

点击查看dubbo官网

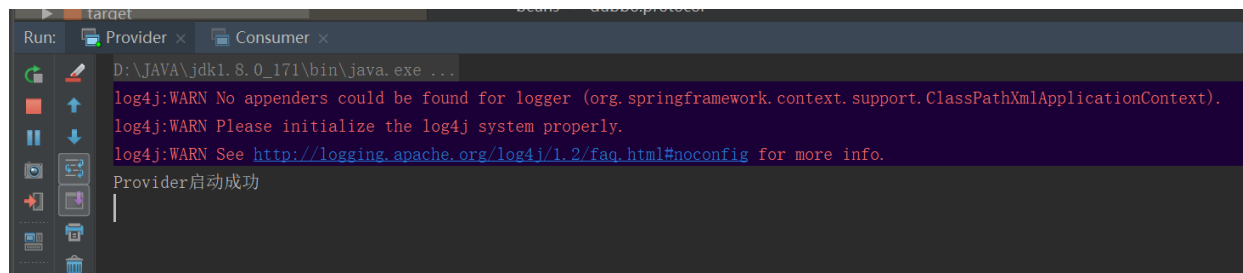
<http://dubbo.apache.org/zh-cn/docs/user/quick-start.html>

点击下载dubbo示例代码

1.项目源码：demo示例.zip

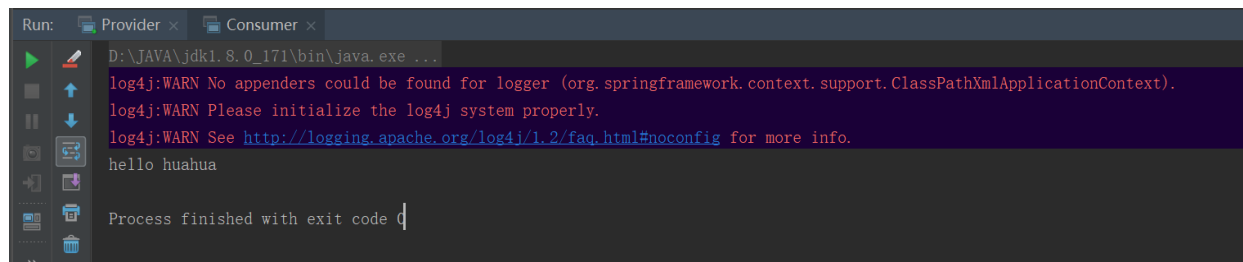
2.运行结果：

1).provider运行结果：



```
Run: Provider x Consumer x
D:\JAVA\jdk1.8.0_171\bin\java.exe ...
log4j:WARN No appenders could be found for logger (org.springframework.context.support.ClassPathXmlApplicationContext).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
Provider启动成功
```

2).consumer运行结果：



```
Run: Provider x Consumer x
D:\JAVA\jdk1.8.0_171\bin\java.exe ...
log4j:WARN No appenders could be found for logger (org.springframework.context.support.ClassPathXmlApplicationContext).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
hello huahua
Process finished with exit code 0
```

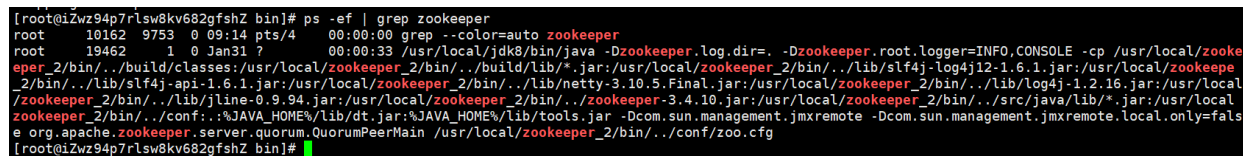
2.启动时检查 (check)

Dubbo缺省会在启动时检查依赖的服务是否可用，不可用时会抛出异常，组织Spring初始化完成，以便上线时，能及早发现问题，默认check="true".

可以通过check="false"关闭检查，比如，测试时，有些服务不挂新，或者出现了循环依赖，必须有一方先启动。

另外，如果你的Spring容器是懒加载的，或者通过API编程延迟引用服务，请关闭check,否则服务临时不可用时，会抛出异常，拿到null引用，如果check="false",总是会返回引用，当服务恢复时，能自动连上。

1.因为我配置了三台zookeeper集群，所以我关掉两台zookeeper不能正常使用，故dubbo项目启动不了，如下图：



```
[root@izwz94p7rlsw8kv682gfhZ bin]# ps -ef | grep zookeeper
root      10162  9753  0 09:14 pts/4    00:00:00 grep --color=auto zookeeper
root      19462   1  0 Jan31 ?        00:00:33 /usr/local/jdk8/bin/java -Dzookeeper.log.dir=. -Dzookeeper.root.logger=INFO,CONSOLE -cp /usr/local/zookeeper_2/bin/../build/classes:/usr/local/zookeeper_2/bin/../build/lib/*:/usr/local/zookeeper_2/bin/../lib/slf4j-log4j12-1.6.1.jar:/usr/local/zookeeper_2/bin/../lib/slf4j-api-1.6.1.jar:/usr/local/zookeeper_2/bin/../lib/netty-3.10.5.Final.jar:/usr/local/zookeeper_2/bin/../lib/log4j-1.2.16.jar:/usr/local/zookeeper_2/bin/../lib/jline-0.9.94.jar:/usr/local/zookeeper_2/bin/../zookeeper-3.4.10.jar:/usr/local/zookeeper_2/bin/../src/java/lib/*:/usr/local/zookeeper_2/bin/../conf:;%JAVA_HOME%/lib/dt.jar;%JAVA_HOME%/lib/tools.jar -Dcom.sun.management.jmxremote -Dcom.sun.management.jmxremote.local.only=false org.apache.zookeeper.server.quorum.QuorumPeerMain /usr/local/zookeeper_2/bin/../conf/zoo.cfg
[root@izwz94p7rlsw8kv682gfhZ bin]#
```

2.修改dubbo-provider.xml配置

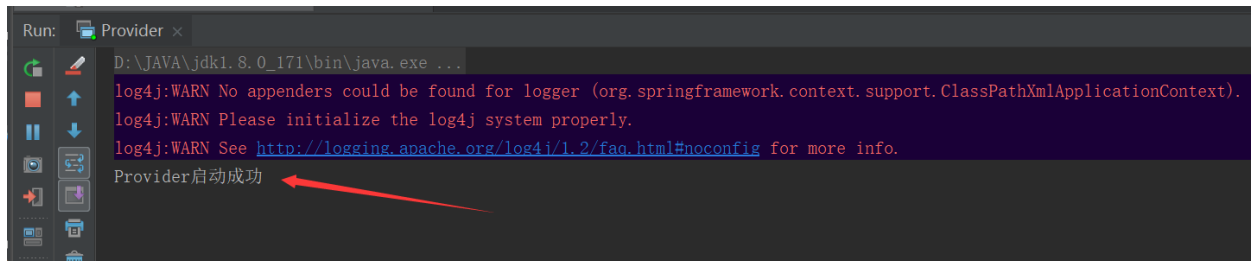
将

```
<dubbo:registry address="zookeeper://119.23.108.42:2181" />
```

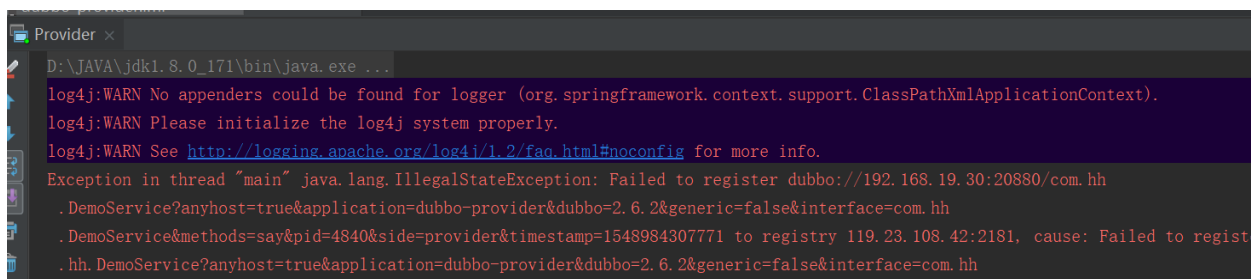
改为

```
<dubbo:registry address="zookeeper://119.23.108.42:2181" check="false" />
```

3.我直接启动Provider类，运行结果如下：



4.如果我再去掉配置里面的check="false",则运行结果会抛出异常，如下图：

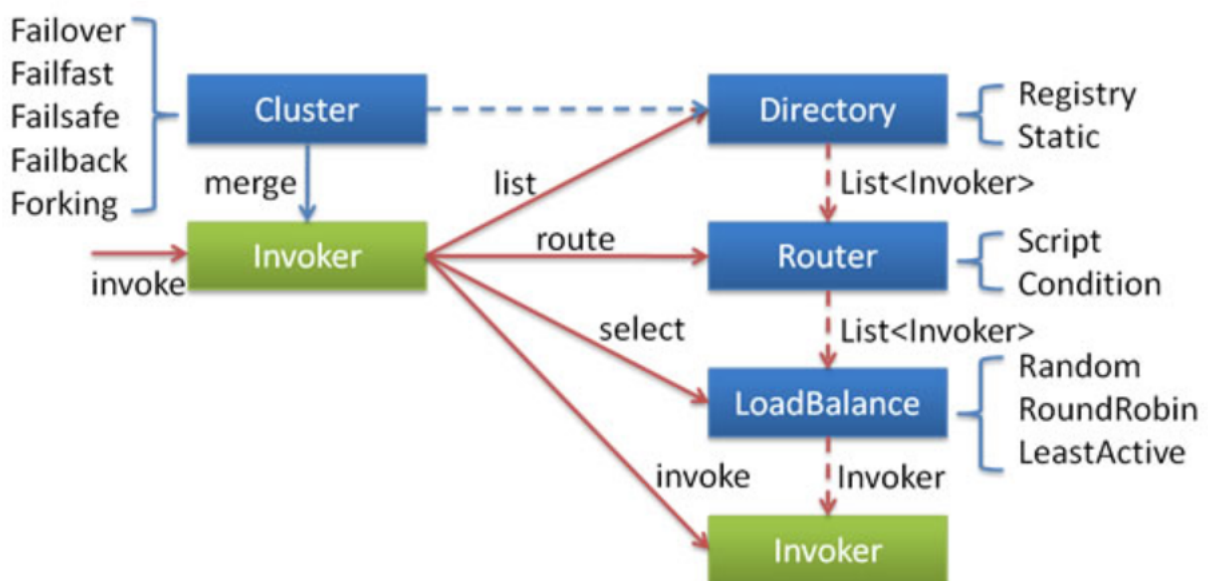


4.非常惊讶,为什么zookeeper明明不可用，但是还是能启动，这就是check的可用之处。

3.集群容错

在集群调用失败时，Dubbo提供了多种容错方案，缺省为failover重试。

1.下面这张图一定要理解到位



各节点关系：

- 这里的Invoker是Provider的一个可调用Service的抽象，Invoker封装了Provider地址及Service接口信息。
- Directory代表多个Invoker,可以把它看成List<Invoker>，但与List不同的是，它的值可能是动态变化的，比如注册中心推送变更。
- Cluster负责从多个Invoker中按路由规则选出子集，比如读写分离，应用隔离等。
- Router负责从等多个Invoker中按路由规则宣城子集，比如读写分离，应用隔离等。
- LoadBalance负责从多个Invoker中选出具体的一个用于本次调用，选的过程包含了负载均衡算法，调用失败后，需要重选。

2.集群容错模式：

集群模式配置：

按照以下示例在服务提供方和消费方配置集群模式

```
<dubbo:service cluster="failover">
```

或

```
<dubbo:reference cluster="failsafe">
```

1.Failover Cluster

失败自动切换，当出现失败，重试其他服务器，通常用于读操作，但重试会带来更长延迟。可通过retries="2"来设置重试次数（不含第一次）。

```
<dubbo:service retries="2">
或
<dubbo:reference retires="2">
或
<dubbo:reference>
  <dubbo:method name="findFoo" retires="2">
</dubbo:reference>
```

2.Failfast Cluster

快速失败，只发起一次调用，失败立即报错，通常用于非幂等性的写操作，比如新增记录。

3.Failsafe Cluster

失败安全，出现异常时，直接忽略，通常用于写入审计日志等操作

4.Failback Cluster

失败自动恢复，后台记录失败请求，定时重发，通常用于消息通知操作

5.Forking Cluster

并行调用多个服务器，只要一个成功即返回，通常用于实时性要求较高的读操作，但需要浪费更多服务资源，可通过fork="2"来设置最大并行数。

6.Broadcast Cluster

广播调用所有提供者，逐个调用，任意一台报错则报错。通常用于通知所有提供者更新缓存或日志等本地资源信息。

4.负载均衡

在集群负载均衡时，Dubbo提供了多种均衡策略，缺省为random随机调用。

负载均衡配置

举例：服务端服务级别配置（其他都类似）

```
<dubbo:service interface="..." loadbalance="roundrobin"/>
```

负载均衡策略

1.Random LoadBalance

随机，按权重设置随机概率。

在一个截面上碰撞的概率高，但调用量越大分布越均匀，而且按概率使用权重后也比较均匀，有利于动态调整提供者权重。

2.RoundRobin LoadBalance

轮询，按公约后的权重设置轮询比率。

存在慢的提供者累积请求的问题，比如：第二台机器很慢，但没挂，当请求调到第二台时就卡到那，久而久之，所有请求都卡在调用第二台上。

3.LeastActive LoadBalance

最少活跃调用数，相同活跃的随机，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。

4.ConsistenceHash LoadBalance

一致性Hash,相同参数的请求总是发到同一提供者。

当某一台提供者挂时，原本发往提供者的请求，基于虚拟节点，平摊到其他提供者，不会引起剧烈变动。

缺省只对第一个参数Hash,如果要修改，请配置<dubbo:parameter key="hash.arguments" value="0,1">

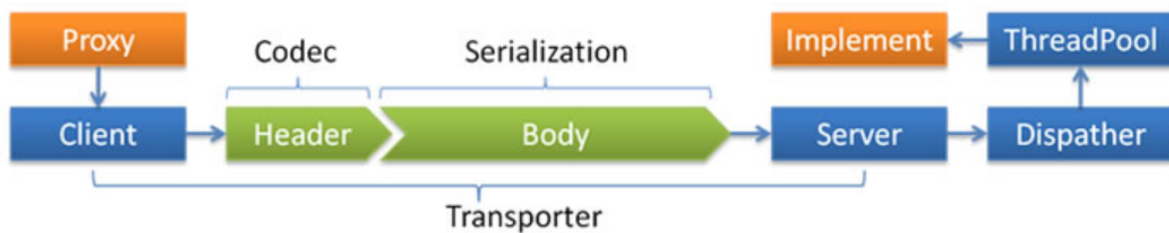
缺省用160份虚拟节点，如果要修改，请配置<dubbo:parameter key="hash.nodes" value="320">

5.线程模型

如果事件处理的逻辑能迅速完成，并且不会发起新的 IO 请求，比如只是在内存中记个标识，则直接在 IO 线程上处理更快，因为减少了线程池调度。

但如果事件处理逻辑较慢，或者需要发起新的 IO 请求，比如需要查询数据库，则必须派发到线程池，否则 IO 线程阻塞，将导致不能接收其它请求。

如果用 IO 线程处理事件，又在事件处理过程中发起新的 IO 请求，比如在连接事件中发起登录请求，会报“可能引发死锁”异常，但不会真死锁。



因此，需要通过不同的派发策略和不同的线程池配置的组合来应对不同的场景：

```
<dubbo:protocol name="dubbo",dispatcher="all" threadpool="fixed" threads="100">
```

Dispatcher

- 1.all: 所有消息都派发到线程池，包括请求，响应，连接事件，断开事件，心跳等。
- 2.direct: 所有消息都不派发到线程池，全部在 IO 线程上直接执行。
- 3.message: 只有请求响应消息派发到线程池，其它连接断开事件，心跳等消息，直接在 IO 线程上执行。
- 4.execution: 只有请求消息派发到线程池，不含响应，响应和其它连接断开事件，心跳等消息，直接在 IO 线程上执行。
- 5.connection: 在 IO 线程上，将连接断开事件放入队列，有序逐个执行，其它消息派发到线程池。

ThreadPool

- 1.fixed: 固定大小线程池，启动时建立线程，不关闭，一直持有。(缺省)
- 2.cache: 缓存线程池，空闲一分钟自动删除，需要时重建。
- 3.limited: 可伸缩线程池，但池中的线程数只会增长不会收缩。只增长不收缩的目的是为了避免收缩时突然来了大流量引起的性能问题。
- 4.eager: 优先创建 Worker 线程池。在任务数量大于 corePoolSize 但是小于 maximumPoolSize 时，优先创建 Worker 来处理任务。当任务数量大于 maximumPoolSize 时，将任务放入阻塞队列中。阻塞队列充满时抛出 RejectedExecutionException。(相比于 cached: cached 在任务数量超过 maximumPoolSize 时直接抛出异常而不是将任务放入阻塞队列)

6.直连提供者

1).修改 dubbo-provider.xml

```
<dubbo:application name="dubbo-provider"/>
<!--<dubbo:registry address="zookeeper://119.23.108.42:2181" />-->
<dubbo:protocol name="dubbo" port="20880"/>

<bean id="demoService" class="com.hh.DemoServiceImpl"/>
<dubbo:service interface="com.hh.DemoService" ref="demoService" registry="N/A" />
```

2).修改dubbo-consumer.xml

```
<dubbo:application name="dubbo-consumer"/>
  <!--<dubbo:registry address="zookeeper://119.23.108.42:2181" />
  <dubbo:protocol name="dubbo" port="20880"/>-->

  <dubbo:reference id="demoService" interface="com.hh.DemoService"
url="dubbo://localhost:20880"/>
</-->
```

3).运行代码执行成功

7.服务分组

1).dubbo-provider.xml

```
<dubbo:application name="dubbo-provider"/>
<dubbo:registry address="zookeeper://119.23.108.42:2181" />
<dubbo:protocol name="dubbo" port="20880"/>
<bean id="demoService" class="com.hh.DemoServiceImpl"/>
<bean id="demoService1" class="com.hh.DemoServiceImpl_01"/>
<dubbo:service interface="com.hh.DemoService" ref="demoService" group="aa"
protocol="dubbo"/>
<dubbo:service interface="com.hh.DemoService" ref="demoService1" group="bb"
protocol="dubbo"/>
```

2).dubbo-consumer.xml

```
<dubbo:application name="dubbo-consumer"/>
<dubbo:registry address="zookeeper://119.23.108.42:2181"/>
<dubbo:protocol name="dubbo" port="20880"/>
<dubbo:reference id="demoService" interface="com.hh.DemoService" group="aa"
protocol="dubbo"/>
<dubbo:reference id="demoService1" interface="com.hh.DemoService" group="bb"
protocol="dubbo"/>
```

3).DemoServiceImpl类代码：

```
@Service
public class DemoServiceImpl implements DemoService {
    @Override
    public String say(String name) {
        return ("DemoServiceImpl "+name);
    }
}
```

4).DemoServiceImpl_01类代码:

```
@Service
public class DemoServiceImpl_01 implements DemoService {
    @Override
    public String say(String name) {
        return (" DemoServiceImpl_01 "+name);
    }
}
```

5).Consumer类代码：

```
public class Consumer {
    public static void main(String[] args) {
        ClassPathXmlApplicationContext context = new
        ClassPathXmlApplicationContext("dubbo-consumer.xml");
        DemoService demoService = (DemoService) context.getBean("demoService");
        DemoService demoService1 = (DemoService) context.getBean("demoService1");
        String hh = demoService.say("huahua");
        String hh1 = demoService1.say("huahua");
        System.out.println(hh);
        System.out.println(hh1);
    }
}
```

6).运行结果：

```
D:\JAVA\jdk1.8.0_171\bin\java.exe ...
log4j:WARN No appenders could be found for logger (org.springframework)
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig
DemoServiceImpl huahua
DemoServiceImpl_01 huahua

Process finished with exit code 0
```

注意： 2.2.0 以上版本支持，总是只调一个可用组的实现

8.多版本

当一个接口实现，出现不兼容升级时，可以用版本号过渡，版本号不同的服务相互间不引用。

可以按照以下的步骤进行版本迁移：

- 在低压力时间段，先升级一半提供者为新版本
- 再将所有消费者升级为新版本

然后将剩下的一半提供者升级为新版本

类似如下配置：

```
<dubbo:service interface="com.foo.BarService" version="1.0.0" />
```