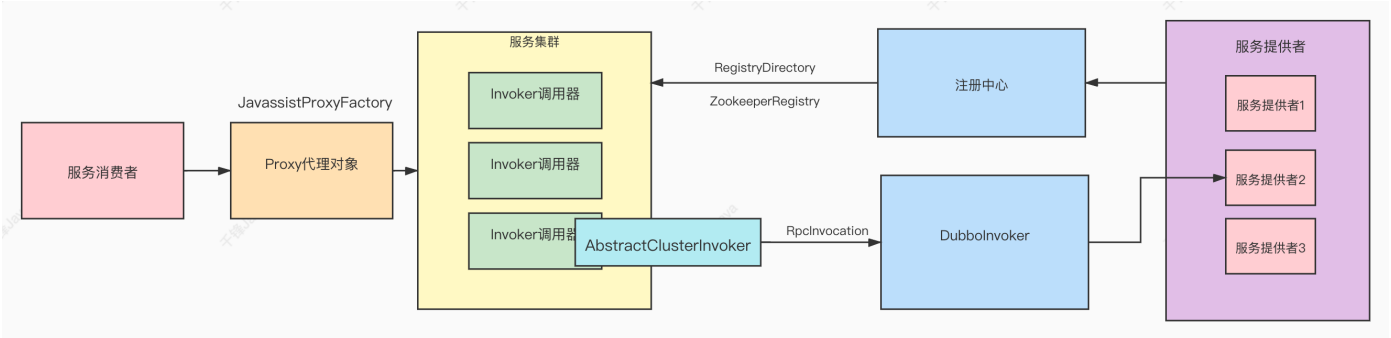
启动后发现会执行CatWrapper的getName()

3）使用dubbo的spi指定使用Http协议

```
1 //拿到的是ProtocolFilterWrapper包装类，实际被包装的是HttpProtocol
2         ExtensionLoader<Protocol> extensionLoader =
  ExtensionLoader.getExtensionLoader(Protocol.class);
3         Protocol protocol = extensionLoader.getExtension("http");
4         System.out.println(protocol);
```

十、Dubbo源码剖析

1. Dubbo服务调用过程



服务提供者的集群集群里面有几个服务提供者，就有几个invoker，invoker理解成调用一个服务提供者需要的完整的细节，封装成的对象

那集群是怎么知道有几个服务提供者——从注册中心获得，注册中心获得的数据封装在RegistryDirectory对象中。那么RegistryDirectory怎么得到注册中心中的url地址呢？必须有一个zk客户端：ZookeeperRegistry

RegistryDirectory里包含了ZookeeperRegistry，RegistryDirectory维护了所有的invoker调用器，调用器通过RegsitryDirectory（ZookeeperRegistry）的方法获得的。

AbstractClusterInvoker里包含了RegistryDirectory，换句话说，RegistryDirectory被AbstractClusterInvoker所使用。真正执行的是AbstractClusterInvoker中的invoker方法，负载均衡也在里面。

proxy是由JavassistProxyFactory生成的，拼装代码来生成的。代理对象通过JavassistProxyFactory中的InvokerInvocationHandler 生成一个代理对象，来发起对集群的调用。

InvokerInvocationHandler里封装了RpcInvocation，RpcInvocation里封装的是这一次请求所需要的所有参数。

这个invoker如果用的是dubbo协议，那么就是DubboInvoker（还有http RMI等协议）

源码中的invoker.invoke()中的invoker，如果是dubbo协议，那么就是DubboInvoker。

2.关于DubboInvoker的装饰

AsyncToSyncInvoker

异步转同步：dubbo 2.7.3 引入了InvokeMode（1.SYNC同步, 2.ASYNC异步, 3.FUTURE调用future.get()时会造成线程阻塞）

在消费者端进行调用时先判断是否是同步调用，如果是同步的话，通过asyncResult.get()获得结果。

如果是异步的话，直接返回Result对象（CompetableFuture）。

ProtocolFilterWrapper

Dubbo内容提供了大量内部实现，用来实现调用过程额外功能， 如向监控中心发送调用数据， Tps限流等等， 每个filer专注一块功能。用户同样可以通过Dubbo的SPI扩展机制现在自己的功能。

ProtocolFilterWrapper:在服务的暴露与引用的过程中构建调用过滤器链。

ListenerInvokerWrapper

dubbo在服务暴露(exporter)以及销毁暴露(unexporter)服务的过程中提供了回调窗口，供用户做业务处理。

ListenerInvokerWrapper装饰exporter, 在构造器中遍历listeners，构建export的监听链。

3. 权重轮询算法

假如目前有三台服务器A、B、C，它们的权重分别是6、2、2，那也就意味着在10次调用中，轮询的结果应该为：`AAAAAABBCC`

但如果把B和C穿插在A中，轮询的结果会更加的平滑，比如`ABACAABACA`

此时可以通过如下设计来实现：

- 每台服务器确定两个权重变量：weight、currentWeight
- weight固定不变：初始化指定了值
- currentWeight每次调整，初始化为0：currentWeight = currentWeight+weight
- 从集群中选择currentWeight最大的服务器作为选择结果。并将该最大服务器的currentWeight减去各服务器的weight总数
- 调整currentWeight = currentWeight+weight，开始新一轮的选择
- 以此往复，经过10次比较厚currentWeight都为0

请求次数	currentWeight	选择结果	选择后的currentWeight（最大的节点-权重总和）
1	6，2，2	A	-4，2，2
2	2，4，4	B	2，-6，4
3	8，-4，6	A	-2，-4，6
4	4，-2，8	C	4，-2，-2
5	10，0，0	A	0，0，0
6	6，2，2	A	-4，2，2
7	2，4，4	B	2，-6，4
8	8，-4，6	A	-2，-4，6
9	4，-2，8	C	4，-2，-2
10	10，0，0	A	0，0，0

作业

- 掌握dubbo的项目开发：创建接口层、创建服务提供者、创建服务消费者，完成简单的调用
- 编写业务：服务消费者提供工地名称，调用服务提供者（springboot）的查询接口，从数据库中查询该工地信息
- 掌握dubbo的基本用法
- 搭建dubbo的监控平台，进行手动的服务治理
- 掌握dubbo的spi案例
- 尝试去跑一跑源码，去读一读里面的设计细节

千锋教育Java教研院 关注公众号【Java架构栈】下载所有课程代码课件及工具 让技术回归本该有的纯静！