Table of Contents

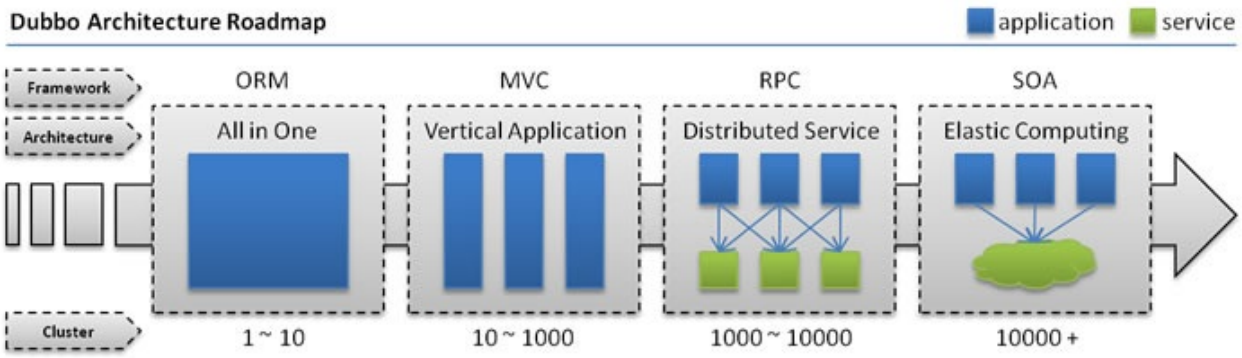
dubbo-user-book

The dubbo cookbook, covering almost all features of dubbo framework.

Introduction

Background

With the fast development of Internet, the scale of web applications expands unceasingly, and finally we find that the traditional vertical architecture(monolithic) can not handle this any more. Distributed service architecture and the flow computing architecture are imperative, and a governance system is urgently needed to ensure an orderly evolution of the architecture.



Monolithic architecture

When the traffic is very low, there is only one application, all the features are deployed together to reduce the deployment node and cost. At this point, the data access framework (ORM) is the key to simplifying the workload of the CRUD.

Vertical architecture

When the traffic gets heavier, add monolithic application instances can not accelerate the access very well, one way to improve efficiency is to split the monolithic into discrete applications. At this point, the Web framework (MVC) used to accelerate front-end page development is the key.

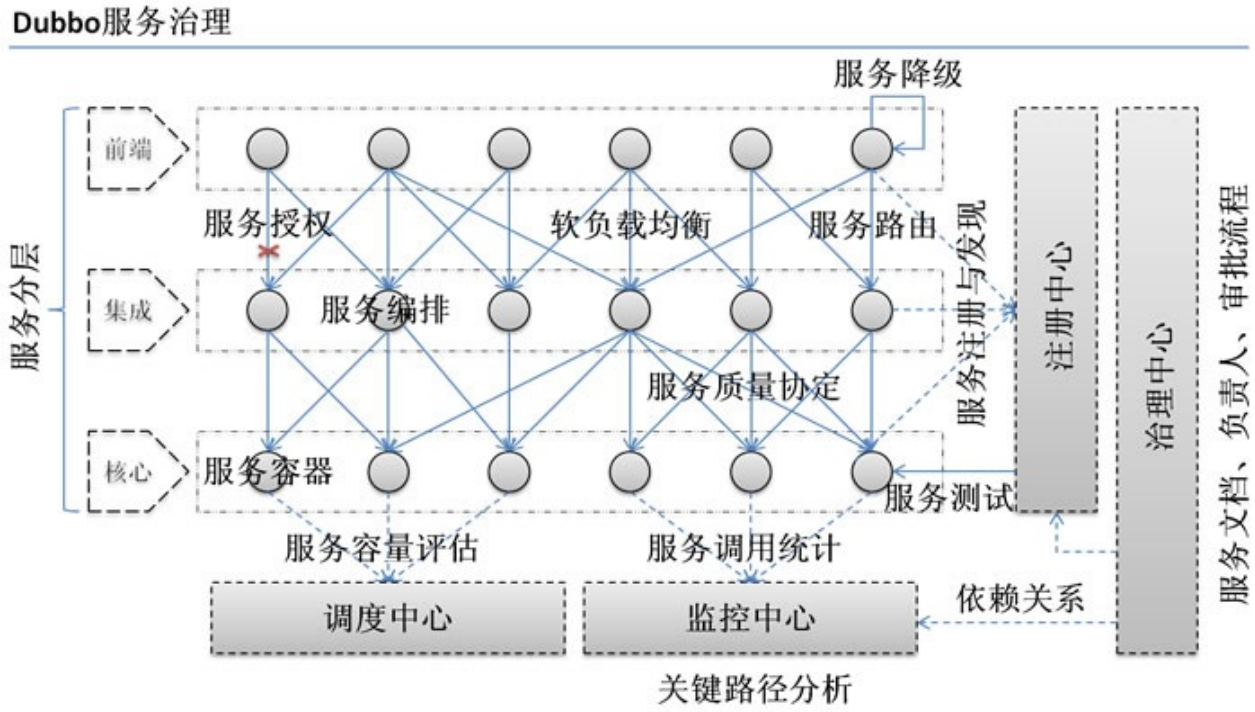
Distributed service architecture

When there are more and more vertical applications, the interaction between applications is inevitable, some core businesses are extracted and served as independent services, which gradually forms a stable service center，this way the front-end application can respond to the changeable market demand more quickly. At this point, the distributed service framework (RPC) for business reuse and integration is the key.

Flow computing architecture

When there are more and more services, capacity evaluation becomes difficult, and also services with small scales often causes waste of resources. To solve these problems, a scheduling center should be added to manage the cluster capacity based on traffics and to improve the utilization of the cluster. At this time, the resource scheduling and governance centers (SOA), which are used to improve machine utilization, are the keys.

Requirements



Before the advent of large-scare services, an application might just exposes or references remote service by using RMI or Hessian, the call is done by configuring serive URL, and load balance is done through hardwares, like F5.

When there are more and more services, it becomes very difficult to configure the service URL, the single point pressure of F5 hardware load balancer is also increasing. At this point, a service registry is needed to dynamically register and discover services to make the service's location transparent. By obtaining the list of service provider addresses in the consumer side, the soft load balancing and Failover can be realized, this reduces the dependence on the F5 hardware load balacer and some of the costs.

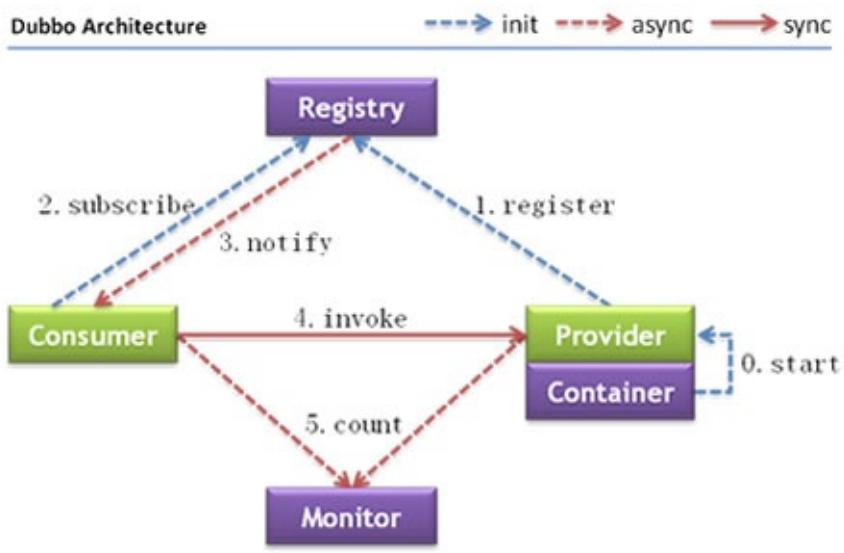
When things go further, the service dependencies become so complex that it can't even tell which applications to start before, even the architect can't fully describe the application architecture relationships. At this time, automatically draw the dependency diagram of the applications is needed to help the architect to be clear of the relationship.

Then, the traffic becomes even heavier, the capacity problem of the service is exposed, how many machines are needed to support this service? When should the machine be added? To solve these problems, first, the daily service calls and the amount of response time should be counted as a reference for capacity planning. Second, dynamically

adjust the weight, increase the weight of an online machine, and recorded the response time changes until it reaches the threshold, record the visits times at this time, then multiply this number of visits by the total number of machines to calculate the capacity in turn.

Above are the most basic requirements of Dubbo.

Architecture



Specification of Node's Role

Node	Role Spec
`Provider`	The provider exposes remote services
`Consumer`	The consumer calls the remote services
`Registry`	The registry is responsible for service discovery and configuration
`Monitor`	The monitor counts the number of service invocations and time-consuming
`Container`	The container manages the services's lifetime

Service relationship

1. `Container` is responsible for launching, loading, and running the service `Provider`.
2. `Provider` registers its services to `Register` at the time it starts.
3. `Consumer` subscribes the services it needs from the `Register` when it starts.
4. `Register` returns the `Provider`s list to `Consumer`, when it changes, the `Register` will push the changed data to `Consumer` through long connection.
5. `Consumer` selects one of the `Provider`s based on soft load balancing algorithm and executes the invocation, if fails, it will choose another `Provider`.
6. Both `Consumer` and `Provider` will count the number service invocations and time-consuming in memory, and send the statistics to `Monitor` every minute.

Dubbo has the following features: Connectivity, Robustness, Scalability and Upgradeability.

Connectivity

- `Register` is responsible for the registration and search of service addresses, like directory services, `Provider` and `Consumer` only interact with the registry during startup, and the registry does not forward requests, so it is less stressed
- 'Monitor' is responsible for counting the number of service invocations and time-consuming, the statistics will assembles in `Provider` 's and `Consumer` 's memory first and then sent to `Monitor`
- 'Provider' registers services to 'Register' and report time-consuming statistic(not include network overhead) to 'Monitor'
- 'Consumer' gets a list of service provider addresses from `Registry` , call the provider directly according to the LB algorithm, report the time-consuming statistic to `Monitor` , which includes network overhead
- The connections between `Register` , `Provider` and `Consumer` are long connections, `Moniter` is an exception
- `Register` is aware of the existence of `Provider` through the long connection, when `Provider` gets down, `Provider` will push the event to `Consumer`
- It doesn't affect the already running instances of `Provider` and `Consumer` even all of the `Register` and `Monitor` get down, since `Consumer` got a cache of `Provider` s list
- `Register` and `Monitor` are optional, `Consumer` can connect `Provider` directly

Robustness

- `Monitor` 's downtime doesn't affect the usage, only lose some sampling data
- When the DB server goes down, `Register` can return service `Provider` s list to `Consumer` by checking its cache, but new `Provider` cannot regiter any services
- `Register` is a peer cluster, it will automatically switch to another when any instance goes down
- Even all `Register` 's instances go down, `Provider` and `Consumer` can still conmunicate by checking their local cache
- Service `Provider` s are stateless, one instance's downtime doesn't affect the usage
- After all the `Provider` s of one service go down, `Consumer` can not use the that service, and infinitely reconnect to wait for service `Provider` to recover

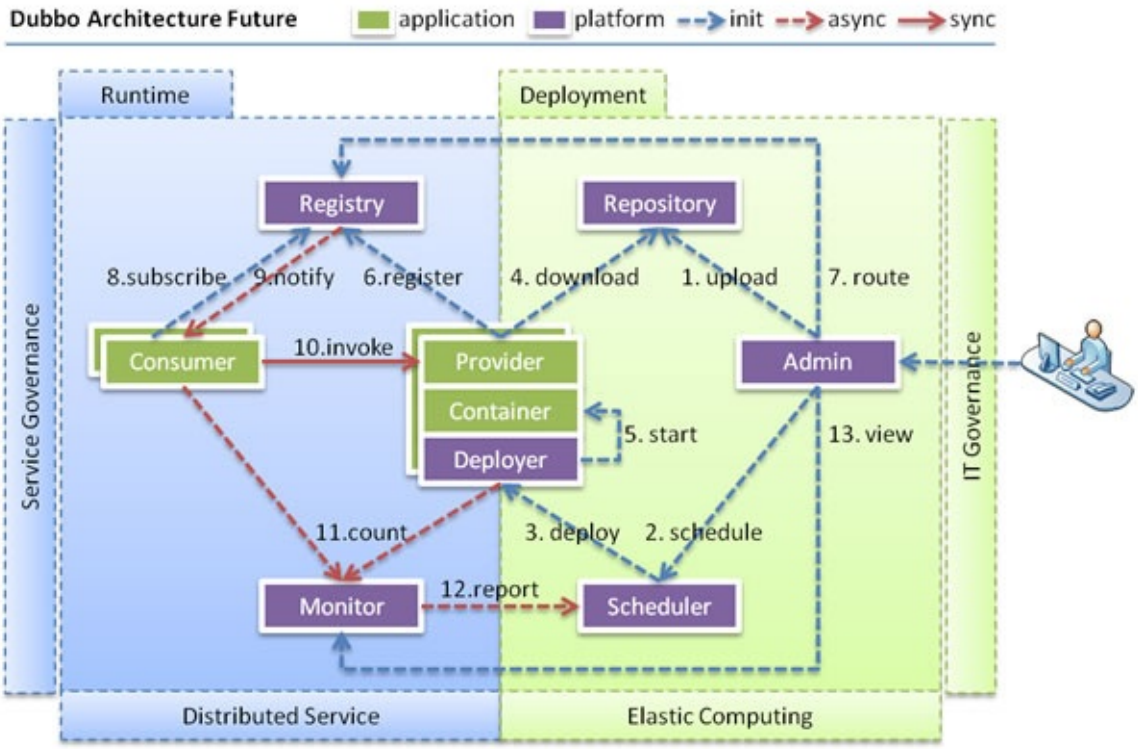
Scalability

- `Register` is a peer cluster that can dynamically increases its instances, all clients will automatically discover the new instances.

- `Provider` is stateless, it can dynamically increases the deployment instances, and the registry will push the new service provider information to the `Consumer`.

Upgradeablity

When the service cluster is further expanded and the IT governance structure is further upgraded, dynamic deployment is needed, and the current distributed service architecture will not bring resistance. Here is a possible future architecture:



Specification of Node's Role

Node	Role Spec
`Deployer`	Local proxy for automatic services deployment
`Repository`	The repository is used to store application packages
`Scheduler`	The scheduler automatically increases or decreases service providers based on the access pressure
`Admin`	Unified management console
`Registry`	the registry is responsible for service discovery and configuration
`Monitor`	The monitor counts the service call times and time-consuming

Usage

Spring configuration of local service

local.xml:

```
<bean id="xxxService" class="com.xxx.XxxServiceImpl" />
<bean id="xxxAction" class="com.xxx.XxxAction">
    <property name="xxxService" ref="xxxService" />
</bean>
```

Spring configuration of remote service

The remote configuration can be done by very little change based on the local configuration:

- split the `local.xml` into two part, put the service define part into `remote-privider.xml` (exists in the provider node), meanwhile the refrence part into `remote-consumer.xml` (exists in the consumer node).
- add `<dubbo:service>` to the provider's configuration, and `<dubbo:reference>` to the consumer's configuration.

remote-provider.xml:

```
<!-- define remote service bean the same way as local service bean -->
<bean id="xxxService" class="com.xxx.XxxServiceImpl" />
<!-- expose the remote service -->
<dubbo:service interface="com.xxx.XxxService" ref="xxxService" />
```

remote-consumer.xml:

```
<!-- reference the remote service -->
<dubbo:reference id="xxxService" interface="com.xxx.XxxService" />
<!-- use remote service the same say as local service -->
<bean id="xxxAction" class="com.xxx.XxxAction">
    <property name="xxxService" ref="xxxService" />
</bean>
```

Quick start

Dubbo uses a full Spring configuration, transparent access application,No API intrusion to your application,Just load the Dubbo configuration with Spring,Dubbo is loaded on the spring based schema extension.

If you don't want to use the Spring configuration, you can call it by the way of API.

Service provider

Complete installation steps, see：Provider demo installation

Defining service interfaces

DemoService.java [1] ：

```
package com.alibaba.dubbo.demo;

public interface DemoService {
    String sayHello(String name);
}
```

Implement interface in service provider

DemoServiceImpl.java [2] ：

```

package com.alibaba.dubbo.demo.provider;

import com.alibaba.dubbo.demo.DemoService;

public class DemoServiceImpl implements DemoService {
    public String sayHello(String name) {
        return "Hello " + name;
    }
}
```

Declaring exposure services with Spring configuration

provider.xml：

```
<?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/s
chema/dubbo        http://code.alibabatech.com/schema/dubbo/dubbo.xsd">

    <!-- Provider application information for computing dependencies -->
    <dubbo:application name="hello-world-app"  />

    <!-- Using the multicast broadcast registry to expose service addresses -->
    <dubbo:registry address="multicast://224.5.6.7:1234" />

    <!-- Exposing service at port 20880 with Dubbo protocol -->
    <dubbo:protocol name="dubbo" port="20880" />

    <!-- Declaration of service interfaces that need to be exposed  -->
    <dubbo:service interface="com.alibaba.dubbo.demo.DemoService" ref="demoService" />

    <!-- Implement services like local bean -->
    <bean id="demoService" class="com.alibaba.dubbo.demo.provider.DemoServiceImpl" />
</beans>
```

Loading Spring Configuration

Provider.java：

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

public class Provider {
    public static void main(String[] args) throws Exception {
        ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext(new
 String[] {"http://10.20.160.198/wiki/display/dubbo/provider.xml"});
        context.start();
        System.in.read(); // Press any key to exit
    }
}
```

Service consumer

Complete installation steps, see : Consumer demo installation

Using the Spring configuration to reference a remote service

consumer.xml：

```
<?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/s
chema/dubbo        http://code.alibabatech.com/schema/dubbo/dubbo.xsd">

    <!-- Consumer application names, used to calculate dependencies,not matching condi
tions, do not be the same as the provider -->
    <dubbo:application name="consumer-of-helloworld-app"  />

    <!-- Using the multicast broadcast registry to discovery the exposed  services -->
    <dubbo:registry address="multicast://224.5.6.7:1234" />

    <!-- Generate a remote service proxy that can be used as demoService as local bean
 -->
    <dubbo:reference id="demoService" interface="com.alibaba.dubbo.demo.DemoService" />

</beans>
```

Load the Spring configuration and call a remote service

Consumer.java [3] ：

```
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.alibaba.dubbo.demo.DemoService;

public class Consumer {
    public static void main(String[] args) throws Exception {
        ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext(new
 String[] {"http://10.20.160.198/wiki/display/dubbo/consumer.xml"});
        context.start();
        DemoService demoService = (DemoService)context.getBean("demoService"); //Obtai
ning a remote service proxy
        String hello = demoService.sayHello("world"); // Executing remote methods
        System.out.println( hello ); // Display the call result
    }
}
```

> [1]. The interface needs to be packaged separately, shared by the service provider and the consumer ↩

> [2]. Hidden realization of service consumer ↩

[3]

> [3]. IoC injection can also be used ↩

Dependencies

Necessary dependencies

JDK 1.6+ [1]

Default dependencies

use `mvn dependency:tree > dep.log` command to analysis，Dubbo default depends on the following 3rd party libraries：

```
[INFO] +- com.alibaba:dubbo:jar:2.5.9-SNAPSHOT:compile
[INFO] |  +- org.springframework:spring-context:jar:4.3.10.RELEASE:compile
[INFO] |  +- org.javassist:javassist:jar:3.21.0-GA:compile
[INFO] |  \- org.jboss.netty:netty:jar:3.2.5.Final:compile
```

All dependencies here are selected for the default configuration of the Dubbo, which are based on stability and performance considerations.

- javassist.jar [3]: if `<dubbo:provider proxy="jdk" />` or `<dubbo:consumer proxy="jdk" />`，or `<dubbo:application compiler="jdk" />`，is not required.
- spring-context.jar [4]: If you are using `ServiceConfig` and `ReferenceConfig` API calls, is not required.
- netty.jar [5]: if `<dubbo:protocol server="mina"/>` or `<dubbo:protocol server="grizzly"/>`，Then change to mina.jar or grizzly.jar. If `<protocol name="rmi"/>`，is not required.

Optinal dependencies

These dependencies needs to be added to project manually，when you need them.

- netty-all 4.0.35.Final
- mina: 1.1.7
- grizzly: 2.1.4
- httpclient: 4.5.3
- hessian_lite: 3.2.1-fixed
- fastjson: 1.2.31

- zookeeper: 3.4.9
- jedis: 2.9.0
- xmemcached: 1.3.6
- hessian: 4.0.38
- jetty: 6.1.26
- hibernate-validator: 5.4.1.Final
- zkclient: 0.2
- curator: 2.12.0
- cxf: 3.0.14
- thrift: 0.8.0
- servlet: 3.0 [6]
- validation-api: 1.1.0.GA [6]
- jcache: 1.0.0 [6]
- javax.el: 3.0.1-b08 [6]
- kryo: 4.0.1
- kryo-serializers: 0.42
- fst: 2.48-jdk-6
- resteasy: 3.0.19.Final
- tomcat-embed-core: 8.0.11
- slf4j: 1.7.25
- log4j: 1.2.16

> [1]. In theory, Dubbo only depend on JDK, not depend on any 3rd party libs, you can finish logic by useing JDK. ↩

> [2]. Log output jar ↩

> [3]. Bytecode generation ↩

> [4]. Configuration parsing ↩

> [5]. Network transmission ↩

> [6]. JAVAEE ↩

Maturity

Function maturity

Feature	Maturity	Strength	Problem	Advise	
Concurrency control	Tested	concurrency control		On trial	
Connection control	Tested	connection number control		On trial	
Connecting certain provider straightly	Tested	Provider service for point-to-point connecting straightly,for test		Can be used in the test environment	A
Grouping polymerization	Tested	Return vlue of grouping polymerization, service for menu aggregation and other services	Used in special scenes	Can be used in the production environment	
Parameters validator	Tested	parameters validator,JSR303 validation framework integration	Have effect on Performance	On trial	La
Result cache	Tested	result cache,for accelerating requests		On trial	
Generic reference	Stable	Generic reference,remote call without a business interface class，for test platforms, open api proxy service, and so on		Can be used in the production environment	A
Generic service	Stable	Generic service,no interface class is required to implement any interface,for mock paltform		Can be used in the production environment	A

Echo test	Tested	echo test		On trial	
Attachment	Stable	Attachment		Can be used in the production environment	
Asynchronous call	Tested	Unavailable asynchronous call		On trial	
Local call	Tested	Local call		On trial	
Callback parameter	Tested	Callback parameter	Used in special scenes	On trial	R
Events notify	Tested	Events notify, triggering before and after the remote call is executed		On trial	
Local stub	Stable	Performing part of the logic on the client side		Can be used in the production environment	A
Local mock	Stable	Forged return results, which can be executed when failed, or directly executed, for service degradation	Need support of registry	Can be used in the production environment	A
Delay publish	Stable	Delay publish,used to wait for the application to load warmup data,or wait for spring context to load completly		Can be used in the production environment	A
Lazy connect	Tested	Delay setting up connections,when invocation is set up		On trial	R
Stickness connections	Tested	Stickness connections,always make a request to the same provider service unless the service is down, and then switch to another		On trial	R

Token authorization	Tested	Token authorization, is used for service authorization	Need support of registry	On trial	
Routing rule	Tested	Dynamically determining the call relationship	Need support of registry	On trial	
Configuration rule	Tested	Distribute the configuration dynamically ,is the switch of business logic	Need support of registry	On trial	
Accesslog	Tested	Accesslog,used to record call information	Local storage, impact performance, limited by disk size	On trial	
Distributed transaction	Research	JTA/XA three phase submission transaction(TCC)	Unstable	Unavailable	

Strategy maturity

Feature	Maturity	Strength	Problem	
Zookeeper registry	Stable	Support the cluster,have various of related open source products, dubbo-2.3.3 and above versions are recommended	Depended on the stability of zookeeper	C th e
Redis registry	Stable	Support the client - based double - write clustering method with high performance	Please ensure server time synchronization,be used to check the expired dirty data of heartbeat	C th e
Multicast registry	Tested	Decentration,no registry needs to be installed	Depending on the network topology and routing, there is a risk across the server rooms	C s d e
Simple registry	Tested	Dogfooding,the registry itself is also a standard RPC service	No cluster support, may occur single-point failure	C

Feature	Maturity	Strength	Problem	A
Simple monitor system	Stable	Support JFreeChart statistics report	No cluster support, may occur single-point failure,but the failure does not affect the RPC call	C th e
Feature	Maturity	Strength	Problem	A
Dubbo protocol	Stable	Use NIO to reuse a single long connection and use a thread pool to process requests concurrently, Reduce handshake and increase concurrency efficiency, good performance	A single connection will become a bottleneck in the transmission of large files	C th e
Rmi protocol	Stable	Interoperable with native RMI, based on the TCP protocol	Occasionally the connection fails, and the stub needs to be rebuilt	C th e
Hessian protocol	Stable	Interoperable with native Hessian, based on the HTTP protocol	Hessian.jar support is required, and the overhead of HTTP short connections is large	C th e
Feature	Maturity	Strength	Problem	A
Netty Transporter	Stable	The NIO framework of JBoss, has good performance	A request sends two events and needs to shield useless events	C th e
Mina Transporter	Stable	Classic NIO framework，stable	The dispatch of the message queue is not timely, under great pressure, there will be FullGC	C th e
Grizzly Transporter	Tested	The NIO framework of Sun,applied in the GlassFish container	The thread pool is not extensible, and Filter can't intercept the next filter	C
Feature	Maturity	Strength	Problem	A
Hessian	Stable	Good performance,multilingual	The compatibility of various versions of Hessian is not good, it may be in conflict with the Hessian used in the	C th

		support (recommended)	application, and the Dubbo is embedded with the source code of the hessian3.2.1	e
Dubbo Serialization	Tested	The performance is better in a large number of POJO transmission by not transmitting the class information of POJO.	When a field is added to the parameter object, an external file declaration is required	C
Json Serialization	Tested	pure text,can be cross-language parsed,default using FastJson	Poor performance	C
Java Serialization	Stable	Java native support	Poor performance	C th e
Feature	Maturity	Strength	Problem	A
Javassist ProxyFactory	Stable	Bytecode generation instead of reflection,good performance(recommended)	Depending on the javassist.jar and taking up the JVM's Perm memory, the Perm may have to be larger:java -XX:PermSize=128m	C th e
Jdk ProxyFactory	Stable	JDK native support	Poor performance	C th e
Feature	Maturity	Strength	Problem	A
Failover Cluster	Stable	Failure automatically switches, when failure occurs, retries other servers, usually used for read operations.(recommended)	Retry will lead to longer delays	C th e
Failfast Cluster	Stable	Fast failure, only one call, failure to be reported immediately, usually used for non idempotent writing.	If a server is being restarted, a call failure may occur	C th e
Failsafe Cluster	Stable	Failsafe, when abnormal, directly ignored, usually used to write to the audit log and other operations	Call information loss	C th e
Failback		Failure auto recovery, backstage record failure request, regular	Unreliable, lost when restart the	C th

Failback Cluster	Tested	request, regular retransmission, usually used for message notification operations	when restart the server	th e
Forking Cluster	Tested	Multiple servers are invoked in parallel, as long as one success is returned, often used for high real-time reading operations.	Need to waste more service resources	C th e
Broadcast Cluster	Tested	A broadcast calls all providers, one by one, and any error is wrongly reported, usually used to update the provider's local state	The speed is slow, and any false report is wrong.	C th e
Feature	Maturity	Strength	Problem	A
Random LoadBalance	Stable	Random probability, set random probability according to weight(recommended)	The probability of a collision on a cross section is high. When retrying, there may be an unequal instantaneous pressure.	C th e
RoundRobin LoadBalance	Stable	Round Robin,setting wheel based ratio according to the weight after the Convention	There is a slow machine accumulation request problem, and extreme circumstances may cause an avalanche	C th e
LeastActive LoadBalance	Stable	The least active call number, the random number of the same active number, the active number is the count difference before and after the call, making the slow machine receive less request.	Do not support the weight, in the capacity planning, not to pressure a machine oriented pressure measurement by weight capacity	C th e
ConsistentHash LoadBalance	Stable	The consistency hash, the same parameters always request to the same provider, when one provider hung, originally sent to the provider's request, based on virtual nodes, spread to other providers, will not cause dramatic changes	Uneven distribution of pressure	C th e

Feature	Maturity	Strength	Problem	A
Condition routing rule	Stable	Routing rules based on conditional expressions, simple and easy to use	There are some complex multi branch conditions, and the rules are difficult to describe	C th e
Script routing rules	Tested	Routing rules based on the script engine, powerful	No sandbox is running, scripting ability is too powerful and may be the back door	C
Feature	Maturity	Strength	Problem	A
Spring Container	Stable	Automatically load all Spring configurations under the META-INF/spring directory		C th e
Jetty Container	Stable	Start an embedded Jetty for reporting state	When a large number of pages are accessed, the threads and memory of the server are affected	C th e
Log4j Container	Stable	Configuring the configuration of the log4j automatically, automatically subdirecting the log files by process at the startup of multiple processes	The user can't control the configuration of log4j, inflexible	C th e

Configuration

XML Configuration

About the XML configuration items, see：XML References. If you prefer use API directly instead of using Spring, see API Configuration. Want a example of how to use configuration, see Quick Start。

provider.xml demo

```
<?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.springf
ramework.org/schema/beans/spring-beans.xsd http://code.alibabatech.com/schema/dubbo ht
tp://code.alibabatech.com/schema/dubbo/dubbo.xsd">
    <dubbo:application name="hello-world-app"  />
    <dubbo:registry address="multicast://224.5.6.7:1234" />
    <dubbo:protocol name="dubbo" port="20880" />
    <dubbo:service interface="com.alibaba.dubbo.demo.DemoService" ref="demoServiceLoca
l" />
    <dubbo:reference id="demoServiceRemote" interface="com.alibaba.dubbo.demo.DemoServ
ice" />
</beans>
```

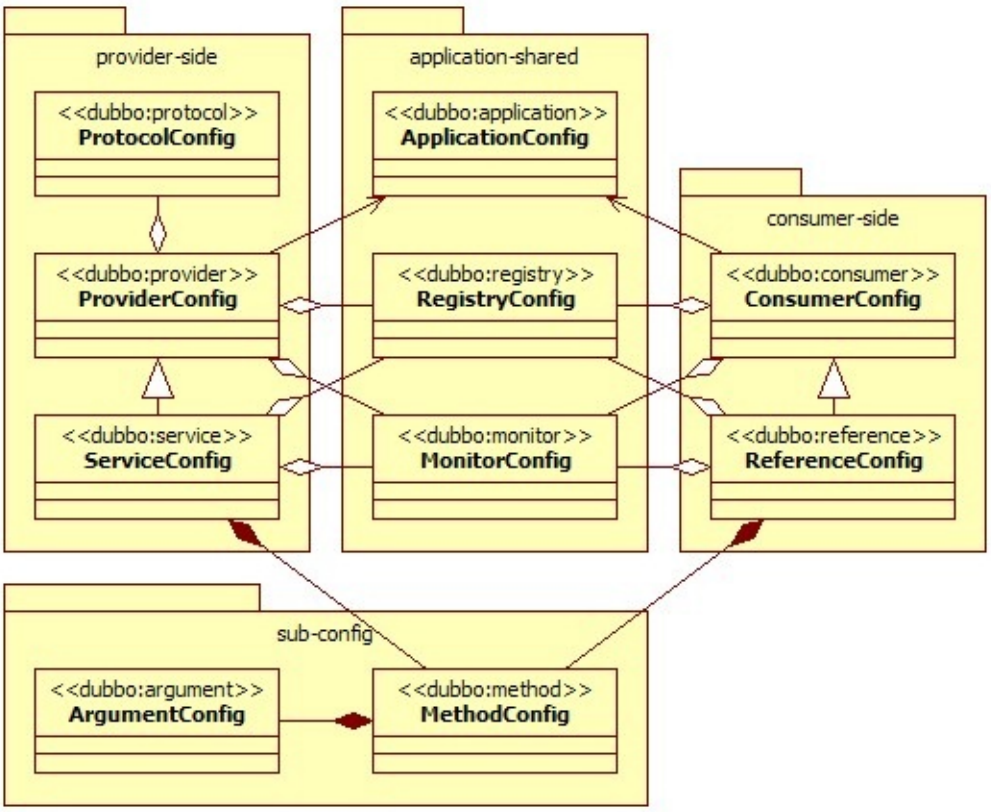
All tags support custom parameters, so we can meet the special config requirements at different extension points, such as:

```
<dubbo:protocol name="jms">
    <dubbo:parameter key="queue" value="your_queue" />
</dubbo:protocol>
```

Or:

```
<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"
    xmlns:p="http://www.springframework.org/schema/p"
    xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springf
ramework.org/schema/beans/spring-beans.xsd http://code.alibabatech.com/schema/dubbo ht
tp://code.alibabatech.com/schema/dubbo/dubbo.xsd">
    <dubbo:protocol name="jms" p:queue="your_queue" />
</beans>
```

The relations between configuration tags



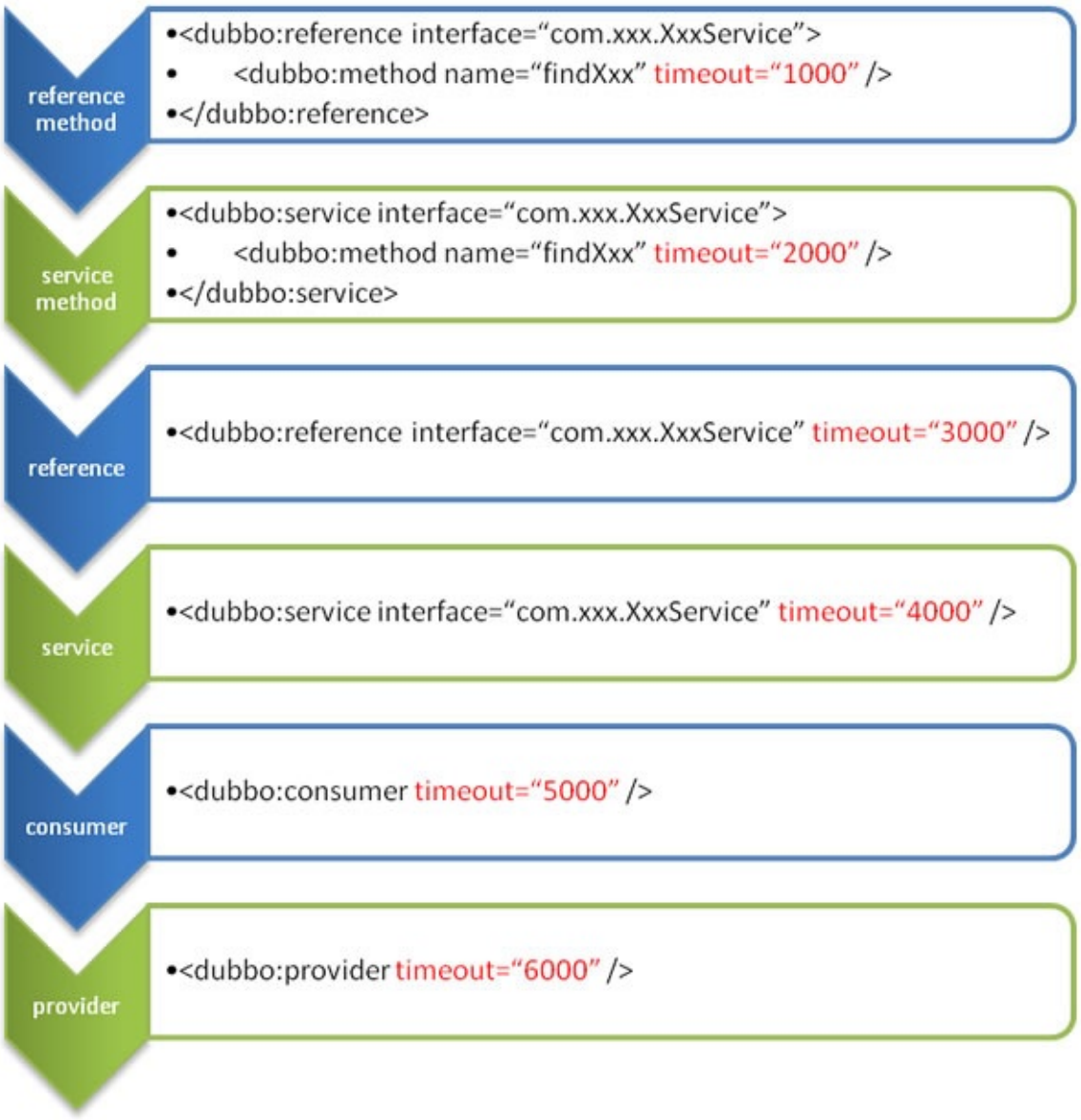
tag	purpose	introduction
`<dubbo:service/>`	Service Export	Used to export service, define service metadata, export service with mutiple protocols, register service to multiple registries
`<dubbo:reference/>`	Service Reference	Used to create a remote proxy, subscribe to multiple registries
`<dubbo:protocol/>`	Protocol Config	Configure the protocol for services on provider side, the consumer side follows.
`<dubbo:application/>`	Application Config	Applies to both provider and consumer.
`<dubbo:module/>`	Module Config	Optional.
`<dubbo:registry/>`	Registry Center	Registry info: address, protocol, etc.
`<dubbo:monitor/>`	Monitor Center	Monitor info: address, address, etc. Optional.
`<dubbo:provider/>`	Default Config for Providers	Default Config for ServiceConfigs. Optional.
`<dubbo:consumer/>`	Default Config for Consumers	Default Config for ReferenceConfigs. Optional.
`<dubbo:method/>`	Method level Config	Method level Config for ServiceConfig and ReferenceConfig.
`<dubbo:argument/>`	Argument Config	Used to specify the method parameter configuration.

Overrides and Priorities

Take timeout as an example, here is the priorities, from high to low (retries, loadbalance, actives also applies the same rule):

- method level，interface level，default/global level。
- at the same leveel, consumer has higher priority than provider

Configurations on the provider side are passed to the consumer side through registry in the form of URL.



It is recommended that the provider set a timeout for every service, because the provider knows exactly how long a method needs to be executed. If a consumer cites multiple services at the same time, it doesn't need to care about the timeout settings of each service.

Theoretically, almost all configuration items supported in ReferenceConfig can be configured with a default value using ConsumerConfig, ServiceConfig, ProviderConfig.

1: Requires spring `3.2.16+`, see announcement for details：`xmlns:p="http://www.springframework.org/schema/p"`
2: The reference bean obeys lazy init by default, only if it is refered by other beans or other instance try to get its instance using `getBean()` method will the reference be initialized. If you need eager init, config this way: `<dubbo:reference ... init="true" />`

Properties Configuration

If your application is simple enough, say, you do not need multi-registries or multi-protocols, and you want to share configuration among Spring containers. You can use `dubbo.properties` as default configuration.

Dubbo will load dubbo.properties under the root of classpath automatically, you can also specify the path for loading this file by using JVM parameter: `-Ddubbo.properties.file=xxx.properties`.

Mapping Rules

Combine the tag name and attribute name of the XML tag, use `.` to split. One property per line.

- `dubbo.application.name=foo` equals to `<dubbo:application name="foo" />`
- `dubbo.registry.address=10.20.153.10:9090` equals to `<dubbo:registry address="10.20.153.10:9090" />`

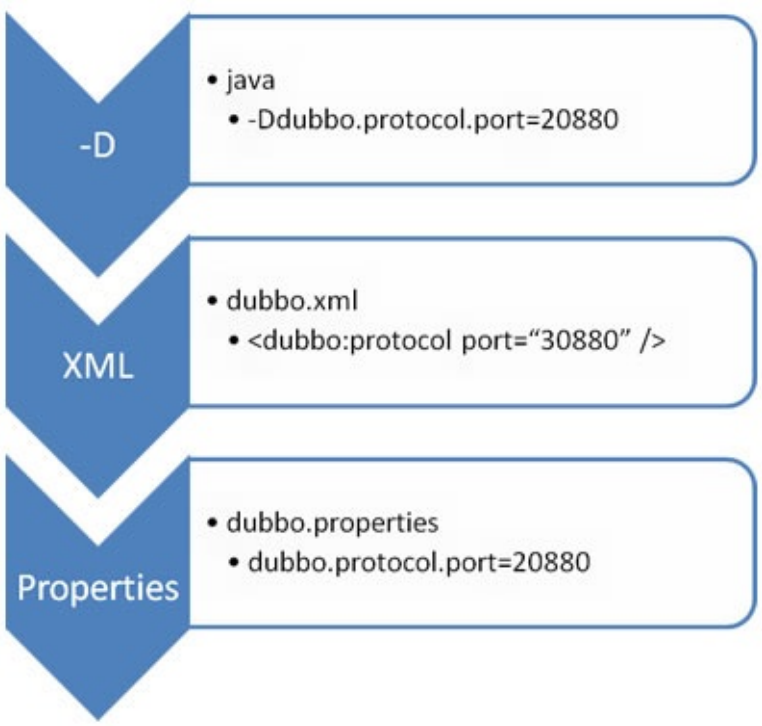
If you have more than one tags in a XML configuration, you can use the `id` value to distinguish. If you don't specify a id, ti will applied to all tags.

- `dubbo.protocol.rmi.port=1234` equals to `<dubbo:protocol id="rmi" name="rmi" port="1099" />`
- `dubbo.registry.china.address=10.20.153.10:9090` equals to `<dubbo:registry id="china" address="10.20.153.10:9090" />`

Here is a typical dubbo.properties demo configuration：

```
dubbo.application.name=foo
dubbo.application.owner=bar
dubbo.registry.address=10.20.153.10:9090
```

Overrides and Priorities



Priorities from high to low:

- JVM -D parameters, you can easily override configuration when deploying or starting applications, e.g., change the port of dubbo protocol.
- XML, the properties present in XML will override that in dubbo.properties.
- Properties, the default value, only works when it is not configured with XML or JVM.

1: If more than one dubbo.properties under classpath, say, two jars contains dubbo.properties separately, Dubbo will arbitarily choose one to to load, and log Error info.
2: If `id` not configured on `protocol`, will use `name` property as default

API Configuration

All API properties have counterparts in XML, see XML References for details. For example `ApplicationConfig.setName("xxx")` equals to `<dubbo:application name="xxx" />` [1]

Provider Side

```
import com.alibaba.dubbo.rpc.config.ApplicationConfig;
import com.alibaba.dubbo.rpc.config.RegistryConfig;
import com.alibaba.dubbo.rpc.config.ProviderConfig;
import com.alibaba.dubbo.rpc.config.ServiceConfig;
import com.xxx.XxxService;
import com.xxx.XxxServiceImpl;

// Implementation
XxxService xxxService = new XxxServiceImpl();

// Application Info
ApplicationConfig application = new ApplicationConfig();
application.setName("xxx");

// Registry Info
RegistryConfig registry = new RegistryConfig();
registry.setAddress("10.20.130.230:9090");
registry.setUsername("aaa");
registry.setPassword("bbb");

// Protocol
ProtocolConfig protocol = new ProtocolConfig();
protocol.setName("dubbo");
protocol.setPort(12345);
protocol.setThreads(200);

// NOTES: ServiceConfig holds the serversocket instance and keeps connections to regis
try, please cache it for performance.

// Exporting
ServiceConfig<XxxService> service = new ServiceConfig<XxxService>(); // In case of mem
ory leak, please cache.
service.setApplication(application);
service.setRegistry(registry); // Use setRegistries() for multi-registry case
service.setProtocol(protocol); // Use setProtocols() for multi-protocol case
service.setInterface(XxxService.class);
service.setRef(xxxService);
service.setVersion("1.0.0");

// Local export and register
service.export();
```

Consumer Side

```
import com.alibaba.dubbo.rpc.config.ApplicationConfig;
import com.alibaba.dubbo.rpc.config.RegistryConfig;
import com.alibaba.dubbo.rpc.config.ConsumerConfig;
import com.alibaba.dubbo.rpc.config.ReferenceConfig;
import com.xxx.XxxService;

// Application Info
ApplicationConfig application = new ApplicationConfig();
application.setName("yyy");

// Registry Info
RegistryConfig registry = new RegistryConfig();
registry.setAddress("10.20.130.230:9090");
registry.setUsername("aaa");
registry.setPassword("bbb");

// NOTES: ReferenceConfig holds the connections to registry and providers, please cach
e it for performance.

// Refer remote service
ReferenceConfig<XxxService> reference = new ReferenceConfig<XxxService>(); // In case
of memory leak, please cache.
reference.setApplication(application);
reference.setRegistry(registry);
reference.setInterface(XxxService.class);
reference.setVersion("1.0.0");

// Use xxxService just like a local bean
XxxService xxxService = reference.get(); // NOTES: Please cache this proxy instance.
```

Specials

Only care about the differences:

Configuration of Method level

```
...

// Method level config
List<MethodConfig> methods = new ArrayList<MethodConfig>();
MethodConfig method = new MethodConfig();
method.setName("createXxx");
method.setTimeout(10000);
method.setRetries(0);
methods.add(method);

// Referring
ReferenceConfig<XxxService> reference = new ReferenceConfig<XxxService>();
...
reference.setMethods(methods);

...
```

Peer to Peer

```
...

ReferenceConfig<XxxService> reference = new ReferenceConfig<XxxService>();
// If you know the address of the provider and want to bypass the registry, use `refer
ence.setUrl()` to specify the provider directly. Refer [How to Invoke a specific provi
der](../demos/explicit-target.md) for details.
reference.setUrl("dubbo://10.20.130.230:20880/com.xxx.XxxService");

...
```

> [1]. When should we usd API: API is very useful for integrating with systems like OpenAPI, ESB, Test, Mock, etc. General Providers and Consumers, we still recommend use XML Configuration. ↩

Annotation Configuration

Requires `2.5.7` or higher

Provider Side

`Service` annotation for exporting

```
import com.alibaba.dubbo.config.annotation.Service;

@Service(timeout = 5000)
public class AnnotateServiceImpl implements AnnotateService {
    // ...
}
```

Use JavaConfig for common parts

```
@Configuration
public class DubboConfiguration {

    @Bean
    public ApplicationConfig applicationConfig() {
        ApplicationConfig applicationConfig = new ApplicationConfig();
        applicationConfig.setName("provider-test");
        return applicationConfig;
    }

    @Bean
    public RegistryConfig registryConfig() {
        RegistryConfig registryConfig = new RegistryConfig();
        registryConfig.setAddress("zookeeper://127.0.0.1:2181");
        registryConfig.setClient("curator");
        return registryConfig;
    }
}
```

Path to scan

```
@SpringBootApplication
@DubboComponentScan(basePackages = "com.alibaba.dubbo.test.service.impl")
public class ProviderTestApp {
    // ...
}
```

Consumer Side

`Reference` annotation for reference

```
public class AnnotationConsumeService {

    @com.alibaba.dubbo.config.annotation.Reference
    public AnnotateService annotateService;

    // ...
}
```

Use JavaConfig for common parts

```
@Configuration
public class DubboConfiguration {

    @Bean
    public ApplicationConfig applicationConfig() {
        ApplicationConfig applicationConfig = new ApplicationConfig();
        applicationConfig.setName("consumer-test");
        return applicationConfig;
    }

    @Bean
    public ConsumerConfig consumerConfig() {
        ConsumerConfig consumerConfig = new ConsumerConfig();
        consumerConfig.setTimeout(3000);
        return consumerConfig;
    }

    @Bean
    public RegistryConfig registryConfig() {
        RegistryConfig registryConfig = new RegistryConfig();
        registryConfig.setAddress("zookeeper://127.0.0.1:2181");
        registryConfig.setClient("curator");
        return registryConfig;
    }
}
```

Path to scan

```
@SpringBootApplication
@DubboComponentScan(basePackages = "com.alibaba.dubbo.test.service")
public class ConsumerTestApp {
    // ...
}
```

NOTES

All annotations in 2.5.7 will be removed later, if you have used these annotations in your project, please upgrade to the latest version.

```
<dubbo:annotation package="com.alibaba.dubbo.test.service" />
```

Example

Check on start up

By default dubbo will check if the dependent service is available at startup . It will throw an exception to prevent Spring complete initialization when it is not available, so that you can find the problems early before publishing you application, the default setting: `check=true` .

You can turn off checking by `check=false` . For example, some services do not care it when you run testing, or you must have one started firstly because of circular dependency.

In addition, if your Spring bean is lazy-loaded or you delay reference service with API programming, turn off the check, otherwise the service will throw an exception when the service is temporarily unavailable ,then get a null reference. If you configure `check=false` ,you can get a reference . When the service is restored, the service can automatically reconnect.

Example

Use the spring configuration file

Disable the startup check of a service (throw some exception/error when no provider is provided):

```
<dubbo:reference interface = "com.foo.BarService" check = "false" />
```

Disable startup checking for all services (throw some exception/error when not provided):

```
<dubbo:consumer check = "false" />
```

Disable the registration center startup check (registration subscription failed error):

```
<dubbo:registry check="false" />
```

Use dubbo.properties

```
dubbo.reference.com.foo.BarService.check = false
dubbo.reference.check = false
dubbo.consumer.check = false
dubbo.registry.check = false
```

Use the -D parameter

```
java -Ddubbo.reference.com.foo.BarService.check = false
java -Ddubbo.reference.check = false
java -Ddubbo.consumer.check = false
java -Ddubbo.registry.check = false
```

Configuration Meaning

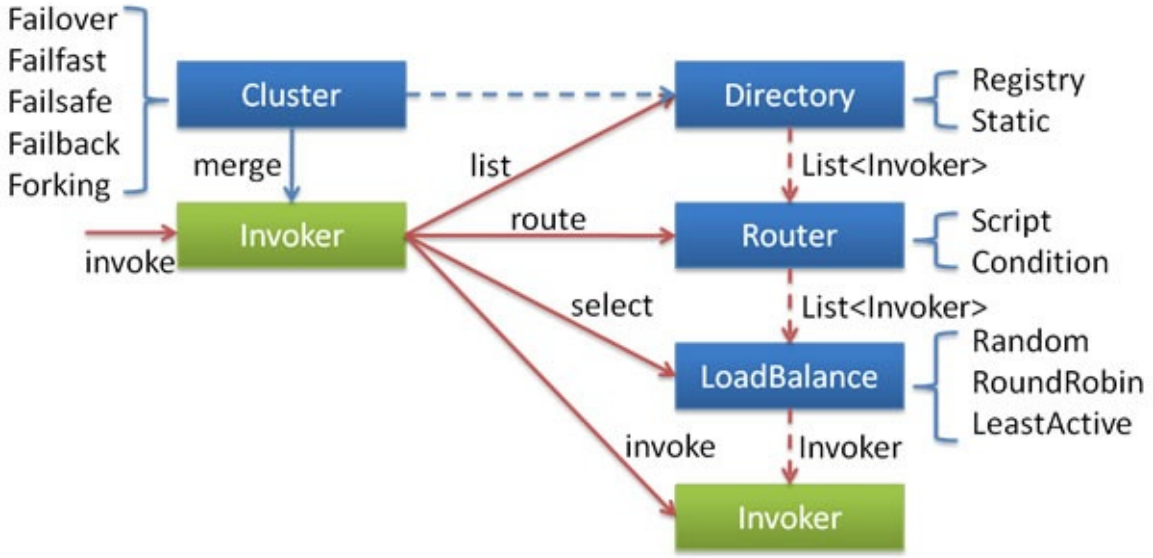
`dubbo.reference.check=false`, Change the check value of all references forcibly, even if the configuration has a declaration, it also will be overwritten.

`dubbo.consumer.check=false` The default value of `check`. It will not be affected if there is an explicit declaration in the configuration such as `` `<dubbo: reference check =" true "/> ``.

`dubbo.registry.check=false`, The two configuration above is to express success of the subscription. If the subscription is also allowed to start when the registration fails for the provider list is empty, you need to use this configuration. The system will try again in the background regularly.

Fault Tolerance Strategy

Dubbo offers a variety of fault-tolerant scenarios when a cluster call fails, with a default failover retry.



The relationship between nodes:

- This `Invoker` is the callable Service's abstract of the `Provider`, and the `Invoker` packaging the `Provider`'s address and `Service`'s interface.
- The `Directory` represent multiple `Invoker`,You can think of it as `List<Invoker>`,But unlike `List`,its value can be dynamically changing.such as registry push changes
- The `Cluster` disguises multiple `Invoker` in `Directory` as a `Invoker`,The upper transparent, masquerade process contains fault-tolerant logic, call failed, try another
- The `Router` is responsible for selecting subsets according to routing rules from multiple `Invoker`s, such as read-write separation, application isolation, etc.
- `LoadBalance` is responsible for selecting a specific one from multiple `Invoker` for this call. The selection process includes the load balancing algorithm. If the call fails, it needs to be re-selected

Cluster fault-tolerant mode

You can expand the cluster fault tolerance strategy, see:Cluster expansion

Failover Cluster

Failure automatically switch, when there is failure, retry the other server (default). Usually used for read operations, but retries can result in longer delays. The times of retries can be set via `retries =2` (excluding the first time).

The times of retries is configured as follows:

```
<dubbo:service retries="2" />
```

OR

```
<dubbo:reference retries="2" />
```

OR

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

Failfast Cluster

Fast failure, only made a call, failure immediately error. Usually used for non-idempotent write operations, such as adding records

Failsafe Cluster

Failure of security, anomalies, directly ignored. Usually used to write audit logs and other operations.

Failback Cluster

Failure automatically restored, failed to record the background request, regular retransmission. Usually used for message notification operations.

Forking Cluster

Multiple servers are invoked in parallel, returning as soon as one succeeds. Usually used for real-time demanding read operations, but need to waste more service resources. The maximum number of parallelism can be set with `forks=2`.

Broadcast Cluster

Calling all providers broadcast, one by one call, any error is reported (`2.1.0+`). It is usually used to notify all providers to update local resource information such as caches or logs.

Cluster mode configuration

Follow the example below to configure cluster mode on service providers and consumers

```
<dubbo:service cluster="failsafe" />
```

OR

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

LoadBalance

Dubbo offers a number of balancing strategies for cluster load balancing, which defaults to `random`.

You can extend the load balancing strategy by yourself, see:LoadBalance extension

LoadBalance strategy

Random LoadBalance

- **Ramdom**, set random probabilities by weight.
- The probability of collisions on one section is high, but the larger the amount of calls, the more uniform the distribution. And when use weight based on probability the distribution turns out to be uniform, which also helps to dynamically adjust the provider weights.

RoundRobin LoadBalance

- **RoundRobin**, use the weight's common advisor to determine round robin ratio.
- Traffic flow to slower providers may cause requests piled up, e.g., if there's a provider processing requests in a very slow speed, but it's still alive, which means it can receive request as normal. According to RoundRobin policy, consumers will continuously send requests to this provider on predetermined pace, have no aware of the provider's bad status. Finally, we will get many requests stuck on this unhealthy provider.

LeastActive LoadBalance

- **LeastActive**, a random mechanism based on actives, `actives` means the num of requests a consumer have sent but not return yet ◦
- Slower providers will receive fewer requests, cause slower provider have higher `actives`.

ConsistentHash LoadBalance

- **ConsistentHash**, the same parameters of the request is always sent to the same provider.
- When a provider fails, the original request to the provider, based on the virtual node

algorithm, averages to other providers, does not cause drastic changes.

- Algorithm reference：http://en.wikipedia.org/wiki/Consistent_hashing
- By default only the first parameter Hash, if you want to modify, please configure `<dubbo:parameter key="hash.arguments" value="0,1" />`
- By default 160 virtual nodes, if you want to modify, please configure `<dubbo:parameter key="hash.nodes" value="320" />`

See the algorithm at http://en.wikipedia.org/wiki/Consistent_hashing

Configuration

Server service level

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

Client service level

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

Server method level

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

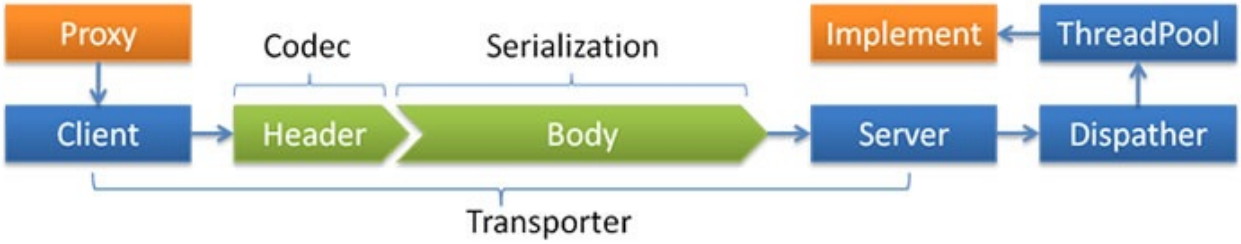
Client method level

```
<dubbo:reference interface="...">
    <dubbo:method name="..." loadbalance="roundrobin"/>
</dubbo:reference>
```

Thread Model

Thread Model

- If events handing can be executed quickly without sending new request like marking in memory. Events should be handled by I/O thread since it reduces thread dispatching.
- If event handling will be executed slowly or needs to send new I/O request like querying from database, events should be handled in thread pool. Otherwise, I/O thread will be blocked and then will be not able to receive requests.
- If events are handled by I/O thread, and send new I/O requests during the handling like sending a l login request during connect event, it will alert with “Potentially leading to deadlock”, but deadlock will not happen actually.



Thus, we need different dispatch strategies and different thread pool configurations to face difference scenarios.

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

Dispatcher

- all: All messages will be dispatched to thread pool, including request, response, connect event, disconnect event and heartbeat.
- direct: All messages will not be dispatched to thread pool and will be executed directly by I/O thread.
- message: Only request, response messages will be dispatched to I/O thread. Other messages like disconnect, connect, heartbeat messages will be executed by I/O thread.
- execution: Only request message will be dispatched to thread pool. Other messages like response, connect, disconnect, heartbeat will be directly executed by I/O thread.
- connection: I/O thread will put disconnect and connect events in the queue and execute them sequentially, other messages will be dispatched to the thread pool.

Thread pool

- fixed: A fixed size of thread pool. It creates threads when starts, never shut down.（default).
- cached: A cached thread pool. Automatically delete the thread when it’s in idle for one minute. Recreate when needed.
- limit: elastic thread pool. But it can only increase the size of the thread pool. The reason is to avoid performance issue caused by traffic spike when decrease the size of the thread pool.

Explicit target

In the development and testing environment, it is often necessary to bypass the registry and test only designated service providers. In this case, point-to-point direct connection may be required, and the service provider will ignore the list of provider registration providers. The interface A configure Point-to-point, does not affect the B interface to obtain a list from the registry.

Configure with XML

If it is online demand needs the point-to-point feature,You can configure the specified provider url at `<dubbo:reference>` .it will bypass the registry, multiple addresses separated by semicolons, the following configuration:

```
<dubbo:reference id="xxxService" interface="com.alibaba.xxx.XxxService" url="dubbo://l
ocalhost:20890" />
```

Configure with the `-D` argument

Add the -D parameter mapping service address to the JVM startup parameters：

```
java -Dcom.alibaba.xxx.XxxService=dubbo://localhost:20890
```

Configure with the `.properties` file

If you have more services, you can also use file mapping to specify the mapping file path with `-Ddubbo.resolve.file`. This configuration takes precedence over the configuration in `<dubbo: reference>`, for example:

```
java -Ddubbo.resolve.file=xxx.properties
```

Then add the configuration in the mapping file `xxx.properties`, where key is the service name and value is the service provider URL:

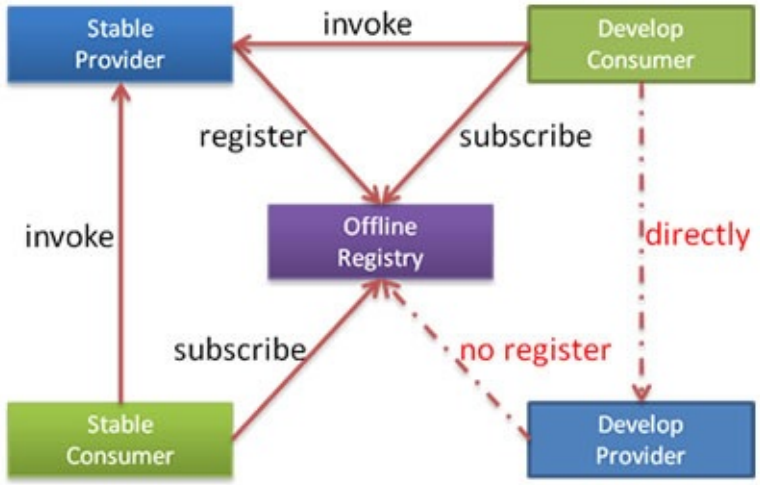
```
com.alibaba.xxx.XxxService=dubbo://localhost:20890
```

NOTE To avoid complicating the online environment, do not use this feature online and should only be used during the testing phase

Subscribe only

To facilitate the development of tests, it is common to have a registry of all services available in develop environment.And the registration of a service provider under development may affect consumers' inability to run.

You can let service provider developers only subscribe to services only (services developed may rely on other services) ,don't register services under development and testing services under development with directly connection.



User configuration:

```
<dubbo:registry address="10.20.153.10:9090" register="false" />
```

or

```
<dubbo:registry address="10.20.153.10:9090?register=false" />
```

Register only

You have two mirroring environments, two registries. You have deployed one service at only one of the registries, another registries have not had time to deploy, and other applications at both registries need to rely on the service. At this time, the service provider registers service to another registrar, but the service consumers do not consume the service from another registrar.

Disable subscription configuration

```
<dubbo:registry id="hzRegistry" address="10.20.153.10:9090" />
<dubbo:registry id="qdRegistry" address="10.20.141.150:9090" subscribe="false" />
```

or

```
<dubbo:registry id="hzRegistry" address="10.20.153.10:9090" />
<dubbo:registry id="qdRegistry" address="10.20.141.150:9090?subscribe=false" />
```

Static Service

- Sometimes we want to manually manage the registration and deregistration for service provider, we need to set registry to non-dynamoic mode.

```
<dubbo:registry address="10.20.141.150:9090" dynamic="false" />
```

Or

```
<dubbo:registry address="10.20.141.150:9090?dynamic=false" />
```

dynamic mode is disabled when service provider initially registers, then we need to enable it manually. When disconnects, the setting will not be deleted automatically, need to disable it manually.

For a third party service provider like “memcachd”, it can directly write the address information of service provider to registry, which can be used by consumer.

```
RegistryFactory registryFactory = ExtensionLoader.getExtensionLoader(RegistryFactory.c
lass).getAdaptiveExtension();
Registry registry = registryFactory.getRegistry(URL.valueOf("zookeeper://10.20.153.10:
2181"));
registry.register(URL.valueOf("memcached://10.20.153.11/com.foo.BarService?category=pr
oviders&dynamic=false&application=foo"));
```

> [1]. usually called by monitor system ↩

Multiple protocols

Dubbo allows you to configure multiple protocols, support different protocols on different services, or support multiple protocols on the same service.

Every service export to one specific protocol separately

Different protocol performance is not the same. Such as big data should use short connection protocol, small data and concurrent should use long connection protocol.

```
<?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/beanshttp://www.springfr
amework.org/schema/beans/spring-beans.xsdhttp://code.alibabatech.com/schema/dubbohttp:
//code.alibabatech.com/schema/dubbo/dubbo.xsd">
    <dubbo:application name="world"  />
    <dubbo:registry id="registry" address="10.20.141.150:9090" username="admin" passwo
rd="hello1234" />
    <!-- multiple protocols -->
    <dubbo:protocol name="dubbo" port="20880" />
    <dubbo:protocol name="rmi" port="1099" />
    <!-- Use dubbo protocol to expose the service -->
    <dubbo:service interface="com.alibaba.hello.api.HelloService" version="1.0.0" ref=
"helloService" protocol="dubbo" />
    <!-- Use rmi protocol to expose services -->
    <dubbo:service interface="com.alibaba.hello.api.DemoService" version="1.0.0" ref="
demoService" protocol="rmi" />
</beans>
```

One service export to several protocols

```
<?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/beanshttp://www.springfr
amework.org/schema/beans/spring-beans.xsdhttp://code.alibabatech.com/schema/dubbohttp:
//code.alibabatech.com/schema/dubbo/dubbo.xsd">
    <dubbo:application name="world"  />
    <dubbo:registry id="registry" address="10.20.141.150:9090" username="admin" passwo
rd="hello1234" />
    <!-- multiple protocols-->
    <dubbo:protocol name="dubbo" port="20880" />
    <dubbo:protocol name="hessian" port="8080" />
    <!-- Service exposes multiple protocols -->
    <dubbo:service id="helloService" interface="com.alibaba.hello.api.HelloService" ve
rsion="1.0.0" protocol="dubbo,hessian" />
</beans>
```

> [1]. custom protocol, see:protocol extension ↩

Multiple registries

Dubbo supports the same service to register multiple registries, or different services were registered to different registries, or even reference the same name service from different registries. In addition, the registry supports custom extensions [1] 。

One service register to multiple registries

For example: Alibaba some services are not deployed in Qingdao, only deployed in Hangzhou. While other applications in Qingdao need to reference this service, you can register your services to both registries at the same time.

```
<?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/beanshttp://www.springfr
amework.org/schema/beans/spring-beans.xsdhttp://code.alibabatech.com/schema/dubbohttp:
//code.alibabatech.com/schema/dubbo/dubbo.xsd">
    <dubbo:application name="world"  />
    <!-- Multi registries -->
    <dubbo:registry id="hangzhouRegistry" address="10.20.141.150:9090" />
    <dubbo:registry id="qingdaoRegistry" address="10.20.141.151:9010" default="false"
/>
    <!-- Service register to multiple registries -->
    <dubbo:service interface="com.alibaba.hello.api.HelloService" version="1.0.0" ref=
"helloService" registry="hangzhouRegistry,qingdaoRegistry" />
</beans>
```

Different services register to different registries

For example: Some CRM services are specifically designed for international stations, and some services are specifically designed for Chinese stations.

```
<?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/beanshttp://www.springfr
amework.org/schema/beans/spring-beans.xsdhttp://code.alibabatech.com/schema/dubbohttp:
//code.alibabatech.com/schema/dubbo/dubbo.xsd">
    <dubbo:application name="world"  />
    <!-- Multi registries -->
    <dubbo:registry id="chinaRegistry" address="10.20.141.150:9090" />
    <dubbo:registry id="intlRegistry" address="10.20.154.177:9010" default="false" />
    <!-- Service register to Chinese station registry -->
    <dubbo:service interface="com.alibaba.hello.api.HelloService" version="1.0.0" ref=
"helloService" registry="chinaRegistry" />
    <!-- Service register to international station registry -->
    <dubbo:service interface="com.alibaba.hello.api.DemoService" version="1.0.0" ref="
demoService" registry="intlRegistry" />
</beans>
```

Reference services from multiple registries

For example: CRM needs to call the PC2 service of Chinese station and international station at the same time. PC2 is deployed in both Chinese station and international station. The interfaces and version numbers are the same, but the database used is different.

```
<?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/beanshttp://www.springfr
amework.org/schema/beans/spring-beans.xsdhttp://code.alibabatech.com/schema/dubbohttp:
//code.alibabatech.com/schema/dubbo/dubbo.xsd">
    <dubbo:application name="world"  />
    <!-- Multi registries -->
    <dubbo:registry id="chinaRegistry" address="10.20.141.150:9090" />
    <dubbo:registry id="intlRegistry" address="10.20.154.177:9010" default="false" />
    <!-- Reference Chinese station service -->
    <dubbo:reference id="chinaHelloService" interface="com.alibaba.hello.api.HelloServ
ice" version="1.0.0" registry="chinaRegistry" />
    <!-- Reference international station service -->
    <dubbo:reference id="intlHelloService" interface="com.alibaba.hello.api.HelloServi
ce" version="1.0.0" registry="intlRegistry" />
</beans>
```

When testing, the service needs to be temporarily register to two registries, which can use vertical signs to separate multiple different registry addresses:

```
<?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/beanshttp://www.springfr
amework.org/schema/beans/spring-beans.xsdhttp://code.alibabatech.com/schema/dubbohttp:
//code.alibabatech.com/schema/dubbo/dubbo.xsd">
    <dubbo:application name="world"  />
    <!-- The vertical separation means that multiple registries are connected at the s
ame time. Multiple cluster addresses of the same registry are separated by commas -->
    <dubbo:registry address="10.20.141.150:9090|10.20.154.177:9010" />
    <!-- service reference -->
    <dubbo:reference id="helloService" interface="com.alibaba.hello.api.HelloService"
version="1.0.0" />
</beans>
```

> [1]. custom registry, see：registry extension ↩

Service Group

When you have multi-impls of a interface,you can distinguish them with the group.

Service

```
<dubbo:service group="feedback" interface="com.xxx.IndexService" />
<dubbo:service group="member" interface="com.xxx.IndexService" />
```

Reference

```
<dubbo:reference id="feedbackIndexService" group="feedback" interface="com.xxx.IndexSe
rvice" />
<dubbo:reference id="memberIndexService" group="member" interface="com.xxx.IndewxServi
ce" />
```

Any group [1] :

```
<dubbo:reference id="barService" interface="com.foo.BarService" group="*" />
```

> [1]. supported after version `2.2.0` ,always select only one available group of implementations to invoke. ↩

Multi versions

When an interface to achieve an incompatible upgrade, you can use the version number transition. Different versions of the services do not reference each other.

You can follow the steps below for version migration:

1. In the low pressure period, upgrade to half of the provider to the new version
2. Then upgrade all consumers to the new version
3. Then upgrade the remaining half providers to the new version

Old version of the service provider configuration:

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

New version of the service provider configuration:

```
<dubbo:service interface="com.foo.BarService" version="2.0.0" />
```

Old version of the service consumer configuration:

```
<dubbo:reference id="barService" interface="com.foo.BarService" version="1.0.0" />
```

New version of the service consumer configuration:

```
<dubbo:reference id="barService" interface="com.foo.BarService" version="2.0.0" />
```

If you do not need to distinguish between versions, can be configured as follows [1] :

```
<dubbo:reference id="barService" interface="com.foo.BarService" version="*" />
```

[1]. `2.2.0` or later support ↩

Group Merger

According to the group to invoke server and return the merge result [1], such as the menu service, the same interface, but there are a variety of implementations, using group distinction, consumers call each group and get the results, the merger can merge the resules, so that you can achieve aggregation Menu Item.

Related code can refer to dubbo project example

Configuration

Merge all groups

```
<dubbo:reference interface="com.xxx.MenuService" group="*" merger="true" />
```

Merge the specified group

```
<dubbo:reference interface="com.xxx.MenuService" group="aaa,bbb" merger="true" />
```

The specified method to merge the results, other unspecified methods, will only call one group

```
<dubbo:reference interface="com.xxx.MenuService" group="*">
    <dubbo:method name="getMenuItems" merger="true" />
</dubbo:service>
```

The Specified a method does not merge the results, others merge the results

```
<dubbo:reference interface="com.xxx.MenuService" group="*" merger="true">
    <dubbo:method name="getMenuItems" merger="false" />
</dubbo:service>
```

Specify the merge strategy, the default according to the type of return value automatically match, if the same type has two mergers, you need to specify the name of the merger[2]

```
<dubbo:reference interface="com.xxx.MenuService" group="*">
    <dubbo:method name="getMenuItems" merger="mymerge" />
</dubbo:service>
```

Specify the merge method, it will call the return type's method for merging, the merging method parameter type must be the return type

```
<dubbo:reference interface="com.xxx.MenuService" group="*">
    <dubbo:method name="getMenuItems" merger=".addAll" />
</dubbo:service>
```

1. since `2.1.0` began to support ↩
2. See also：merger extensions ↩

Parameter Validation

The parameter validation [1] is based on [JSR303] (https://jcp.org/en/jsr/detail?id=303). The user simply add the validation annotation of the JSR303 and declares the filter for validation [2].

Maven Dependency

```
<dependency>
    <groupId>javax.validation</groupId>
    <artifactId>validation-api</artifactId>
    <version>1.0.0.GA</version>
</dependency>
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-validator</artifactId>
    <version>4.2.0.Final</version>
</dependency>
```

Sample

Example of Parameter Annotation

```
import java.io.Serializable;
import java.util.Date;

import javax.validation.constraints.Future;
import javax.validation.constraints.Max;
import javax.validation.constraints.Min;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Past;
import javax.validation.constraints.Pattern;
import javax.validation.constraints.Size;

public class ValidationParameter implements Serializable {
    private static final long serialVersionUID = 71589116685680003921;

    @NotNull // Required
    @Size(min = 1, max = 20) // range
    private String name;

    @NotNull(groups = ValidationService.Save.class) // It is not allowed to be blank w
```

```
hen saving. When it is updated, it is allowed to be blank, indicating that the field i
s not updated
    @Pattern(regexp = "^\\s*\\w+(?:\\.{0,1}[\\w-]+)*@[a-zA-Z0-9]+(?:[-.][a-zA-Z0-9]+)*
\\.[a-zA-Z]+\\s*$")
    private String email;

    @Min(18) // min value
    @Max(100) // max value
    private int age;

    @Past // Must be a past time
    private Date loginDate;

    @Future // Must be a future time
    private Date expiryDate;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public Date getLoginDate() {
        return loginDate;
    }

    public void setLoginDate(Date loginDate) {
        this.loginDate = loginDate;
    }

    public Date getExpiryDate() {
        return expiryDate;
    }

    public void setExpiryDate(Date expiryDate) {
```

```
        this.expiryDate = expiryDate;
    }
}
```

Example of group validation

```
public interface ValidationService { // By default, service interfaces are used to dif
ferentiate authentication scenarios. For example：@NotNull(groups = ValidationService.c
lass)
    @interface Save{} // The same name as the method interface, the first letter capit
alized, used to distinguish between authentication scene. For example：@NotNull(groups
= ValidationService.Save.class)，option
    void save(ValidationParameter parameter);
    void update(ValidationParameter parameter);
}
```

Example of Cascading Validation

```
import javax.validation.GroupSequence;

public interface ValidationService {
    @GroupSequence(Update.class) // validate the Update group rules at the same time
    @interface Save{}
    void save(ValidationParameter parameter);

    @interface Update{}
    void update(ValidationParameter parameter);
}
```

Example of parameter validation

```
import javax.validation.constraints.Min;
import javax.validation.constraints.NotNull;

public interface ValidationService {
    void save(@NotNull ValidationParameter parameter); // Param must not be null
    void delete(@Min(1) int id); // validate the range
}
```

Configuration

Validate Parameter on the client

```
<dubbo:reference id="validationService" interface="com.alibaba.dubbo.examples.validati
on.api.ValidationService" validation="true" />
```

Validate Parameter on the server

```
<dubbo:service interface="com.alibaba.dubbo.examples.validation.api.ValidationService"
ref="validationService" validation="true" />
```

Validate Exception

```
import javax.validation.ConstraintViolationException;
import javax.validation.ConstraintViolationException;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.alibaba.dubbo.examples.validation.api.ValidationParameter;
import com.alibaba.dubbo.examples.validation.api.ValidationService;
import com.alibaba.dubbo.rpc.RpcException;

public class ValidationConsumer {
    public static void main(String[] args) throws Exception {
        String config = ValidationConsumer.class.getPackage().getName().replace('.', '
/') + "/validation-consumer.xml";
        ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext(co
nfig);
        context.start();
        ValidationService validationService = (ValidationService)context.getBean("vali
dationService");
        // Error
        try {
            parameter = new ValidationParameter();
            validationService.save(parameter);
            System.out.println("Validation ERROR");
        } catch (RpcException e) { // throw RpcException
            ConstraintViolationException ve = (ConstraintViolationException) e.getCaus
e(); // Inside a ConstraintViolationException
            Set<ConstraintViolation<?>> violations = ve.getConstraintViolations(); //
You can get the collection of validation error details
            System.out.println(violations);
        }
    }
}
```

1

> [1]. Support since `2.1.0` version. If you want to know how to use it, refer to [Sample code in dubbo project] (https://github.com/alibaba/dubbo/tree/master/dubbo-test/dubbo-test-examples/src/main/java/com/alibaba/dubbo/examples/validation) ↩

> [2]. The validation method is extensible, refer to Developer Extension in the developer's manual. ↩

Cache Result

Cache Result [1] is used to speed up access to popular data. Dubbo provides declarative caching to reduce the user work of adding cache [2] 。

Cache Type

- `lru` Delete excess cache Based on the principle of least recently used. The hottest data is cached.
- `threadlocal` The current thread cache. For example, a page have a lot of portal and each portal need to check user information, you can reduce this redundant visit with this cache.
- `jcache` integrate with JSR107 , you can bridge a variety of cache implementation 。

Caching type can be extended，refer to：Cache extension

Configuration

```
<dubbo:reference interface="com.foo.BarService" cache="lru" />
```

or：

```
<dubbo:reference interface="com.foo.BarService">
    <dubbo:method name="findBar" cache="lru" />
</dubbo:reference>
```

1. Support since `2.1.0` ↩

2. Sample ↩

Generic Reference

Generic invocation is mainly used when the client does not have API interface or model class, all POJOs in parameters and return values are represented by `Map` .Commonly used for framework integration such as: implementing a common service testing framework, all service implementations can be invoked via `GenericService` .

Use generic invocation via Spring

Declared in the Spring configuration file `generic =" true "` :

```
<dubbo:reference id="barService" interface="com.foo.BarService" generic="true" />
```

In Java code, get `barService` and start generic invocation:

```
GenericService barService = (GenericService) applicationContext.getBean("barService");
Object result = barService.$invoke("sayHello", new String[] { "java.lang.String" }, new
 Object[] { "World" });
```

Use generic invocation via API

```
import com.alibaba.dubbo.rpc.service.GenericService;
...

// reference remote service
// The instance is very heavy, which encapsulates all the registration center and serv
ice provider connection, please cache
ReferenceConfig<GenericService> reference = new ReferenceConfig<GenericService>();
// weak type service interface name
reference.setInterface("com.xxx.XxxService");
reference.setVersion("1.0.0");
// declared as generic service
reference.setGeneric(true);

// service stub type is also the com.alibaba.dubbo.rpc.service.GenericService
GenericService genericService = reference.get();

// basic types and Date, List, Map, etc. do not need conversion, direct use them
Object result = genericService.$invoke("sayHello", new String[] {"java.lang.String"},
new Object[] {"world"});

// map POJO parameters, if the return value is POJO will automatically turn into map
Map<String, Object> person = new HashMap<String, Object>();
person.put("name", "xxx");
person.put("password", "yyy");
// if the return value is POJO will automatically turn into map
Object result = genericService.$invoke("findPerson", new String[]
{"com.xxx.Person"}, new Object[]{person});

...
```

Further explanation of generalized types

Consider POJO like this：

```
package com.xxx;

public class PersonImpl implements Person {
    private String name;
    private String password;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }
}
```

The POJO data：

```
Person person = new PersonImpl();
person.setName("xxx");
person.setPassword("yyy");
```

Data represented by `Map` ：

```
Map<String, Object> map = new HashMap<String, Object>();
// Note: If the parameter type is an interface, or List lost the generic class, you ca
n specify the type of the class attribute
map.put("class", "com.xxx.PersonImpl");
map.put("name", "xxx");
map.put("password", "yyy");
```

Generic Service

The implementation of the generic interface is mainly used when there is no API interface and model class on the server side. All POJOs in the parameters and return values are represented by the Map and are usually used for framework integration. For example, to implement a universal remote service Mock framework, handle all service requests by implementing the GenericService interface.

In Java code, implement `GenericService` interface：

```
package com.foo;
public class MyGenericService implements GenericService {

    public Object $invoke(String methodName, String[] parameterTypes, Object[] args) t
hrows GenericException {
        if ("sayHello".equals(methodName)) {
            return "Welcome " + args[0];
        }
    }
}
```

Export generic implements via Spring

Declared in the Spring configuration file：

```
<bean id="genericService" class="com.foo.MyGenericService" />
<dubbo:service interface="com.foo.BarService" ref="genericService" />
```

Export generic implements via API

```
...
// use com.alibaba.dubbo.rpc.service.GenericService can replace all implements
GenericService xxxService = new XxxGenericService();

// The instance is very heavy, which encapsulates all the registration center and serv
ice provider connection, please cache
ServiceConfig<GenericService> service = new ServiceConfig<GenericService>();
// weak type service interface name
service.setInterface("com.xxx.XxxService");
service.setVersion("1.0.0");
// point to a generic serivce instance
service.setRef(xxxService);

// export service to registration center
service.export();
```

Echo Testing

Echo testing is used for check the service is available,Echo testing is performed according to the normal request flow and is able to test whether the entire call is unobstructed and can be used for monitoring.

All the services will be automatically implemented `EchoService` interface,just cast any service reference to `EchoService` to use it.

Spring configuration:

```
<dubbo:reference id="memberService" interface="com.xxx.MemberService" />
```

The java code：

```
// reference the remote service
MemberService memberService = ctx.getBean("memberService");
// case the service reference to EchoService
EchoService echoService = (EchoService) memberService;

// Echo test usability
String status = echoService.$echo("OK");

assert(status.equals("OK"));
```

Context Information

All environment information of during the current call will put into the context,and all configuration information will convert the parameters of `URL` instance,Ref to the column of **URL parameters** at the schema configuration reference book

`RpcContext` is a temporary status recorder of `ThreadLocal` ,when accept `RPC` request or send `RPC` request,The `RpcContext` will be changed.Such as: `A` call `B` and `B` call `C` . On `B` machine,before `B` call `C` ,the `RpcContext` will record the information of `A` call `B` .After `B` call `C` ,the `RpcContext` record the information of `B` call `C` .

At service consumer

```
// remote invoke
xxxService.xxx();
// if return true,then the current side is consumer.
boolean isConsumerSide = RpcContext.getContext().isConsumerSide();
// get the provider ip address of the last invoke.
String serverIP = RpcContext.getContext().getRemoteHost();
// because all configuration information has convert the URL's  parameters,so at this
place can get the application parameter value.
String application = RpcContext.getContext().getUrl().getParameter("application");
// Note:every rpc invoke,then context will be changed.
yyyService.yyy();
```

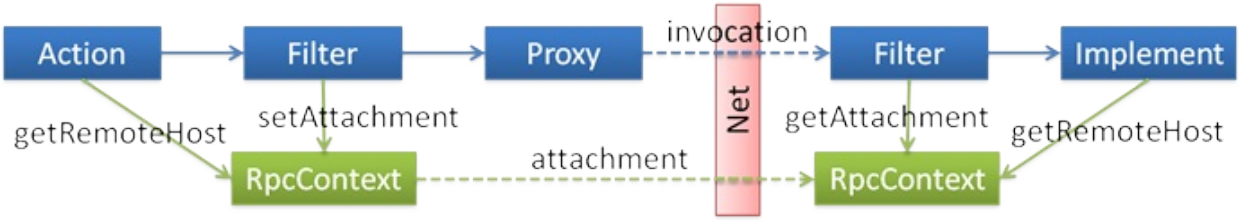
At service provider

```
public class XxxServiceImpl implements XxxService {

    public void xxx() {
        // if return true,then the current side is provider.
        boolean isProviderSide = RpcContext.getContext().isProviderSide();
        // get the invoker ip
        String clientIP = RpcContext.getContext().getRemoteHost();
        // because all configuration information has convert the URL's  parameters,so
at this place can get the application parameter value.
        String application = RpcContext.getContext().getUrl().getParameter("applicatio
n");
        // Note:every rpc invoke,then context will be changed.
        yyyService.yyy();;
    }
}
```

Implicit parameters

You can implicitly pass parameters between service consumers and providers via `setAttachment` and `getAttachment` on `RpcContext` .



Set the implicit parameters at service consumer side

Via `setAttachment` on `RpcContext` set key/value pair for implicitly pass parameters.When finished once remote invoke,will be clear,so multi-invoke must set multi-times.

```
RpcContext.getContext().setAttachment("index", "1"); // implicitly pass parameters,beh
ind the remote call will implicitly send these parameters to the server side, similar
to the cookie, for the framework of integration, not recommended for regular business
use
xxxService.xxx(); // remote call
// ...
```

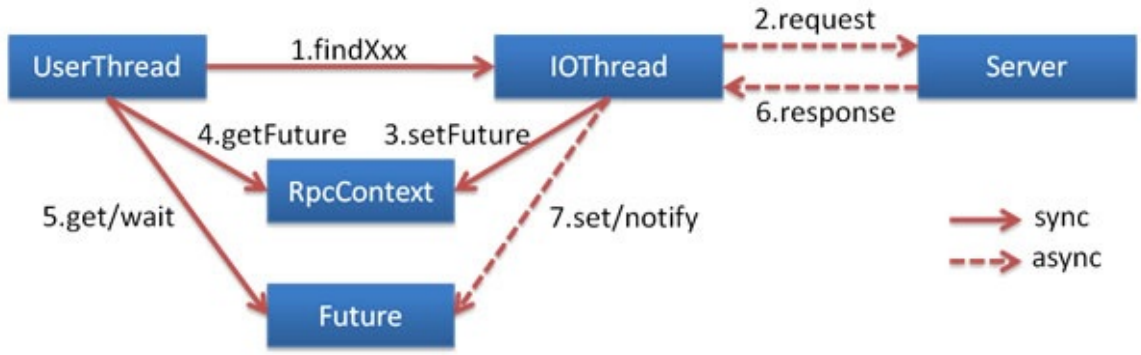
Fetch the implicit parameters at service provider side

```
public class XxxServiceImpl implements XxxService {

    public void xxx() {
        // get parameters which passed by the consumer side,for the framework of integ
ration, not recommended for regular business use
        String index = RpcContext.getContext().getAttachment("index");
    }
}
```

Asynchronous Call

As dubbo is based on a non-blocking NIO network layer, the client can start parallel call to multiple remote services without explicitly starting mulithreads, which costs relatively fewer resources.



You can config at `consumer.xml` for setup asynchronous call some remote service.

```
<dubbo:reference id="fooService" interface="com.alibaba.foo.FooService">
      <dubbo:method name="findFoo" async="true" />
</dubbo:reference>
<dubbo:reference id="barService" interface="com.alibaba.bar.BarService">
      <dubbo:method name="findBar" async="true" />
</dubbo:reference>
```

Configure the above configuration information,you can invoke the remote service in your code.

```
// the invoke will return null immediately
fooService.findFoo(fooId);
// get current invoke Future instance,when the remote service has return result,will n
otify this Future instance.
Future<Foo> fooFuture = RpcContext.getContext().getFuture();

// the invoke will return null immediately
barService.findBar(barId);
// get current invoke Future instance,when the remote service has return result,will n
otify this Future instance.
Future<Bar> barFuture = RpcContext.getContext().getFuture();

// now the request of findFoo and findBar was executed at same time,The client not nee
d setup multithreading for parallel call, which is NIO-based non-blocking implementati
on of parallel calls

// Current thread will be blocking,and wait findFoo has return. when remote service ha
s return findFoo result,the current thread will be wake up.
Foo foo = fooFuture.get();
// same to findFoo
Bar bar = barFuture.get();

// if findFoo expend five second for wait remote service  return result,and findBar ex
pend six second. Actually,only expend six second will get findFoo and findBar result,a
nd proceed to the next step.
```

You can also set whether to wait for the message to be sent:

- `sent="true"` wait for the message to be send,if send failure，will throw exception.
- `sent="false"` do not wait for the message to be send,when the message will push into io queue,will return immediately.

The Example:

```
<dubbo:method name="findFoo" async="true" sent="true" />
```

if you only want to asynchronous call,and don't care the return.you can config `return="false"` ,To reduce the cost of creating and managing Future objects.

```
<dubbo:method name="findFoo" async="true" return="false" />
```

Note `2.0.6+` version supported.

Local call

The local call uses the `injvm` protocol, a pseudo-protocol that does not turn on the port, does not initiate remote calls, is directly associated within the JVM, but executes the Dubbo Filter chain.

Configuration

Configure `injvm` protocol

```
<dubbo:protocol name="injvm" />
```

Configure default provider

```
<dubbo:provider protocol="injvm" />
```

Configure default service

```
<dubbo:service protocol="injvm" />
```

Use injvm first

```
<dubbo:consumer injvm="true" .../>
<dubbo:provider injvm="true" .../>
```

or

```
<dubbo:reference injvm="true" .../>
<dubbo:service injvm="true" .../>
```

Note: Both service provider and service references need to declare `injvm="true"`

Automatically exposed, local service references

`2.2.0` or later, each service is exposed locally by default. When referring to the service, the local service is referenced by default. If you want to reference a remote service, you can use the following configuration to force a reference to a remote service.

```
<dubbo:reference ... scope="remote" />
```

Callback parameter

The parameter callback is the same as calling a local callback or listener, just declare which parameter is a callback type in Spring's configuration file, and Dubbo will generate a reverse proxy based on the long connection so that client logic can be called from the server.Can ref to Sample code in the dubbo project.

Example of service interface

CallbackService.java

```
package com.callback;

public interface CallbackService {
    void addListener(String key, CallbackListener listener);
}
```

CallbackListener.java

```
package com.callback;

public interface CallbackListener {
    void changed(String msg);
}
```

Example of service provider interface implementation

```
package com.callback.impl;

import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;

import com.callback.CallbackListener;
import com.callback.CallbackService;

public class CallbackServiceImpl implements CallbackService {

    private final Map<String, CallbackListener> listeners = new ConcurrentHashMap<Stri
ng, CallbackListener>();

    public CallbackServiceImpl() {
        Thread t = new Thread(new Runnable() {
            public void run() {
                while(true) {
                    try {
                        for(Map.Entry<String, CallbackListener> entry : listeners.entr
ySet()){
                            try {
                                entry.getValue().changed(getChanged(entry.getKey()));
                            } catch (Throwable t) {
                                listeners.remove(entry.getKey());
                            }
                        }
                        Thread.sleep(5000); // Timed trigger change notification
                    } catch (Throwable t) { // Defense fault tolerance
                        t.printStackTrace();
                    }
                }
            }
        });
        t.setDaemon(true);
        t.start();
    }

    public void addListener(String key, CallbackListener listener) {
        listeners.put(key, listener);
        listener.changed(getChanged(key)); // send change notification
    }

    private String getChanged(String key) {
        return "Changed: " + new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(new Da
te());
    }
}
```

Example of service provider configuration

```
<bean id="callbackService" class="com.callback.impl.CallbackServiceImpl" />
<dubbo:service interface="com.callback.CallbackService" ref="callbackService" connecti
ons="1" callbacks="1000">
    <dubbo:method name="addListener">
        <dubbo:argument index="1" callback="true" />
        <!--also can via specified argument type-->
        <!--<dubbo:argument type="com.demo.CallbackListener" callback="true" />-->
    </dubbo:method>
</dubbo:service>
```

Example of service consumer configuration

```
<dubbo:reference id="callbackService" interface="com.callback.CallbackService" />
```

Example of service consumer call

```
ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext("classpath
:consumer.xml");
context.start();

CallbackService callbackService = (CallbackService) context.getBean("callbackService")
;

callbackService.addListener("http://10.20.160.198/wiki/display/dubbo/foo.bar", new Cal
lbackListener(){
    public void changed(String msg) {
        System.out.println("callback1:" + msg);
    }
});
```

NOTE `2.0.6+` version supported.

Event Notify

Before calling, after calling, when an exception occurs,will trigger `oninvoke`, `onreturn`, `onthrow` events.You can configure which method to notify when an event occurs.

Service Interface

```
interface IDemoService {
    public Person get(int id);
}
```

Service provider implement the service.

```
class NormalDemoService implements IDemoService {
    public Person get(int id) {
        return new Person(id, "charles`son", 4);
    }
}
```

Service provider configure the service which it provided.

```
<dubbo:application name="rpc-callback-demo" />
<dubbo:registry address="http://10.20.160.198/wiki/display/dubbo/10.20.153.186" />
<bean id="demoService" class="com.alibaba.dubbo.callback.implicit.NormalDemoService" />

<dubbo:service interface="com.alibaba.dubbo.callback.implicit.IDemoService" ref="demoS
ervice" version="1.0.0" group="cn"/>
```

Declare the Callback interface at service consumer-side.

```
interface Notify {
    public void onreturn(Person msg, Integer id);
    public void onthrow(Throwable ex, Integer id);
}
```

Implement the Callback at service consumer-side.

```
class NotifyImpl implements Notify {
    public Map<Integer, Person>    ret    = new HashMap<Integer, Person>();
    public Map<Integer, Throwable> errors = new HashMap<Integer, Throwable>();

    public void onreturn(Person msg, Integer id) {
        System.out.println("onreturn:" + msg);
        ret.put(id, msg);
    }

    public void onthrow(Throwable ex, Integer id) {
        errors.put(id, ex);
    }
}
```

Configure the Callback at service consumer-side.

```
<bean id ="demoCallback" class = "com.alibaba.dubbo.callback.implicit.NofifyImpl" />
<dubbo:reference id="demoService" interface="com.alibaba.dubbo.callback.implicit.IDemo
Service" version="1.0.0" group="cn" >
      <dubbo:method name="get" async="true" onreturn = "demoCallback.onreturn" onthrow=
"demoCallback.onthrow" />
</dubbo:reference>
```

`callback` and `async` functions are orthogonally decomposed. `async = true` means that the result is returned immediately. `onreturn` means that a callback is required.

There are several situations with the tow attributes[2].

- Asynchronous callback mode: `async=true onreturn="xxx"`
- Synchronous callback mode: `async=false onreturn="xxx"`
- Asynchronous no callback: `async=true`
- Synchronous no callback: `async=true`

Testing code

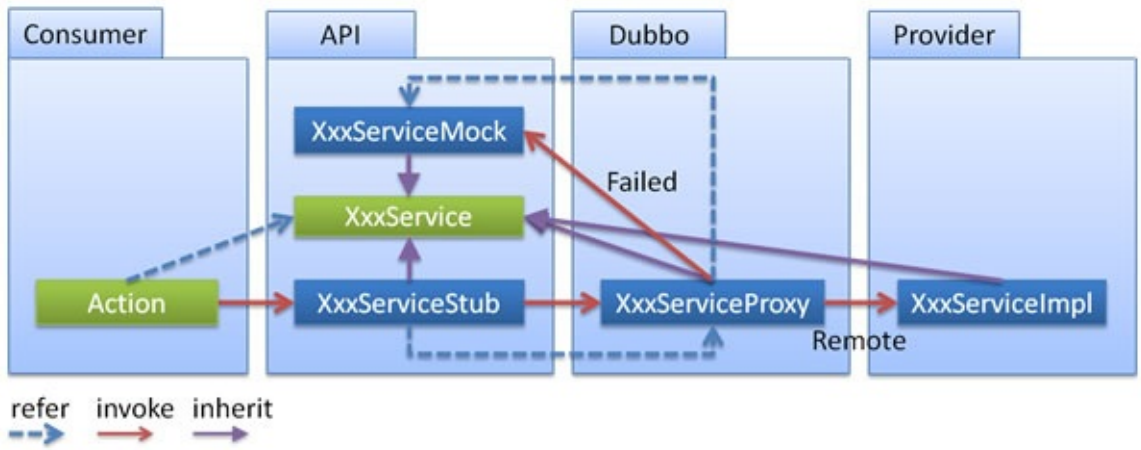
```
IDemoService demoService = (IDemoService) context.getBean("demoService");
NofifyImpl notify = (NofifyImpl) context.getBean("demoCallback");
int requestId = 2;
Person ret = demoService.get(requestId);
Assert.assertEquals(null, ret);
//for Test:Just used to illustrate the normal callback callback, the specific business
 decisions.
for (int i = 0; i < 10; i++) {
    if (!notify.ret.containsKey(requestId)) {
        Thread.sleep(200);
    } else {
        break;
    }
}
Assert.assertEquals(requestId, notify.ret.get(requestId).getId());
```

NOTE `2.0.7+` version, `async=false` is default.

Local stub

When using rpc, the client usually only the interface, but sometimes the client also want to perform part of the logic in the client. For example: do ThreadLocal cache, verify parameters, return mock data when call fails., etc.

To solve this problem, you can configure the stub in the API, so that when the client generates the proxy instance, it passes the proxy to the `Stub` via the constructor [1], and then you can implement your logic in the stub implementation code.



Configured in the spring configuration file as follows:

```
<dubbo:service interface="com.foo.BarService" stub="true" />
```

or

```
<dubbo:service interface="com.foo.BarService" stub="com.foo.BarServiceStub" />
```

Provide Stub implementation [2] :

```
package com.foo;
public class BarServiceStub implements BarService {
    private final BarService barService;

    // The real remote proxy object is passed in through the constructor
    public (BarService barService) {
        this.barService = barService;
    }

    public String sayHello(String name) {
        // The following code is executed on the client. You can do local ThreadLocal
caching on the client side, or verify parameters, etc.
        try {
            return barService.sayHello(name);
        } catch (Exception e) {
            // You can return the mock data.
            return "MockData";
        }
    }
}
```

1. The Stub must have a constructor that can pass in the proxy. ↩

2. BarServiceStub implements BarService ，it has a constructor passed in the remote BarService instance ↩

Local mock

Local mock [1] is usually used for service downgrade, such as a verification service, the client does not throw an exception when the service provider hangs up all the time, but returns the authorization failed through the Mock data.

Configured in the spring configuration file as follows:

```
<dubbo:service interface="com.foo.BarService" mock="true" />
```

or

```
<dubbo:service interface="com.foo.BarService" mock="com.foo.BarServiceMock" />
```

Mock implementation in the project [2] :

```
package com.foo;
public class BarServiceMock implements BarService {
    public String sayHello(String name) {
        // You can return mock data, this method is only executed when an RpcException
 is thrown.
        return "mock data";
    }
}
```

If the service consumer often needs `try-catch` to catch exceptions, such as:

```
Offer offer = null;
try {
    offer = offerService.findOffer(offerId);
} catch (RpcException e) {
   logger.error(e);
}
```

Consider changing to Mock implementation and return null in Mock implementation. If you just want to simply ignore the exception, `2.0.11` version or later version is available:

```
<dubbo:service interface="com.foo.BarService" mock="return null" />
```

> [1]. Mock is a subset of the Stub. If you use Stub, you may need to rely on the RpcException class. If you use Mock, you do not need to rely on RpcException, when throwing RpcException, it will callback Mock implementation class. ↩

> [2]. BarServiceMock implements BarService and has a no-argument constructor. ↩

Delay publish service

If your service need time to warm up.such as:initialization cache,or another reference resources has to be ready.so you can use the delay feature for delay publish service.

Delay five second publish

```
<dubbo:service delay="5000" />
```

Delay until Spring initialization is complete before exposing the service

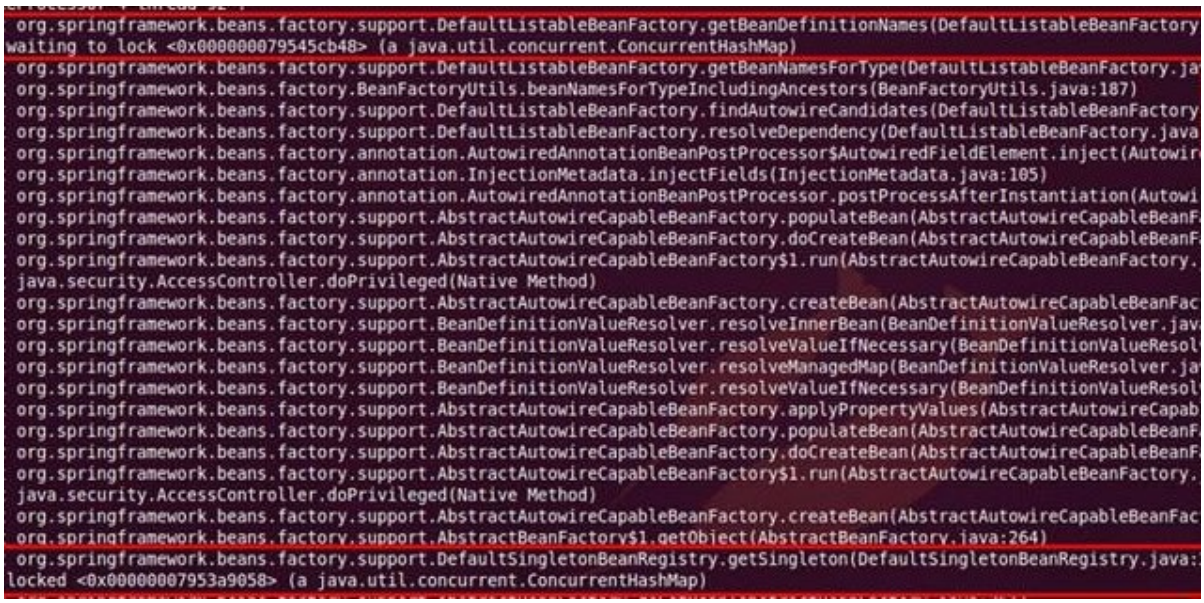
```
<dubbo:service delay="-1" />
```

The initialization deadlock problem of Spring 2.x

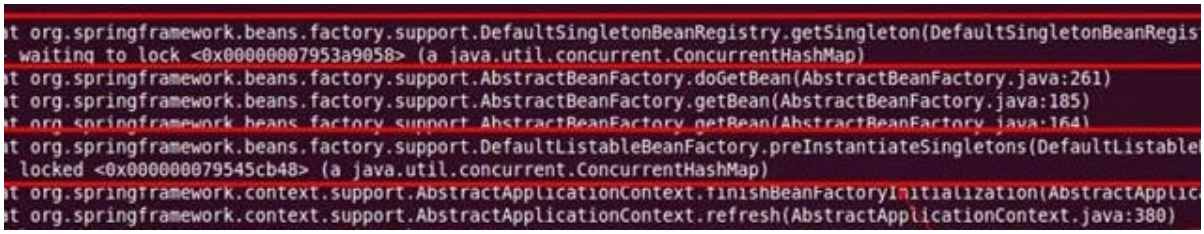
Trigger condition

The service has already published when `Spring` parse the `<dubbo:service />` element,but the `Spring` is still initializing other beans.If there is a request coming in, and the service implementation class has a call to `applicationContext.getBean ()` usage.

1. Request thread applicationContext.getBean() call, the first synchronization `singletonObjects` determine whether the existence of the bean, the synchronization does not exist to initialize the `beanDefinitionMap` , and re-synchronize `singletonObjects` write Bean instance cache.



2. But the `Spring` initialization thread,because need to determine the `Bean` is exist,Directly synchronize beanDefinitionMap to initialize, and synchronize singletonObjects write Bean instance cache.



This will cause the getBean thread to lock the singletonObjects first, then lock the beanDefinitionMap, and lock the singletonObjects again.The Spring initialization thread, the first lock beanDefinitionMap, then lock singletonObjects. Reverse lock thread deadlock, can not provide services, can not start.

Avoid ways

1. It is highly recommended not to call applicationContext.getBean() in the service implementation class, all using Spring's beans using IoC injection.
2. If you really want to tune getBean(), you can put the configuration of Dubbo Spring final loading.
3. If you do not want to rely on the configuration order, you can use `<dubbo:provider deplay ="-1"/>` to make Dubbo expose the service after the Spring container has been initialized.
4. If you use getBean() extensively, the equivalent of degenerating Spring to factory mode is to isolate Dubbo's service from a separate Spring container.

> [1]. Base on the `ContextRefreshedEvent` event of the Spring to trigger publish service. ↩

Parallel control

Example of configuration

- Example 1: Control the concurrency of all method for a specified service interface at server-side

Limit each method of `com.foo.BarService` to no more than 10 concurrent server-side executions (or take up thread pool threads):

```
<dubbo:service interface="com.foo.BarService" executes="10" />
```

- Example 2: Control the concurrency of specified method for a specified service interface at server-side

Limit the `sayHello` method of `com.foo.BarService` to no more than 10 concurrent server-side executions(or take up thread pool threads):

```
<dubbo:service interface="com.foo.BarService">
    <dubbo:method name="sayHello" executes="10" />
</dubbo:service>
```

- Example 3: Control the concurrency of all method for a specified service interface at client-side Limit each method of `com.foo.BarService` to no more than 10 concurrent client-side executions (or take up thread pool threads):

  ```
  <dubbo:service interface="com.foo.BarService" actives="10" />
  ```

  OR

  ```
  <dubbo:reference interface="com.foo.BarService" actives="10" />
  ```

- Example 4: Control the concurrency of specified method for a specified service interface at client-side Limit the `sayHello` method of `com.foo.BarService` to no more than 10 concurrent client-side executions(or take up thread pool threads):

```
<dubbo:service interface="com.foo.BarService">
  <dubbo:method name="sayHello" actives="10" />
</dubbo:service>
```

OR

```
<dubbo:reference interface="com.foo.BarService">
    <dubbo:method name="sayHello" actives="10" />
</dubbo:service>
```

If `<dubbo:service>` and `<dubbo:reference>` are both configured with `actives` , `<dubbo:reference>` is preferred.Ref to:Configuration coverage strategy.

Load Balance

You can config the `loadbalance` attribute with `leastactive` at server-side or client-side,then the framework will make consumer call the minimum number of concurrent one.

```
<dubbo:reference interface="com.foo.BarService" loadbalance="leastactive" />
```

OR

```
<dubbo:service interface="com.foo.BarService" loadbalance="leastactive" />
```

Config connections

Control connections at server-side

Limit server-side accept to no more than 10 connections

```
<dubbo:provider protocol="dubbo" accepts="10" />
```

OR

```
<dubbo:protocol name="dubbo" accepts="10" />
```

Control connections at client-side

Limit client-side creating connection to no more than 10 connections for interface `com.foo.BarService` .

```
<dubbo:reference interface="com.foo.BarService" connections="10" />
```

OR

```
<dubbo:service interface="com.foo.BarService" connections="10" />
```

NOTE: If used default protocol(`dubbo` protocol), and the value of `connections` attribute is great than 0,then each service reference will has itself connection,else all service which belong to same remote server will share only one connection. In this framework,we called `private` connection or `share` connection.

If `<dubbo:service>` and `<dubbo:reference>` are both configured accepts/connections, `<dubbo:reference>` is preferred,Ref to Configuration coverage strategy.

- : Because connection is connect on Server,so configure at Provider.

Lazy Connect

Lazy connect can reduce the number of keep-alive connections. When a call is initiated, create a keep-alive connection.[1]

```
<dubbo:protocol name="dubbo" lazy="true" />
```

> [1]. Note: This configuration takes effect only for dubbo protocols that use keep-alive connections. ↩

stickiness

Sticky connections are used for stateful services, as much as possible so that clients always make calls to the same provider, unless the provider hangs up and connects to the other one.

Sticky connections will automatically open Delayed Connections to reduce the number of long connections.

```
<dubbo:protocol name="dubbo" sticky="true" />
```

Token Authorization

Through the token authorization control center at the registry to decide whether to issue tokens to consumers, you can prevent consumers from bypassing the registry access provider, another through the registry can flexibly change the authorization without modification or upgrade provider

You can turn on token authentication globally:

```
<!--Random token , generated using a UUID-->
<dubbo:provider interface="com.foo.BarService" token="true" />
```

or

```
<!--Fixed token, equivalent to the password-->
<dubbo:provider interface="com.foo.BarService" token="123456" />
```

Of course can turn on token authentication at service level:

```
<!--Random token , generated using a UUID-->
<dubbo:service interface="com.foo.BarService" token="true" />
```

or

```
<!--Fixed token, equivalent to the password-->
<dubbo:service interface="com.foo.BarService" token="123456" />
```

Also can turn on token authentication at protocol level:

```
<!--Random token , generated using a UUID-->
<dubbo:protocol name="dubbo" token="true" />
```

or

```
<!--Fixed token, equivalent to the password-->
<dubbo:protocol name="dubbo" token="123456" />
```

Routing Rules

The routing rules [1] determine the target server of one service call. It has two kinds of routing rules: conditional routing rules and script routing rules. It also support extension[2].

Write Routing Rules

Writing routing rules to the registry is usually done by the monitoring center or the console page.

```
RegistryFactory registryFactory = ExtensionLoader.getExtensionLoader(RegistryFactory.c
lass).getAdaptiveExtension();
Registry registry = registryFactory.getRegistry(URL.valueOf("zookeeper://10.20.153.10:
2181"));
registry.register(URL.valueOf("condition://0.0.0.0/com.foo.BarService?category=routers
&dynamic=false&rule=" + URL.encode("host = 10.20.153.10 => host = 10.20.153.11") + "))
;
```

其中：

- `condition://` It indicates the type of routing rules, supports routing rules and script routing rules, and can be extended. **Required**。
- `0.0.0.0` It indicates that all IP addresses are valid. If you want to take effect for only one IP address, fill in the IP address. **Required**。
- `com.foo.BarService` It indicates that the specified service is effective. **Required**。
- `category=routers` It indicates that the data is a dynamic configuration type. **Required**。
- `dynamic=false` It indicates that it is persistent data. When the registrant exits, the data is still stored in the registry. **Required**。
- `enabled=true` It indicates whether this routing rules is effective. Option, and default effective.
- `force=false` It indicates whether it is forced to be executed when the routing result is null. If it is not enforced, the route will be automatically invalidated. Option, and default `false`.
- `runtime=false` It indicates whether to execute routing rules at every call. If not, the result is only pre-executed and cached when the provider's address list changes. It will get routing result from cache when the service is invoked. If you use parameter routing, you must to configure it as `true`. Be careful that the configuration will affect the performance. Option, and default `false`.
- `priority=1` The priority of the routing rules. it is used for sorting, the greater the

priority, the more front execution. Option, and default `0` 。

- `rule=URL.encode("host = 10.20.153.10 => host = 10.20.153.11")` It indicates the content of routing rule，**Required**。

Conditional routing rules

Routing rules based on conditional expressions, such as：`host = 10.20.153.10 => host = 10.20.153.11`

Rules：

- The previous of `=>` is matched condition for consumer. All parameters compare with URL of consumers. When the consumer meet the condition, it will continue to execute the behind filter rules for consumer.
- After the `=>` aims to filter the provider address list. All the parameters are compared against the provider's URL, and consumer get only the filtered address list at finally.
- If the previous condition for consumer is empty, it means all consumers can matched. such as：`=> host != 10.20.153.11`
- If the filter condition for provider is empty, it means it is forbidden to visit. such as：`host = 10.20.153.10 =>`

Expressions：

Parameter Support：

- Service call information, such as: method, argument etc. Parameter routing is currently not supported
- URL field (On URL own), such as: protocol, host, port etc.
- All parameters on the URL. such as: application, organization etc.

Condition Support：

- Equal sign `=` indicates match. such as: `host = 10.20.153.10`
- Not equal sign `!=` indicates "does not match". such as: `host != 10.20.153.10` .

Value Support：

- Separate multiple values with a comma `,` . Such as：`host != 10.20.153.10,10.20.153.11`
- End with `*` to indicate wildcard. Such as： `host != 10.20.*`
- Start with `$` to indicate reference to consumer parameters. Such as: `host = $host`

Samples

1. Exclude pre-release machine：

```
=> host != 172.22.3.91
```

2. Whitelist [3]：

```
host != 10.20.153.10,10.20.153.11 =>
```

3. Blacklist：

```
host = 10.20.153.10,10.20.153.11 =>
```

4. Service boarding application only expose part of the machine to prevent the entire cluster hanging up:

```
=> host = 172.22.3.1*,172.22.3.2*
```

5. Additional machines for important applications：

```
application != kylin => host != 172.22.3.95,172.22.3.96
```

6. Read and write separation：

```
method = find*,list*,get*,is* => host = 172.22.3.94,172.22.3.95,172.22.3.96
method != find*,list*,get*,is* => host = 172.22.3.97,172.22.3.98
```

7. Separation of Front and Background Application：

```
application = bops => host = 172.22.3.91,172.22.3.92,172.22.3.93
application != bops => host = 172.22.3.94,172.22.3.95,172.22.3.96
```

8. Isolate different network segments：

```
host != 172.22.3.* => host != 172.22.3.*
```

9. Providers and consumers deployed in the same cluster, the machine only visit the local service：

```
=> host = $host
```

Script routing rules

Script routing rules [4] support all scripts of JDK script engine. such as: javascript, jruby, groovy, etc. Configure the script type by `type=javascript`, the default is javascript.

```
"script://0.0.0.0/com.foo.BarService?category=routers&dynamic=false&rule=" + URL.encod
e("function route(invokers) { ... } (invokers)")
```

Routing rules that base on script engine is as follow：

```
function route(invokers) {
    var result = new java.util.ArrayList(invokers.size());
    for (i = 0; i < invokers.size(); i ++) {
        if ("10.20.153.10".equals(invokers.get(i).getUrl().getHost())) {
            result.add(invokers.get(i));
        }
    }
    return result;
} (invokers); // Indicates that the method is executed immediately
```

1. Support since `2.2.0` ↩

2. Routing Rules Extension Point: Route Extension ↩

3. Note: A service can only have one whitelist rule, otherwise the two rules will be filtered out. ↩

4. Note: Scripts have no sandbox constraints, can execute arbitrary code, and poses a backdoor risk. ↩

Configure rule

Write then dynamic configuration to the registry center,This feature is usually done by the monitoring center or the center's page.

```
RegistryFactory registryFactory = ExtensionLoader.getExtensionLoader(RegistryFactory.c
lass).getAdaptiveExtension();
Registry registry = registryFactory.getRegistry(URL.valueOf("zookeeper://10.20.153.10:
2181"));
registry.register(URL.valueOf("override://0.0.0.0/com.foo.BarService?category=configur
ators&dynamic=false&application=foo&timeout=1000"));
```

In the config override url：

- `override://` Indicates that the data is overwritten,support `override` and `absent` ，can extends，**Required**.
- `0.0.0.0` Indicates that the configurations is valid for all IP addresses，If only want to overwritten specified ip data,you can replace that specified ip address.**Required**.
- `com.foo.BarService` Indicates that is valid for specified service,**Required**.
- `category=configurators` Indicates that the data is dynamic configuration,**Required**。
- `dynamic=false` Indicates that the data is persistent,When the registered party withdraws,the data is still stored in the registry **Required**。
- `enabled=true` override strategy is enable,can absent,if absent,then enable.
- `application=foo` Indicates that is valid for specified application,can absent,if absent,then valid for all application.
- `timeout=1000` Indicates that the value of the `timeout` parameter that satisfies the above conditions is overwritten by 1000,if want to override another parameters, add directly to the `override` URL parameter.

Example：

1. Disable service provider.(Usually used to temporarily kick off a provider machine, similar to the prohibition of consumer access, please use the routing rules)

   ```
   override://10.20.153.10/com.foo.BarService?category=configurators&dynamic=false&d
   isbaled=true
   ```

2. Adjustment weight:(Usually used to capacity assessment,default is 100)

   ```
   override://10.20.153.10/com.foo.BarService?category=configurators&dynamic=false&w
   eight=200
   ```

3. Adjustment load balance strategy.(default random)

```
 override://10.20.153.10/com.foo.BarService?category=configurators&dynamic=false&l
oadbalance=leastactive
```

4. Service downgrade:(Usually used to temporarily mask an error of non-critical services)

```
 override://0.0.0.0/com.foo.BarService?category=configurators&dynamic=false&applic
ation=foo&mock=force:return+null
```

NOTE: `2.2.0+` version supported.

Service-Downgrade

You can temporarilly shield a non-critical service through the service downgrade and define the return policy for it.

Publish dynamic configuration rule to the registry:

```
RegistryFactory registryFactory = ExtensionLoader.getExtensionLoader(RegistryFactory.c
lass).getAdaptiveExtension();
Registry registry = registryFactory.getRegistry(URL.valueOf("zookeeper://10.20.153.10:
2181"));
registry.register(URL.valueOf("override://0.0.0.0/com.foo.BarService?category=configur
ators&dynamic=false&application=foo&mock=force:return+null"));
```

- The configuration `mock=force:return+null` means that all calls of this service will return null value directly,without making remote calls.Usually used to reduce the effect of some slow non-critical services.
- Also you can change that configuration to `mock=fail:return+null`.Then you will get null value after a failed call.Consumer will try to make a remote call to get the truely result if succeed,and if the call failed you will get null value.Usually used to tolerate some non-critical services.

[1]. supported after version `2.2.0` ↩

Graceful Shutdown

Dubbo is graceful shutdown through the `ShutdownHook` of the JDK, so graceful shutdowns are not performed if you force shutdown the command, such as `kill -9 PID`, and will only be executed if `kill PID` is passed.

Howto

Service provider

- When stop, first marked as not receiving new requests, the new request directly return the error, so that the client retries other machines.
- Then check thread pool thread is running, if any, waiting for all threads to complete execution, unless overtime, then forced to close.

Service consumer

- When stop, No longer initiate a new request, all request on the client that got an error.
- Then check the request has not returned the response, waiting for the response to return, unless overtime, then forced to close.

Configuration shutdown wait time

Set graceful shutdown timeout, the default timeout is 10 seconds, if the overtime is forced to close.

```
# dubbo.properties
dubbo.service.shutdown.wait=15000
```

If ShutdownHook does not take effect, you can call it yourself, **in tomcat, it is recommended by extending the ContextListener and call the following code for graceful shutdown**：

```
ProtocolConfig.destroyAll();
```

Hostname Binding

Lookup order

Default host IP lookup order：

- Get local address via `LocalHost.getLocalHost()`.
- If it is `127.*` loopback address, then scan the network for host IP

Host configuration

Registered address if it is not correct, such as the need to register public address, you can do this:

1. edit `/etc/hosts` : add machinename and public ip, such as:

   ```
   test1 205.182.23.201
   ```

2. in `dubbo.xml` add host address configuration:

   ```
   <dubbo:protocol host="205.182.23.201">
   ```

3. or config that in `dubbo.properties` :

   ```
   dubbo.protocol.host=205.182.23.201
   ```

Port configuration

The default port and protocol:

Protocol	Port
dubbo	20880
rmi	1099
http	80
hessian	80
webservice	80
memcached	11211
redis	6379

You can configure the port as follows:

1. in `dubbo.xml` add port configuration:

```
<dubbo:protocol name="dubbo" port="20880">
```

2. or config that in `dubbo.properties` :

```
dubbo.protocol.dubbo.port=20880
```

Logger adapter

`2.2.1` or later, dubbo support log4j、slf4j、jcl、jdk adapters [1], you can also explicitly configure the log output policy in the following ways:

1. Command

```
java -Ddubbo.application.logger=log4j
```

2. Configure in `dubbo.properties`

```
dubbo.application.logger=log4j
```

3. Configure in `dubbo.xml`

```
<dubbo:application logger="log4j" />
```

> [1]. Custom Extensionslogger-adapter ↩

Access Log

If you want to logging the access information for each provide service,you can turn on the `accesslog` switch,which like the access log of `Apache` .

Note: The size of the access log maybe too much,please check the disk capacity. Now I will show you how to config the access log.

Logging by logging framework

```
<dubbo:protocol accesslog="true" .../>
```

The above configuration will turn on `accesslog` switch for all provide services,and logging the access log with logging framework(log4j/logback/slf4j...).You can config the logging framework of `logger` and `appender` for logging the access log.The simplest way is config logger name with `dubbo.accesslog` . The Example:

```
<appender name="accesslogAppender" class="ch.qos.logback.core.rolling.RollingFileAppen
der">
        <file>${loggingRoot}/accesslog/logging.log</file>
        <encoding>${loggingCharset}</encoding>
        <append>true</append>
        <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
            <FileNamePattern>${loggingRoot}/accesslog/%d{yyyyMMdd}/logging.log.%d{yyyy
MMdd}%i.gz
            </FileNamePattern>
            <MaxHistory>15</MaxHistory>
            <TimeBasedFileNamingAndTriggeringPolicy class="ch.qos.logback.core.rolling
.SizeAndTimeBasedFNATP">
                <MaxFileSize>1024MB</MaxFileSize>
            </TimeBasedFileNamingAndTriggeringPolicy>
        </rollingPolicy>
        <layout class="ch.qos.logback.classic.PatternLayout">
            <pattern><![CDATA[%level|%d{yyyy-MM-dd HH:mm:ss}|%m%n]]></pattern>
        </layout>
</appender>
<logger name="dubbo.accesslog" level="INFO" additivity="false">
        <appender-ref ref="accesslogAppender"/>
</logger>
```

The above is the demonstration of logback framework.Other logging framework is same too.It will logging the access log of all provide services into single file(`accesslog/logging.log`). And you can also config the access log of each provide service to logging separately,Only change `name` attribute of the `logger` tag,set the `name` attribute to `dubbo.accesslog.serviceInterfaceClassFullName` .The Example:

```
<logger name="dubbo.accesslog.com.dubbo.FooServiceInterface" level="INFO" additivity="
false">
        <appender-ref ref="fooServiceAccesslogAppender"/>
</logger>
```

If you only want logging the access log of specified provide service,but not all services.It's supported too.The Example:

```
<dubbo:service accesslog="true" .../>
```

Logging by specified file path

You can specify the file path with the `accesslog` attribute.The Example:

```
<dubbo:protocol accesslog="/home/admin/logs/service/accesslog.log" .../>
```

OR

```
<dubbo:service accesslog="/home/admin/logs/service/accesslog.log" .../>
```

Service container

The service container is a standalone launcher because the backend service does not require the functionality of a Web container ,such as Tomcat or JBoss. If you insist on using web containers to load service providers, that increase complexity and is waste of resources.

The service container is just a simple Main method and loads a simple Spring container to expose the service.

The content of Service container can be extended, built-in spring, jetty, log4j etc.. This can be expanded with Container Extension Points. Configure it with the -D parameter in the java command or `dubbo.properties` .

Container type

Spring Container

- Automatically load all spring configurations in the `META-INF/spring` .

```
dubbo.spring.config=classpath*:META-INF/spring/*.xml
```

Jetty Container

- Start an embedded Jetty for reporting status.
- Configure:
  - `dubbo.jetty.port=8080` : configure jetty start up port
  - `dubbo.jetty.directory=/foo/bar` : static file that can be visited by jetty directly.
  - `dubbo.jetty.page=log,status,system` : configure the displayed page, loading all pages by default

Log4j Container

- Automatic configuration log4j configuration. At the start of the multi-process, log files automatically by process sub-directory.
- Configure:
  - `dubbo.log4j.file=/foo/bar.log` : configure log file path
  - `dubbo.log4j.level=WARN` : configure log level

- `dubbo.log4j.subdirectory=20880` : configure log sub directory for multi-process startup and avoiding conflict

Container startup

load spring by default.

```
java com.alibaba.dubbo.container.Main
```

Load the container that passed in by the main method

```
java com.alibaba.dubbo.container.Main spring jetty log4j
```

Load the container that passed in by the JVM option.

```
java com.alibaba.dubbo.container.Main -Ddubbo.container=spring,jetty,log4j
```

Load the container that passed in by `dubbo.properties` in the classpath.

```
dubbo.container=spring,jetty,log4j
```

ReferenceConfig Cache

The instance of `ReferenceConfig` is heavy. It encapsulates the connection to the registry and the connection to the provider, so it need to be cached. Otherwise, repeatedly generating `ReferenceConfig` may cause performance problems , memory and connection leaks. This problem is easy to ignored when programming in API mode.

Therefore, since `2.4.0` , dubbo provides a simple utility ReferenceConfigCache for caching instances of `ReferenceConfig` .

Use as follows：

```
ReferenceConfig<XxxService> reference = new ReferenceConfig<XxxService>();
reference.setInterface(XxxService.class);
reference.setVersion("1.0.0");
......
ReferenceConfigCache cache = ReferenceConfigCache.getCache();
// cache.get will cache the instance of Reference ，and call ReferenceConfig.get method
 to start ReferenceConfig
XxxService xxxService = cache.get(reference);
// Note: Cache will hold ReferenceConfig, do not call destroy method of ReferenceConfi
g outside. If you do this, it will invalidate ReferenceConfig in Cache!
// Use xxxService instance
xxxService.sayHello();
```

Destroy `ReferenceConfig` in the Cache, it also remove `ReferenceConfig` and release the corresponding resources。

```
ReferenceConfigCache cache = ReferenceConfigCache.getCache();
cache.destroy(reference);
```

By default , `ReferenceConfigCache` caches one `ReferenceConfig` for the same service Group, interface, version. The key of `ReferenceConfigCache` is from the group of service Group, interface, and the version.

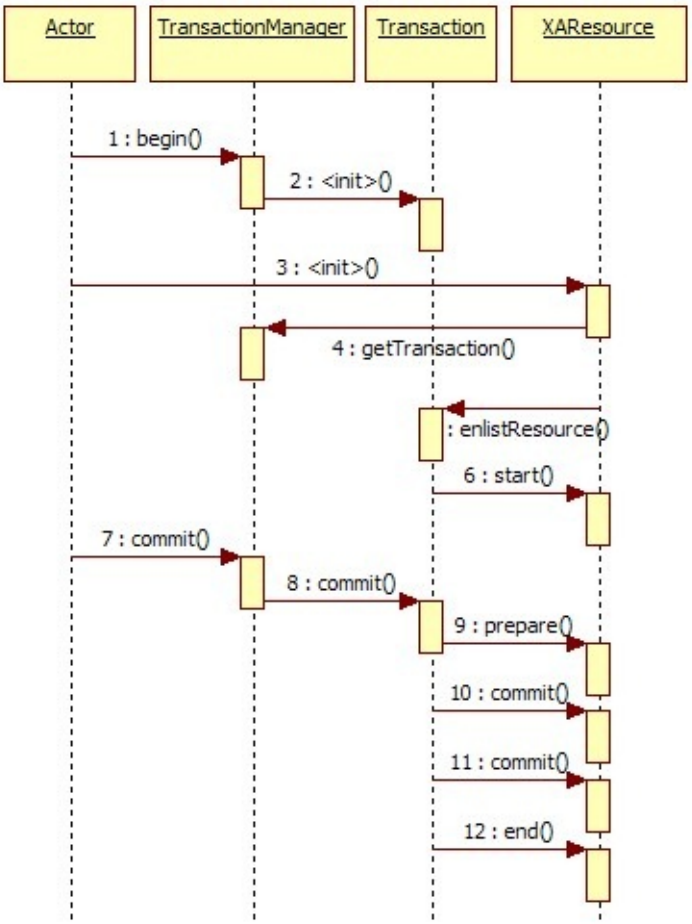
You can modify the strategy. Define an instance of KeyGenerator, pass it as parameter of getCache method. Refer to `ReferenceConfigCache` for information。

```
KeyGenerator keyGenerator = new ...
ReferenceConfigCache cache = ReferenceConfigCache.getCache(keyGenerator );
```

Distributed transaction

Distributed transactions are based on the JTA / XA specification(this feature has not yet been implemented)

Two-phase commit:



Dump

When the business thread pool is full, we need to know what resources/conditions are waiting for the thread , to find the bottleneck point of the system or abnormal point. `dubbo` automatically export thread stack through `Jstack` to keep the scene for easy to troubleshoot the problem.

Default policy:

- Export file path，user.home directory
- Export interval，The shortest interval allows you to export every 10 minutes

Specified export file path:

```
# dubbo.properties
dubbo.application.dump.directory=/tmp
```

```
<dubbo:application ...>
    <dubbo:parameter key="dump.directory" value="/tmp" />
</dubbo:application>
```

Add support for netty4 communication module in 2.5.6 version of dubbo, enabled as follows:

provider：

```
<dubbo:protocol server="netty4" />
```

or

```
<dubbo:provider server="netty4" />
```

consumer：

```
<dubbo:consumer client="netty4" />
```

> **NOTES**
>
> 1. If provider need to use different communication layer framework for different protocols , please configure multiple protocols separately.
> 2. consumer configuration as follow：
>
> ```
> <dubbo:consumer client="netty">
> <dubbo:reference />
> </dubbo:consumer>
> ```
>
> ```
> <dubbo:consumer client="netty4">
> <dubbo:reference />
> </dubbo:consumer>
> ```
>
> Next we will continue to do something：
>
> 1. We will provide a reference data on the performance test indicators and performance test comparison with the version of netty 3.

API Reference

Generally speaking, dubbo keeps its functionality no intrusive as much as possible, but for some particular features, there's no other way not only API can achieve. [1]

These APIs are summarized here below:

Configuration API

```
com.alibaba.dubbo.config.ServiceConfig
com.alibaba.dubbo.config.ReferenceConfig
com.alibaba.dubbo.config.ProtocolConfig
com.alibaba.dubbo.config.RegistryConfig
com.alibaba.dubbo.config.MonitorConfig
com.alibaba.dubbo.config.ApplicationConfig
com.alibaba.dubbo.config.ModuleConfig
com.alibaba.dubbo.config.ProviderConfig
com.alibaba.dubbo.config.ConsumerConfig
com.alibaba.dubbo.config.MethodConfig
com.alibaba.dubbo.config.ArgumentConfig
```

Pls. refer to API Configuration for further information.

Annotation API

```
com.alibaba.dubbo.config.annotation.Service
com.alibaba.dubbo.config.annotation.Reference
```

Pls. refer to Annotation Configuration for further information.

Model API

```
com.alibaba.dubbo.common.URL
com.alibaba.dubbo.rpc.RpcException
```

Context API

```
com.alibaba.dubbo.rpc.RpcContext
```

Pls. refer to context & pass parameter in attachment & asynchronous call for further information.

Service API

```
com.alibaba.dubbo.rpc.service.GenericService
com.alibaba.dubbo.rpc.service.GenericException
```

Pls. refer to generic reference & generic service for further information.

```
com.alibaba.dubbo.rpc.service.EchoService
```

Pls. refer to test via echo service for further details.

> [1]. Attention: do not rely on APIs other than what're mentioned here, otherwise your application may face the risk of incompatibility after upgrade dubbo. ↩

schema configuration reference

The following pages show all the configuration properties [2] with XML Config [1] as an example. For other configurations, please reference: Properties Configuration, Annotation Configuration, API Configuration.

All configuration properties fall into three categories, see the "Function" in the table below.

- Service discovery: used for service registration and discovery in order to find providers for consumers.
- Service governance: used for service management and governance, such as to provide conveninence for dev or test.
- Performance optinize: used for optimizing performance. Diffenent properties may has different performance impact.
- All properties will transform into URL [3] which is generated by provider. The url will be subscribed by consumers through registry. Please see the `Corresponding URL parameter` in the table below for each property.

> [1]. XML Schema: http://code.alibabatech.com/schema/dubbo/dubbo.xsd ↩

> [2]. Notice: These three properties, group, interface, and version determine a service. All other properties are used for service governance or performance optimize. ↩

> [3]. URL format：`protocol://username:password@host:port/path?key=value&key=value` ↩

dubbo:service

The configuration of the service provider. The corresponding class is `com.alibaba.dubbo.config.ServiceConfig`.

Attribute	Corresponding URL parameter	Type	Required	Defau
interface		class	**True**	
ref		object	**True**	
version	version	string	False	0.0.0
group	group	string	False	
path	<path>	string	False	defaul is the interfa name
delay	delay	int	False	0
timeout	timeout	int	False	1000
retries	retries	int	False	2
connections	connections	int	False	100
loadbalance	loadbalance	string	False	rando

async	async	boolean	False	false
stub	stub	class/boolean	False	false
mock	mock	class/boolean	False	false
token	token	string/boolean	False	false
registry		string	False	regist regist defau
provider		string	False	use th config provid
deprecated	deprecated	boolean	False	false
dynamic	dynamic	boolean	False	true

accesslog	accesslog	string/boolean	False	false
owner	owner	string	False	
document	document	string	False	
weight	weight	int	False	
executes	executes	int	False	0
proxy	proxy	string	False	javass
cluster	cluster	string	False	failove
filter	service.filter	string	False	defau
listener	exporter.listener	string	False	defau
protocol		string	False	
layer	layer	string	False	
register	register	boolean	False	true

dubbo:reference

The configuration of service consumer. The corresponding class is `com.alibaba.dubbo.config.ReferenceConfig`

Attribute	Corresponding URL parameter	Type	Required	Def
id		string	**True**	
interface		class	**True**	
version	version	string	False	
group	group	string	False	
timeout	timeout	long	False	By de <dubb timeo
retries	retries	int	False	By de <dubb retries
connections	connections	int	False	By de <dubb conne
loadbalance	loadbalance	string	False	By de <dubb loadb
async	async	boolean	False	By de <dubb async
generic	generic	boolean	False	By de <dubb gener
check	check	boolean	False	By de <dubb check

url	url	string	False	
stub	stub	class/boolean	False	
mock	mock	class/boolean	False	
cache	cache	string/boolean	False	
validation	validation	boolean	False	
proxy	proxy	boolean	False	javass
client	client	string	False	
registry		string	False	By de merge servic that g regist
owner	owner	string	False	
actives	actives	int	False	0
cluster	cluster	string	False	failove
filter	reference.filter	string	False	defau

listener	invoker.listener	string	False	defau
layer	layer	string	False	
init	init	boolean	False	false
protocol	protocol	string	False	

dubbo:protocol

Service provider protocol configuration. The corresponding class is `com.alibaba.dubbo.config.ProtocolConfig`. If you need to support multiple protocols, you could declare multiple `<dubbo:protocol>` tags, and specify the protocol via `protocol` property.

Attribute	Corresponding URL parameter	Type	Required	Defa
id		string	False	dubbo
name	<protocol>	string	**True**	dubbo
port	<port>	int	False	The d of dub protoc 20880 protoc 1099, hessia protoc 80；It allocat unuse `port` filled c `-1`. T the po is cont the po increa on the corres

				protoc port af Dubbo
host	<host>	string	False	Find lo autom
threadpool	threadpool	string	False	fixed
threads	threads	int	False	100
iothreads	threads	int	False	The co CPU +
accepts	accepts	int	False	0
payload	payload	int	False	88388
codec	codec	string	False	dubbo
serialization	serialization	string	False	The de serializ dubbo is hess protoc

				http pr json
accesslog	accesslog	string/boolean	False	
path	<path>	string	False	
transporter	transporter	string	False	The de value protoc
server	server	string	False	The de value protoc netty, protoc servle
client	client	string	False	The de value protoc
				The de

dispatcher	dispatcher	string	False	value protoc
queues	queues	int	False	0
charset	charset	string	False	UTF-8
buffer	buffer	int	False	8192
heartbeat	heartbeat	int	False	0
telnet	telnet	string	False	
register	register	boolean	False	true
				Defau

				an em

dubbo:registry

The configuration of the registry center. The corresponding class is `com.alibaba.dubbo.config.RegistryConfig`. If you have multiple different registries, you can declare multiple `<dubbo:registry>` tags, and then reference specified registry with `registry` property in `<dubbo:service>` or `<dubbo:reference>` tag.

Attribute	Corresponding URL parameter	Type	Required	Default Value
id		string	False	
address	<host:port>	string	**True**	
protocol	<protocol>	string	False	dubbo
port	<port>	int	False	9090

username	<username>	string	False	
password	<password>	string	False	
transport	registry.transporter	string	False	netty
timeout	registry.timeout	int	False	5000
session	registry.session	int	False	60000
file	registry.file	string	False	
check	check	boolean	False	true

register	register	boolean	False	true
subscribe	subscribe	boolean	False	true
dynamic	dynamic	boolean	False	true

dubbo:monitor

Monitor center configuration. The corresponding class:
`com.alibaba.dubbo.config.MonitorConfig`

Property	The corresponding class	Type	Requisite	Default	Effect
protocol	protocol	string	N	dubbo	service governance
address	<url>	string	N	N/A	service governance

dubbo:application

Application configuration. The corresponding class:
`com.alibaba.dubbo.config.ApplicationConfig`

Property	Corresponding URL parameter	Type	Requisite	Default	
name	application	string	**Y**		s g
version	application.version	string	N		s g
owner	owner	string	N		s g
organization	organization	string	N		s g
architecture	architecture	string	N		s g
environment	environment	string	N		s g
compiler	compiler	string	N	javassist	p o
logger	logger	string	N	slf4j	p o

dubbo:module

Module configuration. The corresponding class `com.alibaba.dubbo.config.ModuleConfig`

Property	The corresponding class	Type	Requisite	Default	Effect
name	module	string	**Y**		service governar
version	module.version	string	N		service governar
owner	owner	string	N		service governar
organization	organization	string	N		service governar

dubbo:provider

The default configuration of service provider. The corresponding class is `com.alibaba.dubbo.config.ProviderConfig`. This tag provider default values for `<dubbo:service>` and `<dubbo:protocol>`.

Attribute	Corresponding URL parameter	Type	Required	Defa
id		string	False	dubbo
protocol	<protocol>	string	False	dubbo
host	<host>	string	False	Find l autom
threads	threads	int	False	100
payload	payload	int	False	88388
path	<path>	string	False	
server	server	string	False	Defau for du protoc for htt
client	client	string	False	Defau for du protoc
codec	codec	string	False	dubbo
serialization	serialization	string	False	Defau hessia dubbo json f protoc
default		boolean	False	false

filter	service.filter	string	False	
listener	exporter.listener	string	False	
threadpool	threadpool	string	False	fixed
accepts	accepts	int	False	0
version	version	string	False	0.0.0
group	group	string	False	
delay	delay	int	False	0
timeout	default.timeout	int	False	1000
retries	default.retries	int	False	2
connections	default.connections	int	False	0
loadbalance	default.loadbalance	string	False	rando
async	default.async	boolean	False	false
stub	stub	boolean	False	false
mock	mock	boolean	False	false

token	token	boolean	False	false
registry	registry	string	False	By de regist regist
dynamic	dynamic	boolean	False	true
accesslog	accesslog	string/boolean	False	false
owner	owner	string	False	
document	document	string	False	
weight	weight	int	False	
executes	executes	int	False	0
actives	default.actives	int	False	0
proxy	proxy	string	False	javass
cluster	default.cluster	string	False	failove
deprecated	deprecated	boolean	False	false
queues	queues	int	False	0
charset	charset	string	False	UTF-8

buffer	buffer	int	False	8192
iothreads	iothreads	int	False	CPU +
telnet	telnet	string	False	
contextpath	contextpath	String	False	Empty
layer	layer	string	False	

dubbo:consumer

Consumer default configuration. The corresponding clas：
`com.alibaba.dubbo.config.ConsumerConfig`. It is also default configuration of
`<dubbo:reference>`.

Property	Corresponding URL parameter	Type	Requisite	Defa
timeout	default.timeout	int	N	1000
retries	default.retries	int	N	2
loadbalance	default.loadbalance	string	N	rando
async	default.async	boolean	N	false
connections	default.connections	int	N	100
generic	generic	boolean	N	false
check	check	boolean	N	true
proxy	proxy	string	N	javas
owner	owner	string	N	
actives	default.actives	int	N	0
cluster	default.cluster	string	N	failov

filter	reference.filter	string	N	
listener	invoker.listener	string	N	
registry		string	N	regist with t regist
layer	layer	string	N	
init	init	boolean	N	false
cache	cache	string/boolean	N	
validation	validation	boolean	N	

dubbo:method

Method level configuration. The corresponding class: `com.alibaba.dubbo.config.MethodConfig`. This tag is a child tag of `<dubbo:service>` or `<dubbo:reference>`, for accuracy to method level.

Property	Corresponding URL parameter	Type	Requisite	
name		string	**Y**	
timeout	<metodName>.timeout	int	N	<dubb timeo
retries	<metodName>.retries	int	N	<dubb retrie
loadbalance	<metodName>.loadbalance	string	N	<dubb loadb
async	<metodName>.async	boolean	N	<dubb async
sent	<methodName>.sent	boolean	N	true
actives	<metodName>.actives	int	N	0
executes	<metodName>.executes	int	N	0

deprecated	<methodName>.deprecated	boolean	N	false
sticky	<methodName>.sticky	boolean	N	false
return	<methodName>.return	boolean	N	true
oninvoke		String	N	
onreturn		String	N	
onthrow		String	N	
cache	<methodName>.cache	string/boolean	N	
validation	<methodName>.validation	boolean	N	

For example:

```
<dubbo:reference interface="com.xxx.XxxService">
    <dubbo:method name="findXxx" timeout="3000" retries="2" />
</dubbo:reference>
```

dubbo:argument

Method argument configuration. The corresponding class： `com.alibaba.dubbo.config.ArgumentConfig` . This tag is child of `<dubbo:method>` , which is for feature description of method argument, such as:

```
<dubbo:method name="findXxx" timeout="3000" retries="2">
    <dubbo:argument index="0" callback="true" />
</dubbo:method>
```

Property	Corresponding URL parameter	Type	Requisite	Default	
index		int	**Y**		ide
type		String	Index and type choose one		ide
callback	<metodName><index>.retries	boolean	N		ser gov

dubbo:parameter

Optional parameter configuration. The corresponding class is `java.util.Map`. This tag is used as a sub tag to configure custom parameters for extending `<dubbo:protocol>`, `<dubbo:service>`, `<dubbo:provider>`, `<dubbo:reference>` or `<dubbo:consumer>`.

Attribute	Corresponding URL parameter	Type	Required	Default Value
key	key	string	**True**	
value	value	string	**True**	

For example：

```
<dubbo:protocol name="napoli">
    <dubbo:parameter key="http://10.20.160.198/wiki/display/dubbo/napoli.queue.name" v
alue="xxx" />
</dubbo:protocol>
```

you can also use it like this:

```
<dubbo:protocol name="jms" p:queue="xxx" />
```

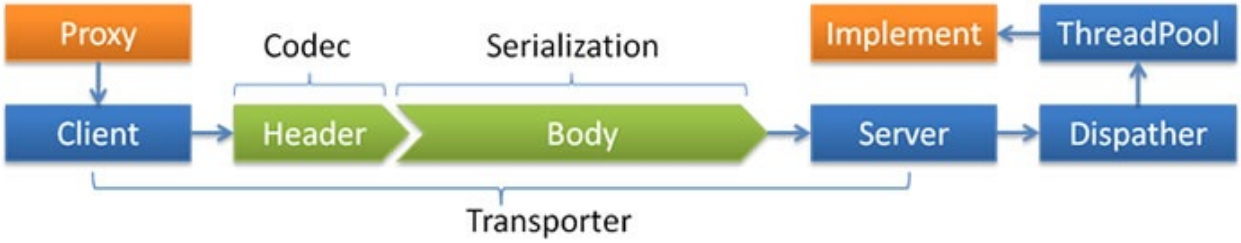
Protocol reference Book

Dubbo protocol is recommended. The performance of each protocol, see:Performance

dubbo://

Dubbo protocol which is the default protocol of Dubbo RPC Framework uses a single long connection and NIO asynchronous communication,it is suitable for small data but with high concurrency RPC call and the number of consumer machine is much greater than provider

On the other hand, the Dubbo protocol is not suitable for transmitting large amounts of data, such as file transmission, video transmission, etc., unless the request is very low.



- Transporter: mina, netty, grizzy
- Serialization: dubbo, hessian2, java, json
- Dispatcher: all, direct, message, execution, connection
- ThreadPool: fixed, cached

Features

The default protocol is Dubbo protocol ,based on netty `3.2.5.Final` and Hessian2 `3.2.1-fixed-2(Alibaba embed version)` .

- Default connection number: single connection
- Default connection mode: long connection
- Transmission protocol: TCP
- Transmission mode: NIO asynchronous transmission
- Serialization: Hessian2 serialization
- Scope of application: incoming and outgoing data packets are small (recommended less than 100K),try not to transfer large files or large strings with Dubbo protocol.
- Applicable scenarios:: most RPC scenarios

Constraint

- Parameters and return values must implement `Serializable` interface
- Parameters and return values can not be customized to implement `List` , `Map` ,

`Number`, `Date`, `Calendar` interface, can only be implemented with the JDK, because Hessian2 will do some special treatment, Attribute values in the class will be lost.

- Hessian serialization:to solve compatibility issues, only serialize class name,all the fields declared by the class,not included static fields,method information

Data transformation	Cases	Result
A->B	Class A has one more property than Class B	It doesn't throw exception ，Class B doesn't have Class A new property,other is normal
A->B	enum Class A has one more new enum than enum Class B，when use Class A new enum to transfor to B	throw exception
A->B	enum Class A has one more new enum than enum Class B，when don't use Class A new enum to transfor to B	It doesn't throw exception
A->B	Class A and Class B have same property name,but the property type is different	throw exception
A->B	serialId is not same	normal

The interface new addition method has no effect on the client. If the method is not required by the client, the client does not need to redeploy it. The input parameter and result class add new properties, and if the client does not need new properties, it does not need to be redeployed too.

The change of input parameter and result property name has no effect on client serialization, but if the client is not redeployed, no matter the input or output, the value of which property name had change is not available.

Summary: the server side and the client do not need to be fully consistent with the domain objects,but you still should know about what would happen.

Configuration

configure protocol

```
<dubbo:protocol name="dubbo" port="20880" />
```

configure provider level default protocol:

```
<dubbo:provider protocol="dubbo" />
```

configure service level default protocol:

```
<dubbo:service protocol="dubbo" />
```

configure multiple port：

```
<dubbo:protocol id="dubbo1" name="dubbo" port="20880" />
<dubbo:protocol id="dubbo2" name="dubbo" port="20881" />
```

configure protocol options:

```
<dubbo:protocol name=“dubbo” port=“9090” server=“netty” client=“netty” codec=“dubbo” s
erialization=“hessian2” charset=“UTF-8” threadpool=“fixed” threads=“100” queues=“0” io
threads=“9” buffer=“8192” accepts=“1000” payload=“8388608” />
```

configure multiple connectios:

Dubbo protocol default uses a single long connection per service per consumer for each service provider,and multiple connections can be used if the amount of data is large

```
<dubbo:protocol name="dubbo" connections="2" />
```

- `<dubbo:service connections="0">` OR `<dubbo:reference connections="0">` It means that the service uses a share long connection per provider. `default`
- `<dubbo:service connections="1">` OR `<dubbo:reference connections="1">` It means that the service uses a separate long connection.
- `<dubbo:service connections="2">` OR `<dubbo:reference connections="2">` It means that the service uses two separate long connection.

To prevent being hung up by a large number of connections, you can limit the number of connections at the service provider side.

```
<dubbo:protocol name="dubbo" accepts="1000" />
```

or configure in `dubbo.properties` ：

```
dubbo.service.protocol=dubbo
```

rmi://

The RMI protocol uses the JDK standard `java.rmi.*` Implementation, using a block short connection and JDK standard serialization.

Features

- Number of connections: multiple connections
- Connection: short connection
- Transmission protocol: HTTP
- Transmission: synchronous transmission
- Serialization: Java standard Object Serialization
- Scope of application:the number of providers is more than that of consumers and can transfer files.
- Applicable scenarios: Conventional remote service method calls, interoperating with native RMI services

Constraint

- Parameters and return values must implement `Serializable` interface
- The timeout configuration for RMI is invalid, you need to use java startup parameter settings: `-Dsun.rmi.transport.tcp.responseTimeout=3000` ,see the RMI configuration below

Configuration in dubbo.properties

```
dubbo.service.protocol=rmi
```

RMI Configuration

```
java -Dsun.rmi.transport.tcp.responseTimeout=3000
```

more RMI options please check JDK Document

Interface

If the service interface implement the `java.rmi.Remote` interface, it can interoperate with the native RMI, ie:

- Providers expose services using Dubbo's RMI protocol, consumers call directly with the standard RMI interface,
- Or the provider exposes services using standard RMI, and consumers invoke with Dubbo's RMI protocol.

If the service interface doesn't implement the `java.rmi.Remote` interface:

- Default Dubbo will automatically generate a `com.xxx.XxxService$Remote` interface and implement the `java.rmi.Remote` interface, expose the service as this interface,
- But if `<dubbo: protocol name = 'rmi' codec = 'spring' />` is set, the `$Remote` interface will not be generated, but Spring's `RmiInvocationHandler` interface will be used to expose services.

Configuration

configure RMI protocol：

```
<dubbo:protocol name="rmi" port="1099" />
```

configure provider level default protocol:

```
<dubbo:provider protocol="rmi" />
```

configure service level default protocol:

```
<dubbo:service protocol="rmi" />
```

configure multiple port：

```
<dubbo:protocol id="rmi1" name="rmi" port="1099" />
<dubbo:protocol id="rmi2" name="rmi" port="2099" />

<dubbo:service protocol="rmi1" />
```

Compatible with Spring：

```
<dubbo:protocol name="rmi" codec="spring" />
```

hessian://

Hessian protocol is used for integrate Hessian services, and it use http protocol to communicate and expose services by servlet.Dubbo use Jetty server as default servlet container.

Dubbo's Hessian protocol interoperates with native Hessian services:

- Providers use Dubbo's Hessian protocol to expose services that consumers call directly using standard Hessian interfaces
- Alternatively, the provider exposes the service using standard Hessian and the consumer calls it using Dubbo's Hessian protocol.

Features

- Number of connections: multiple connections
- Connection: short connection
- Transmission protocol: HTTP
- Transmission: synchronous transmission
- Serialization: Hessian binary serialization
- Scope of application: Incoming and outgoing parameter packets are large, the number of providers is more than that of consumers and can transfer files.
- Applicable scenarios: page transfer, file transfer, or interoperability with native hessian services

dependency

```
<dependency>
    <groupId>com.caucho</groupId>
    <artifactId>hessian</artifactId>
    <version>4.0.7</version>
</dependency>
```

Constraint

- Parameters and return class must implement `Serializable` interface
- Parameters and return values can not be customized to implement `List` , `Map` ,

`Number` , `Date` , `Calendar` interface, can only be implemented with the JDK, because Hessian2 will do some special treatment, Attribute values in the class will be lost.

Configuration

configure hessian protocol：

```
<dubbo:protocol name="hessian" port="8080" server="jetty" />
```

configure provider level default protocol:

```
<dubbo:provider protocol="hessian" />
```

configure service level default protocol:

```
<dubbo:service protocol="hessian" />
```

configure multiple port：

```
<dubbo:protocol id="hessian1" name="hessian" port="8080" />
<dubbo:protocol id="hessian2" name="hessian" port="8081" />
```

configure direct connect mode：

```
<dubbo:reference id="helloService" interface="HelloWorld" url="hessian://10.20.153.10:
8080/helloWorld" />
```

http://

Dubbo http protocol is base on HTTP form and Spring's HttpInvoker

Features

- Number of connections: multiple connections
- Connection: short connection
- Transmission protocol: HTTP
- Transmission: synchronous transmission
- Serialization: form serialization
- Scope of application: Available browser view, the form or URL can be passed parameters, Temporary files are not supported.
- Applicable scenarios: Services that need to be available to both application and browser

Constraint

- Parameters and return values must be consistent with Bean specifications

Configuration

configure http protocol：

```
<dubbo:protocol name="http" port="8080" />
```

configure Jetty Server (default)：

```
<dubbo:protocol ... server="jetty" />
```

configure Servlet Bridge Server (recommend)：

```
<dubbo:protocol ... server="servlet" />
```

configure DispatcherServlet：

```
<servlet>
        <servlet-name>dubbo</servlet-name>
        <servlet-class>com.alibaba.dubbo.remoting.http.servlet.DispatcherServlet</ser
vlet-class>
        <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
        <servlet-name>dubbo</servlet-name>
        <url-pattern>/*</url-pattern>
</servlet-mapping>
```

Note that if you use servlets to dispatch requests:

- the port of protocol `<dubbo:protocol port="8080" />` must same as servlet container's.
- the context path of protocol `<dubbo:protocol contextpath="foo" />` must same as servlet application's.

webservice://

WebService-based remote calling protocol，base on Apache CXF `frontend-simple` and `transports-http` implements。

Interoperable with native WebService services：

- Providers expose services using Dubbo's WebService protocol, which consumers invoke directly using the standard WebService interface,
- Or the provider exposes the service using the standard WebService, which consumers invoke using the Dubbo WebService protocol.

dependency

```
<dependency>
    <groupId>org.apache.cxf</groupId>
    <artifactId>cxf-rt-frontend-simple</artifactId>
    <version>2.6.1</version>
</dependency>
<dependency>
    <groupId>org.apache.cxf</groupId>
    <artifactId>cxf-rt-transports-http</artifactId>
    <version>2.6.1</version>
</dependency>
```

Features

- Number of connections: multiple connections
- Connection: short connection
- Transmission protocol: HTTP
- Transmission: synchronous transmission
- Serialization: SOAP text serialization
- Applicable scenarios: System integration, cross-language calls

Constraint

- Parameters and return class should implement `Serializable` interface
- Parameters should try to use the basic types and POJO

Configuration

configure webservice protocol：

```
<dubbo:protocol name="webservice" port="8080" server="jetty" />
```

configure provider level default protocol:

```
<dubbo:provider protocol="webservice" />
```

configure service level default protocol:

```
<dubbo:service protocol="webservice" />
```

configure multiple port：

```
<dubbo:protocol id="webservice1" name="webservice" port="8080" />
<dubbo:protocol id="webservice2" name="webservice" port="8081" />
```

configure direct connect mode：

```
<dubbo:reference id="helloService" interface="HelloWorld" url="webservice://10.20.153.
10:8080/com.foo.HelloWorld" />
```

WSDL：

```
http://10.20.153.10:8080/com.foo.HelloWorld?wsdl
```

Jetty Server (Default)：

```
<dubbo:protocol ... server="jetty" />
```

Servlet Bridge Server (recommend)：

```
<dubbo:protocol ... server="servlet" />
```

configure DispatcherServlet：

```
<servlet>
        <servlet-name>dubbo</servlet-name>
        <servlet-class>com.alibaba.dubbo.remoting.http.servlet.DispatcherServlet</ser
vlet-class>
        <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
        <servlet-name>dubbo</servlet-name>
        <url-pattern>/*</url-pattern>
</servlet-mapping>
```

Note that if you use servlets to dispatch requests:

- the port of protocol `<dubbo:protocol port="8080" />` must same as servlet container's.
- the context path of protocol `<dubbo:protocol contextpath="foo" />` must same as servlet application's.

thrift://

The current dubbo support thrift protocol is an extension of the thrift native protocol, adding some additional header information to the native protocol, such as service name, magic number, and so on.

The use of dubbo thrift protocol also need to use thrift idl compiler to generate the corresponding java code, follow-up version will do some enhancement in this aspect.

dependency

```
<dependency>
    <groupId>org.apache.thrift</groupId>
    <artifactId>libthrift</artifactId>
    <version>0.8.0</version>
</dependency>
```

Configuration

```
<dubbo:protocol name="thrift" port="3030" />
```

Example

you can check dubbo thrift example

Common problem

- Thrift does not support null values, that is, you can not pass null values

memcached://

RPC protocol based on memcached implementation.

Register memcached service address

```
RegistryFactory registryFactory = ExtensionLoader.getExtensionLoader(RegistryFactory.c
lass).getAdaptiveExtension();
Registry registry = registryFactory.getRegistry(URL.valueOf("zookeeper://10.20.153.10:
2181"));
registry.register(URL.valueOf("memcached://10.20.153.11/com.foo.BarService?category=pr
oviders&dynamic=false&application=foo&group=member&loadbalance=consistenthash"));
```

Use in client

get service reference:

```
<dubbo:reference id="cache" interface="java.util.Map" group="member" />
```

or direct access by IP:

```
<dubbo:reference id="cache" interface="java.util.Map" url="memcached://10.20.153.10:11
211" />
```

you can also use a custom interface：

```
<dubbo:reference id="cache" interface="com.foo.CacheService" url="memcached://10.20.15
3.10:11211" />
```

The method name is the same as the standard method name of memcached, just like get(key), set(key, value), delete(key)。

If the method name and the memcached standard method name are not the same, you need to configure the mapping

```
<dubbo:reference id="cache" interface="com.foo.CacheService" url="memcached://10.20.15
3.10:11211" p:set="putFoo" p:get="getFoo" p:delete="removeFoo" />
```

redis://

RPC protocol based on memcached implementation.

Register redis service address

```
RegistryFactory registryFactory = ExtensionLoader.getExtensionLoader(RegistryFactory.c
lass).getAdaptiveExtension();
Registry registry = registryFactory.getRegistry(URL.valueOf("zookeeper://10.20.153.10:
2181"));
registry.register(URL.valueOf("redis://10.20.153.11/com.foo.BarService?category=provid
ers&dynamic=false&application=foo&group=member&loadbalance=consistenthash"));
```

Use in client

get service reference:

```
<dubbo:reference id="store" interface="java.util.Map" group="member" />
```

or direct access by IP:

```
<dubbo:reference id="store" interface="java.util.Map" url="redis://10.20.153.10:6379"
/>
```

you can also use a custom interface：

```
<dubbo:reference id="store" interface="com.foo.StoreService" url="redis://10.20.153.10
:6379" />
```

The method name is the same as the standard method name of memcached, just like get(key), set(key, value), delete(key)。

If the method name and the memcached standard method name are not the same, you need to configure the mapping

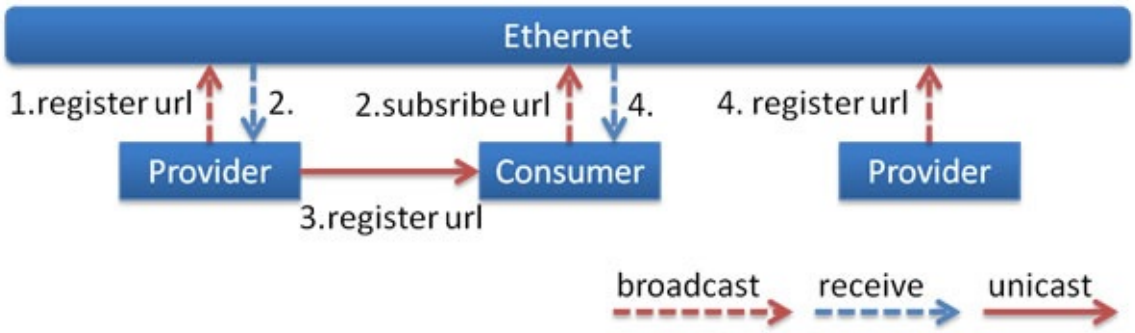
```
<dubbo:reference id="cache" interface="com.foo.CacheService" url="memcached://10.20.15
3.10:11211" p:set="putFoo" p:get="getFoo" p:delete="removeFoo" />
```

Registry Server References

It is recommended to use zookeeper registry server

Multicast Registry

Multicast registry doesn't require to setup any central node. Just like IP address broadcast, dubbo service providers and consumers can discover each other through this mechanism.



1. Service provider broadcasts its address when it boots up.
2. Service consumer broadcasts its subscription request when it boots up.
3. Once service provider receives subscription request, it unicasts its own address to the corresponding consumer, if `unicast=false` is set, then broadcast will be used instead.
4. When service consumer receives provider's address, it can start RPC invocation on the received address.

Multicast is limited to network topology, and is only suitable for development purpose or small deployment. The valid multcast addresses scope is: 224.0.0.0 - 239.255.255.255.

Configuration

```
<dubbo:registry address="multicast://224.5.6.7:1234" />
```

Or

```
<dubbo:registry protocol="multicast" address="224.5.6.7:1234" />
```

In order to avoid multicast as much as possible, dubbo uses unicast for address information from service provider to service consumer, if there are multiple consumer processes on one single machine, consumers need to set `unicast=false`, otherwise only one consumer can be able to receive the address info:

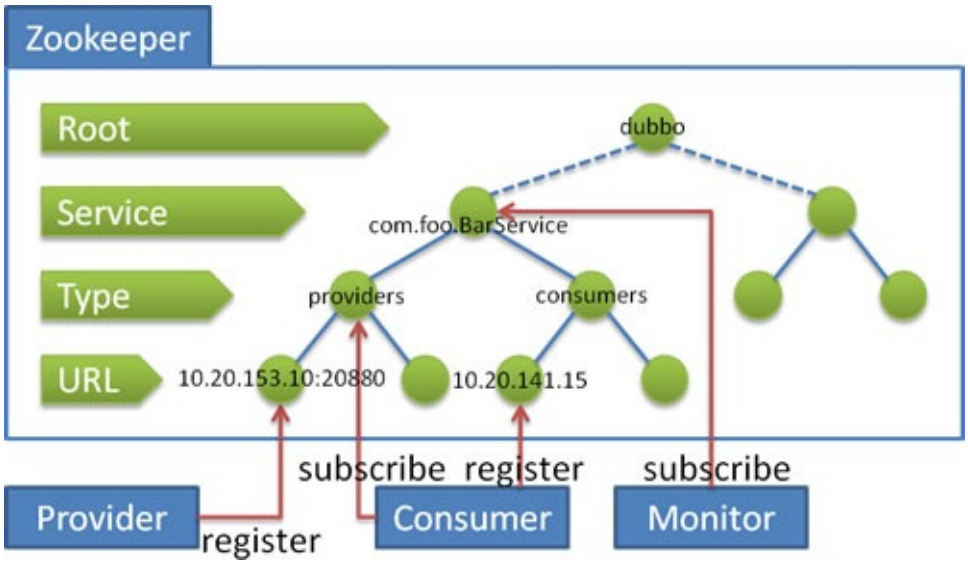
```
<dubbo:registry address="multicast://224.5.6.7:1234?unicast=false" />
```

Or

```
<dubbo:registry protocol="multicast" address="224.5.6.7:1234">
    <dubbo:parameter key="unicast" value="false" />
</dubbo:registry>
```

Zookeeper Registry Server

Zookeeper is the child project of apache hadoop. Since it offers tree-like directory service and supports change notification, it's suitable to use it as dubbo's registry server. It's a field-proven product, therefore it's recommended to use it in the production environment. [1]



Description on registration procedure:

- When service provider boots up: write service URL address under directory `/dubbo/com.foo.BarService/providers`
- When service consumer boots up: subscribe to `/dubbo/com.foo.BarService/providers` for provider's URL addresses. At the same time, write consumer's URL address under `/dubbo/com.foo.BarService/providers` .
- When monitor center boots up: subscribe to `/dubbo/com.foo.BarService` for the URL addresses from all providers and consumers.

The following abilities are supported:

- When provider stops by accident, registry server can remove its info automatically.
- When registry server reboots, all registration data and subscription requests can be recovered automatically.
- When session is expired, all registration data and subscription requests can be recovered automatically.
- When `<dubbo:registry check="false" />` is configured, failed requests for subscription and registration will be recorded and kept retrying in the background.
- Configure `<dubbo:registry username="admin" password="1234" />` for zookeeper login.
- Configure `<dubbo:registry group="dubbo" />` for dubbo's root node on zookeeper. Default root node will be used if it's not specified.

- Support to use wildcard `*` in `<dubbo:reference group="*" version="*" />` in order to subscribe all groups and all versions for the services to be referenced.

How to Use

Add zookeeper client dependency in both provider and consumer:

```
<dependency>
    <groupId>org.apache.zookeeper</groupId>
    <artifactId>zookeeper</artifactId>
    <version>3.3.3</version>
</dependency>
```

Or download directly from apache.

Dubbo supports two zookeeper clients: zkclient and curator:

Use zkclient

Since `2.2.0` dubbo uses zkclient by default, in order to improve the robustness. zkclient is a zookeeper client implementation open-sourced by Datameer.

Default configuration:

```
<dubbo:registry ... client="zkclient" />
```

Or:

```
dubbo.registry.client=zkclient
```

Or:

```
zookeeper://10.20.153.10:2181?client=zkclient
```

In order to use it, need to explicitly declare the following maven dependency or download its client.

```
<dependency>
    <groupId>com.github.sgroschupf</groupId>
    <artifactId>zkclient</artifactId>
    <version>0.1</version>
</dependency>
```

Use curator

Since `2.3.0` dubbo also supports curator but explicit configuration is required. Curator is the zookeeper client open-sourced by Netflix.

In order to switch to curator, use the configuration below:

```
<dubbo:registry ... client="curator" />
```

Or:

```
dubbo.registry.client=curator
```

Or:

```
zookeeper://10.20.153.10:2181?client=curator
```

Also need to explicitly add maven dependency or directly download the jar:

```
<dependency>
    <groupId>com.netflix.curator</groupId>
    <artifactId>curator-framework</artifactId>
    <version>1.1.10</version>
</dependency>
```

Zookeeper single node configuration:

```
<dubbo:registry address="zookeeper://10.20.153.10:2181" />
```

Or:

```
<dubbo:registry protocol="zookeeper" address="10.20.153.10:2181" />
```

Zookeeper cluster configuration：

```
<dubbo:registry address="zookeeper://10.20.153.10:2181?backup=10.20.153.11:2181,10.20.
153.12:2181" />
```

Or:

```
<dubbo:registry protocol="zookeeper" address="10.20.153.10:2181,10.20.153.11:2181,10.2
0.153.12:2181" />
```

Configure single zookeeper to serve as multiple registry servers:

```
<dubbo:registry id="chinaRegistry" protocol="zookeeper" address="10.20.153.10:2181" gr
oup="china" />
<dubbo:registry id="intlRegistry" protocol="zookeeper" address="10.20.153.10:2181" gro
up="intl" />
```

Zookeeper Installation

Pls. refer to zookeeper install manual for how to install zookeeper based registry server. To set it up, specify `dubbo.registry.address` to `zookeeper://127.0.0.1:2181` in `conf/dubbo.properties` for both provider and consumer (you can refer to quick start) after install a zookeeper server.

Declaration of Reliability

A home-brewed service registry server is used in Alibaba instead of zookeeper server. Zookeeper based registry center does not have long-run practice within Alibaba, therefore we cannot guarantee its reliability. Zookeeper registry server is provided for dubbo community, and its reliability relies on zookeeper itself largely.

Declaration of Compatibility

The original designed data structure for zookeeper in `2.0.8` has the limitation that data type cannot extended, it's redesigned in `2.0.9`. But at the same time incompatibility is introduced, thereby `2.0.9` is required for all service providers and service consumers.

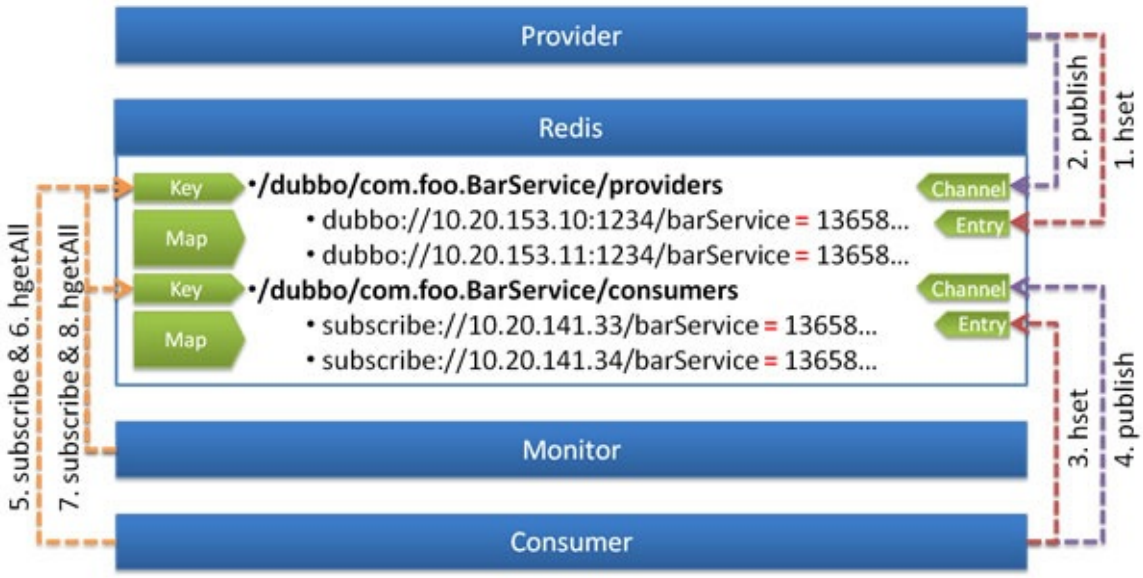
Since `2.2.0` zkclient is used by default, therefore its dependency is needed.

Since `2.3.0` curator is supported as alternative option.

> [1]. Suggest to use `2.3.3` or above for zookeeper registry client ↩

Redis Registry Server

It is a registry server implementation [2] based on redis [1].



Use key/map structure in redis to save the registration info:

- Main key for service name and type
- Key in the map is URL address
- Value in the map is the expiration time. Monitor center uses it to track and remove dirty data [3]

Publish/Subscribe events in redis is leveraged for data change notification:

- Distinguish event type with event's value: `register`, `unregister`, `subscribe`, `unsubscribe`.
- Regular subscriber subscribes the particular key presenting service provider, then will receive `register` and `unregister` events fired from the specified service.
- Monitor center subscribes `/dubbo/*` via `psubscribe`, then will receive all change notifications from all services.

Procedure:

1. When service provider boots up, it adds its address under `Key:/dubbo/com.foo.BarService/providers`.
2. Then service provider sends `register` event to `Channel:/dubbo/com.foo.BarService/providers`
3. When service consumer boots up, it subscribe events `register` and `unregister` from `Channel:/dubbo/com.foo.BarService/providers`

4. Then service consumer add its address under `Key:/dubbo/com.foo.BarService/providers`
5. When service consumer receives events `register` and `unregister`, it will fetch provider's addresses from `Key:/dubbo/com.foo.BarService/providers`
6. When monitor center boots up, it subscribes events `register`, `unregister`, `subscribe`, and `unsubsribe`.
7. After monitor center receives `register` and `unregister`, it fetches provider's addresses from `Key:/dubbo/com.foo.BarService/providers`
8. After monitor center receives `subscribe` and `unsubscribe`, it fetches consumer's addresses from `Key:/dubbo/com.foo.BarService/consumers`

Configuration

```
<dubbo:registry address="redis://10.20.153.10:6379" />
```

Or

```
<dubbo:registry address="redis://10.20.153.10:6379?backup=10.20.153.11:6379,10.20.153.
12:6379" />
```

Or

```
<dubbo:registry protocol="redis" address="10.20.153.10:6379" />
```

Or

```
<dubbo:registry protocol="redis" address="10.20.153.10:6379,10.20.153.11:6379,10.20.15
3.12:6379" />
```

Options

- Config key's prefix in redis via `<dubbo:registry group="dubbo" />`, the default value is `dubbo`.
- Config redis cluster strategy via `<dubbo:registry cluster="replicate" />`, the default value is `failover`:
  - `failover`: when read/write error happens, try another instance, require the cluster to support data replication.
  - `replicate`: client writes to all nodes of the cluster, but only peeks a random node for read. The cluster doesn't need to take care of data replication, but may require

more nodes and higher performance for each node, compared to option 1.

Declaration of Reliability

A home-brewed service registry server is used in Alibaba instead of redis server. Redis based registry center does not have long-run practice within Alibaba, therefore we cannot guarantee its reliability. This registry server implementation is provided for dubbo community, and its reliability relies on redis itself largely.

Installation

Pls. refer to redis install manual for how to install a redis based registry server. To set it up, specify `dubbo.registry.addrss` to `redis://127.0.0.1:6379` in `conf/dubbo.properties` for both provider and consumer (you can refer to quick start) after install a redis server.

> [1]. Redis is a high performance KV cache server ↩

> [2]. Support since `2.1.0` ↩

> [3]. Heartbeat mechanism is used to detect the dirty data in redis. It requires time among servers must be sync in advanced, otherwise expiration check may inaccurate, plus, heartbeats may add extra pressure on servers. ↩

Simple Registry Server

Simple registry server itself is a regular dubbo service. In this way, third-party dependency is unnecessary, and communication keeps consistent at the same moment.

Configuration

Register simple registry server as dubbo service:

```
<?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.springf
ramework.org/schema/beans/spring-beans-2.5.xsdhttp://code.alibabatech.com/schema/dubbo
 http://code.alibabatech.com/schema/dubbo/dubbo.xsd">
    <!-- application info configuration -->
    <dubbo:application name="simple-registry" />
    <!-- service protocol configuration -->
    <dubbo:protocol port="9090" />
    <!-- service configuration -->
    <dubbo:service interface="com.alibaba.dubbo.registry.RegistryService" ref="registr
yService" registry="N/A" ondisconnect="disconnect" callbacks="1000">
        <dubbo:method name="subscribe"><dubbo:argument index="1" callback="true" /></d
ubbo:method>
        <dubbo:method name="unsubscribe"><dubbo:argument index="1" callback="false" />
</dubbo:method>
    </dubbo:service>
    <!-- simple registry server implementation, register other implementation if clust
er ability is a requirement-->
    <bean id="registryService" class="com.alibaba.dubbo.registry.simple.SimpleRegistry
Service" />
</beans>
```

Reference simple registry server service:

```
<dubbo:registry address="127.0.0.1:9090" />
```

Or:

```
<dubbo:service interface="com.alibaba.dubbo.registry.RegistryService" group="simple" v
ersion="1.0.0" ... >
```

Or:

```
<dubbo:registry address="127.0.0.1:9090" group="simple" version="1.0.0" />
```

Applicability

This `SimpleRegistryService` is just a simple implementation for register server, and it doesn't have cluster support. It is useful for the implementation reference for the custom registery server, but not suitable for use in production environment directly.

Telnet Command Reference

Since `2.0.5` dubbo starts supporting to use telnet command to govern services.

How To Use

```
telnet localhost 20880
```

Or:

```
echo status | nc -i 1 localhost 20880
```

It is possible to extend command `status` to check more resources, pls. refer to extension references for more details.

Supported Commands

The built-in telnet commands are listed below. Furthermore, it is possible to extend telnet commands, pls. refer to extend telnet command for more details.

`ls`

1. `ls` : list services
2. `ls -l` : list services in more details
3. `ls XxxService` : list methods for the particular service
4. `ls -l XxxService` : list methods for the particular service in more dtails

`ps`

1. `ps` : list service ports
2. `ps -l` : list service addresses
3. `ps 20880` : show connection info for the particular port
4. `ps -l 20880` : show connection info for the particular port in more details

`cd`

1. `cd XxxService` : switch default service. When default service is set, service parameter can be ignored in all commands when it's needed
2. `cd /` : reset default service

pwd

`pwd` : show what current default service is

trace

1. `trace XxxService` : trace method invocation once for the given service
2. `trace XxxService 10` : trace method invocations 10 times for the given service
3. `trace XxxService xxxMethod` : trace particular method invocation once for the given service
4. `trace XxxService xxxMethod 10` : trace particular method invocations 10 times for the given service

count

1. `count XxxService` : count method invocation once for the given service
2. `count XxxService 10` : count method invocations 10 times for the given service
3. `count XxxService xxxMethod` : count particular method invocation once for the given service
4. `count XxxService xxxMethod 10` : count particular method invocation 10 times for the given service

invoke

1. `invoke XxxService.xxxMethod({"prop": "value"})` : invoke particular method for the given service
2. `invoke xxxMethod({"prop": "value"})` : invoke particular method for the default service

status

1. `status` : show summarized status. This status summarizes statuses from all resources, and it shows OK when all resources are OK, shows ERROR when any resource has ERROR, and WARN when any has WARN.
2. `status -l` : show status list

log [1]

1. `log debug` : modify logger level to debug
2. `log 100` : examine the last 100 characters from the file logger

help

1. `help` : show help for telnet commands
2. `help xxx` : 显示xxx命令的详细帮助信息
3. `help xxx` : show help for particular telnet command

clear

1. `clear` : clear screen
2. `clear 100` : only clear particular lines on the screen

exit

`exit` : exit current telnet session

> [1]. support since `2.0.6` ↩

Maven Plugin Reference

Start a simple registry server

Start a simple registry server listening on port 9099 [1]:

```
mvn dubbo:registry -Dport=9099
```

Generate a service provider demo application

Generate a service provider with the specified interface and version:

```
mvn dubbo:create -Dapplication=xxx -Dpackage=com.alibaba.xxx -Dservice=XxxService,YyyS
ervice -Dversion=1.0.0
```

> [1]. Default port is 9090 if the port is not specified ↩

Servitization best practice

Modularization

It is recommended to put service interfaces, service models, service exceptions, and so on in the API package,Because the service model and exception are part of the API, it is also in conformity with the modularization principle:Reusing the publish equivalence principle (REP) and the Common Reuse Principle (CRP).

If you need, you can also consider placing a spring reference configuration in the API package, so that the user can only use the configuration in the spring loading process, and the configuration suggestion is placed in the package directory of the module, so as not to conflict, eg: `com/alibaba/china/xxx/dubbo-reference.xml` 。

Granularity

The service interface should have large granularity as possible.Each service method should represent a function rather than a step of a function, otherwise it will be faced with distributed transaction problem. Dubbo does not provide distributed transaction support at present.

The service interface recommends the division of the business scene as a unit and abstract the similar business to prevent the explosion of the number of interfaces.

It is not recommended to use an too abstract universal interface, such as Map query (Map), which has no explicit semantics, which will inconvenience later maintenance.

Version

Each interface should define a version number to provide possible subsequent incompatible upgrades,eg: `<dubbo:service interface="com.xxx.XxxService" version="1.0" />` 。

It is recommended to use a two bit version number, because the third - bit version number is usually compatible with a compatible upgrade, and a change of service version is required only when incompatible.

When incompatible, half of the provider is upgraded to a new version, and all the consumers are upgraded to a new version, and the remaining half providers are upgraded to a new version.

Compatibility

The service interface adds method or the service model adds fields. It can be backward compatible, delete methods or delete fields, and will not be compatible. The new fields of the enumerated type are not compatible, so we need to upgrade by changing the version number.

The compatibility of each protocol is different, see: Protocol introduction

Enumeration type

If it is a complete set, you can use Enum, eg: `ENABLE` , `DISABLE` °

If it is the type of business, there will be an obvious type of increase in the future, and it is not recommended to use `Enum` , and it is not recommended to use Enum and can be replaced by `String` .

If you use `Enum` in the return value,And add the `Enum` value,suggestions to upgrade the service consumption, so that the service provider does not return a new value.

If the `Enum` value is used in the incoming parameter,and add the `Enum` value,it is suggested that the service provider be upgraded first, so that the service consumer will not pass the new value.

Serialization

The service parameters and return values suggest that the POJO object is used, that is, the object of the attribute is represented by the `setter` , `getter` method.

Service parameters and return values do not recommend the use of interfaces, because data model abstraction is of little significance, and serialization requires interfaces to implement meta information of classes, and can not play the purpose of hiding implementation.

Service parameters and return values must be byValue, but not byReference. The reference or return values of consumers and providers are not the same, but the values are the same. Dubbo does not support remote objects.

Exception

It is suggested that abnormal reporting errors are used rather than return error codes, and exception information can carry more information and have more semantic friendliness.

If you are worried about performance problems, you can use the override () method of fillInStackTrace () out of the exception class as an empty method to make it not a copy of the stack information when necessary.

Query method is not recommended throws checked, otherwise the caller in the query will be too much `try...catch, and can not be processed.`

Service providers should not throw the exception of DAO or SQL to the consumer side. They should package the exception that consumers do not care about in service implementation, otherwise consumers may not be able to serialize the corresponding exception.

Call

Not just because it is a Dubbo call, wrap the call logic eith `try...catch` clause. `try...catch` should be added to the appropriate rollback boundary.

The check logic for the input parameters should be available at the Provider side. For performance considerations, the service implementer may consider adding a service Stub class to the API package to complete the test.

Recommended usage

Configuring the attributes of the consumer side as much as possible on the provider side

the reason is：

- Service providers are more aware of service performance parameters than service users，Such as the timeout time of the call, the reasonable retry times, and so on.
- If a attribute is configurated in provider side, not configurated in consumer side, consumer service will use the attribute in provider side. That is to say, the provider side's attribute can be used as consumer's default value [1]. Otherwise, consumer service will use consumer-side's attribute，but can't cnotrol the provider service,it's usually unreasonable.

Configuring the attributes of the consumer side as much as possible on the provider side，Make the provider service developer think more about the characteristics and quality of the provider side service.

Examples：

```
<dubbo:service interface="com.alibaba.hello.api.HelloService" version="1.0.0" ref="hel
loService"
    timeout="300" retry="2" loadbalance="random" actives="0"
/>

<dubbo:service interface="com.alibaba.hello.api.WorldService" version="1.0.0" ref="hel
loService"
    timeout="300" retry="2" loadbalance="random" actives="0" >
    <dubbo:method name="findAllPerson" timeout="10000" retries="9" loadbalance="leasta
ctive" actives="5" />
<dubbo:service/>
```

The consumer side properties that can be configured on provider are：

1. `timeout` Method call timeout
2. `retries` The number of failed retries, default value is 1 [2]
3. `loadbalance` Load balance algorithm [3]，default algorithm is random `random` ,and polling `roundrobin` 、least active [4] `leastactive`
4. `actives` Consumer side, maximum concurrent call limitation. That is , when the concurrent requests of consumer service reach maximum configuration,the new call will

wait until to catch a timeout error. Configurated in `dubbo:method` (method level configuration) , then the concurrent limitation point at method.Configurated in `dubbo:service` (service level configuration),then the concurrent limitation point at service.

Detailed configuration instructions see：Dubbo configuration introduction

Configuring reasonable provider end properties on provider

```
<dubbo:protocol threads="200" />
<dubbo:service interface="com.alibaba.hello.api.HelloService" version="1.0.0" ref="hel
loService"
    executes="200" >
    <dubbo:method name="findAllPerson" executes="50" />
</dubbo:service>
```

The provider side properties that can be configured on provider service are：

1. `threads` service thread pool size
2. `executes` If concurrent requests number that a provider service handled reach the maximum thead pool count , the new call will wait,then the consumer call may catch a timeout error. Configurated in `dubbo:method` (method level configuration) , then the concurrent limitation point at method.Configurated in `dubbo:service` (service level configuration),then the concurrent limitation point at service.

Configuration management information

Now we have the owner information and organization infomation to differentiate the sites。It's easy to contact with the service owners when there is a problem, please write at least two persons for backup. The information of owners and organizations can be seen in the registry.

application configuration owners,organizations:

```
<dubbo:application owner=”ding.lid,william.liangf” organization=”intl” />
```

service configuration owners:

```
<dubbo:service owner=”ding.lid,william.liangf” />
```

reference configuration owners:

```
<dubbo:reference owner="ding.lid,william.liangf" />
```

`dubbo:service` 、 `dubbo:reference` have no configuration owner, then use the owner configured in `dubbo:application` .

Set up the Dubbo cache file

Provider service list caching file:

```
<dubbo:registry file="${user.home}/output/dubbo.cache" />
```

Notations:

1. You can modify the cahe file path of the application according to the needs. Ensure that the file will not be cleared during the release process.
2. If there are more than one application process, do not use the same file path to avoid the content being overwritten.

This file caches the list of the registry and the list of service providers. With this configuration, when the application is restarted , if the Dubbo registry is not available, the application will read the information from the service provider list from the cache file. That can ensure the availability of the application.

Monitor configuration

1. Expose service with a fixed port, instead of using a random port

```
  In this way, when there is a delay in the registry push, the consumer can also call
the  original provider service address hrough the cache list and succeed。
```

1. Use Dragoon's HTTP monitoring item to monitor the service provider on the registry

   The state of Dragoon monitoring service in the registry : http://dubbo-reg1.hst.xyi.cn.alidc.net:8080/status/com.alibaba.morgan.member.MemberService:1.0.5 Ensure that the service exists on the registry .

2. Service provider,use Dragoon's telnet mommand or shell monitor command

Monitoring service provider port status ： `echo status | nc -i 1 20880 | grep OK | wc -l` , 20880 is the service port

3. Service consumer side, cast the service to EchoService，and call `$echo()` to test whether the provider of the service is available

   eg: `assertEqauls(“OK”, ((EchoService)memberService).$echo(“OK”));`

Don't use the configuration of dubbo.properties file, suggeset to use the configuration of XML

All of the configuration items in the dubbo can be configured in the spring configuration file,and can be configured for a single service.

The Dubbo default value is used if completely not set up , please see the instructions in the article Dubbo configuration introduction .

The relation between attribute name of dubbo.properties and XML

1. application name `dubbo.application.name`

```
<dubbo:application name="myalibaba" >
```

2. registry address `dubbo.registry.address`

```
<dubbo:registry address="11.22.33.44:9090" >
```

3. call timeout `dubbo.service.*.timeout`

   Timeout can be set in multiple configuration items `timeout` ,cover from top to bottom （The top one have a higher priority ）[5]，The coverage strategy of other parameters（ `retries` 、 `loadbalance` 、 `actives` and so on）is：

   Certain method Configuration of a provider service

```
<dubbo:service interface="com.alibaba.xxx.XxxService" >
    <dubbo:method name="findPerson" timeout="1000" />
</dubbo:service>
```

Configuration of a provider specific interface

```
<dubbo:service interface="com.alibaba.xxx.XxxService" timeout="200" />
```

4. Service provider protocol `dubbo.service.protocol` 、Service monitor port `dubbo.service.server.port`

```
<dubbo:protocol name="dubbo" port="20880" />
```

5. Service thread pool size `dubbo.service.max.thread.threads.size`

```
<dubbo:protocol threads="100" />
```

6. No provider throws exceptions (Fast-Fail) when the consumer is started () `alibaba.intl.commons.dubbo.service.allow.no.provider`

```
<dubbo:reference interface="com.alibaba.xxx.XxxService" check="false" />
```

> [1]. Overlay rules for configuration: 1) The method level configuration has a higher priority than the interface level, that is to say,small scope have a high priority 2) Consumer side configuration has a higher priority than provider side, better than global configuration, the last one is the Dubbo hard coded configuration value（Dubbo configuration introduction） ↩

> [2]. With the first call, the call will be called 3 times ↩

> [3]. How to select a service to call when there are multiple Provider services ↩

> [4]. It means that consumer service can call the best provider service, and reduce to call the the slow provider service. ↩

> [5]. `timeout` Can be set in multiple places, configuration items and overlay rules：Dubbo Schema configuration introduction ↩

Capacity plan

The following data for reference：

Use member service project of Dubbo

- Receive 400,000,000 remote calls one day
- Use 12 standard servers to provide services (CPU:8 core, memory: 8G)
- The average load is less than 1 (For 8 core CPU, the load is very low)
- The average response time is 2.3 to 2.5 ms，Network cost about 1.5 to 1.6 ms(Related to the size of the packet)

Use product authorization service project of Dubbo

- Receive 300,000,000 remote calls one day
- Use 8 standard servers to provide services (CPU:8 core, memory: 8G)
- The average load is less than 1 (For 8 core CPU, the load is very low)
- The average response time is 1.4 to 2.8 ms，Network cost about 1.0 to 1.1 ms(Related to the size of the packet)

Performance test report

Test instructions

1. In this performance test, the performance of all Dubbo 2.0 supported protocols in different sizes and data types is tested and compared with the Dubbo 1.0.
2. The overall performance is increased by 1.0 compared with 10%, and the average increase is 10%. The performance improvement of 10%~50% can also be achieved by using the new Dubbo serialization of Dubbo 2.0 .
3. In the stability test, because the underlying communication framework is changed from Mina to netty, the growth of objects in old area is greatly reduced, and the 50 hour operation increases less than 200m and no fullgc.
4. There is a problem: performance of 2.0 is less than 1.0 in 50K data, and it is doubted that it may be a buffer setting problem, and the next version will be further confirmed.

Test environment

Hardware deployment and parameter adjustment

Model	CPU	Memory	Network	Disk	Kernel
Tecal BH620	model name : Intel(R) Xeon(R) CPU E5520 @ 2.27GHz cache size : 8192 KB processor_count : 16	Total System Memory: 6G Hardware Memory Info: Size: 4096MB	eth0: Link is up at 1000 Mbps, full duplex. peth0: Link is up at 1000 Mbps, full duplex.	/dev/sda: 597.9 GB	2.6.18-128.el5xen x86_64

Software architecture

Software name and version	key parameter
java version "1.6.0_18" Java(TM) SE Runtime Environment (build 1.6.0_18-b07) Java HotSpot(TM) 64-Bit Server VM (build 16.0-b13, mixed mode)	-server -Xmx2g -Xms2g -Xmn256m -XX:PermSize=128m -Xss256k -XX:+DisableExplicitGC -XX:+UseConcMarkSweepGC -XX:+CMSParallelRemarkEnabled -XX:+UseCMSCompactAtFullCollection -XX:LargePageSizeInBytes=128m -XX:+UseFastAccessorMethods -XX:+UseCMSInitiatingOccupancyOnly -XX:CMSInitiatingOccupancyFraction=70
jboss-4.0.5.GA	
httpd-2.0.61	KeepAlive On MaxKeepAliveRequests 100000 KeepAliveTimeout 180 MaxRequestsPerChild 1000000 StartServers 5 MaxClients 1024 MinSpareThreads 25 MaxSpareThreads 75 ThreadsPerChild 64 ThreadLimit 128 ServerLimit 16

Test purpose

Expected performance indicators (quantized)

Scene name	Corresponding index name	Range of expected values	Actual value	Whether or not to meet expectations (yes / no)
1k data	Response time	0.9ms	0.79ms	Yes
1k data	TPS	10000	11994	Yes

Expected operating conditions (non quantified, optional)

- The performance of 2.0 is not less than 1, and the performance of the intermodulation of 2.0 and 1.0 is not significantly reduced. In addition to 50K string, the rest are passed
- JVM memory is running stable, no OOM, and there is no reasonable large object in the heap memory. Passed
- CPU, memory, network, disk, file handle are occupied smoothly. Passed
- There is no frequent thread lock, and the number of threads is stable. Passed
- Business thread load balance. Passed

Test script

1. Performance test scence (10 concurrency)
    - Pass in 1K String, do not do anything, return the original
    - Pass in 50K String, do not do anything, return the original
    - Pass in 200K String, do not do anything, return the original
    - Incoming 1K POJO (nested complex person objects) without any processing, return to the original

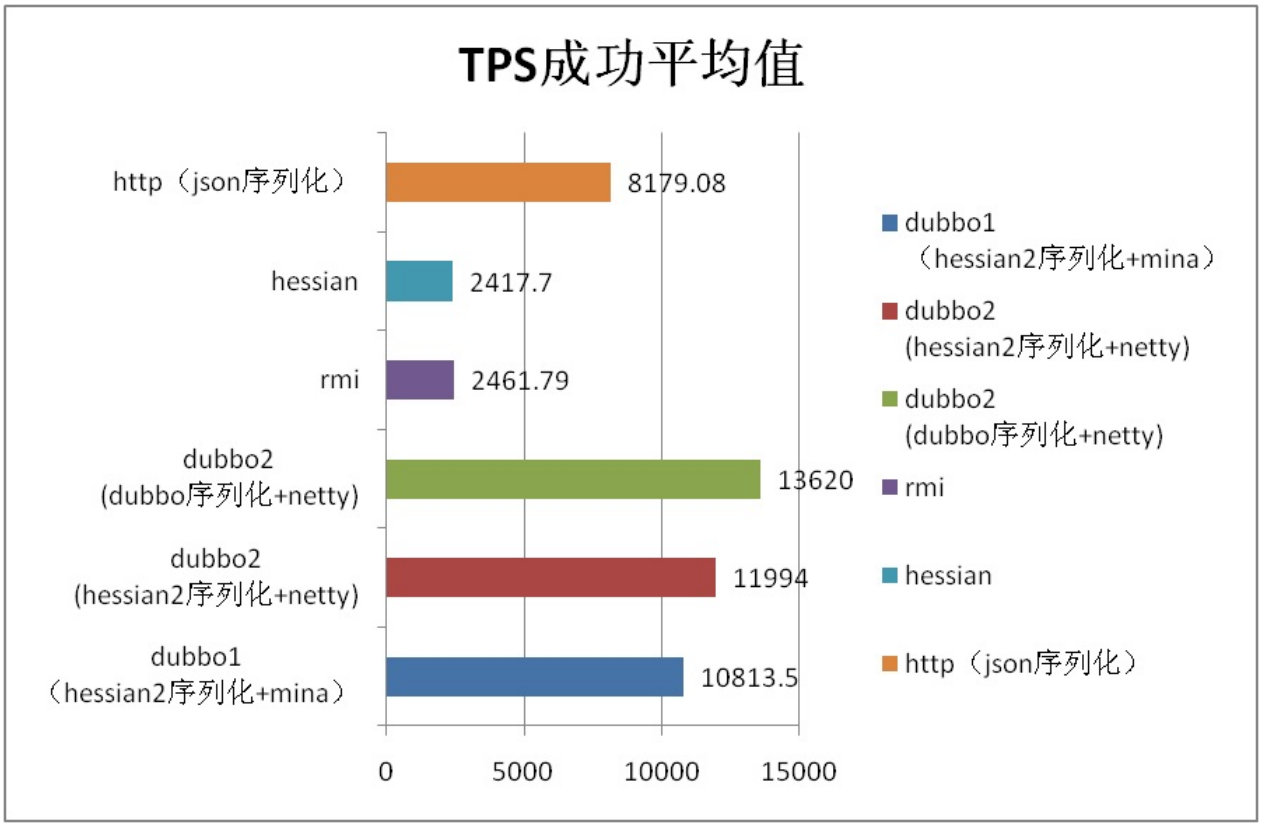
      The above scenario is tested for 10 minutes in Dubbo 1.0, Dubbo 2.0 (hessian2 serialization), Dubbo 2.0 (Dubbo serialization), RMI, Hessian 3.2.0, HTTP (JSON serialization). It mainly examines the performance of serialization and network IO, so the server has no business logic. 10 is to consider the concurrent HTTP protocol in high with the use of CPU high rate may hit the bottleneck.
2. Concurrent scene (20 concurrency) 1K String is introduced into the server segment for 1W times, and a random number is regenerated each time and then assembled. Examine whether business threads can be assigned to each CPU.
3. Stability scence (20 concurrency) At the same time, we call the 1 parameter String (5K) method, the 1 parameter is the person object method, the 1 parameter is map (the value is 3 person), and it runs for 50 hours continuously.
4. High pressure scene (20 concurrency) On the basis of the stability scenario, the providers and consumers are arranged into 2 sets (one machine and 2 instances), and the parameters of String are 20byte to 200K, and are randomly transformed every 10 minutes.

Test result

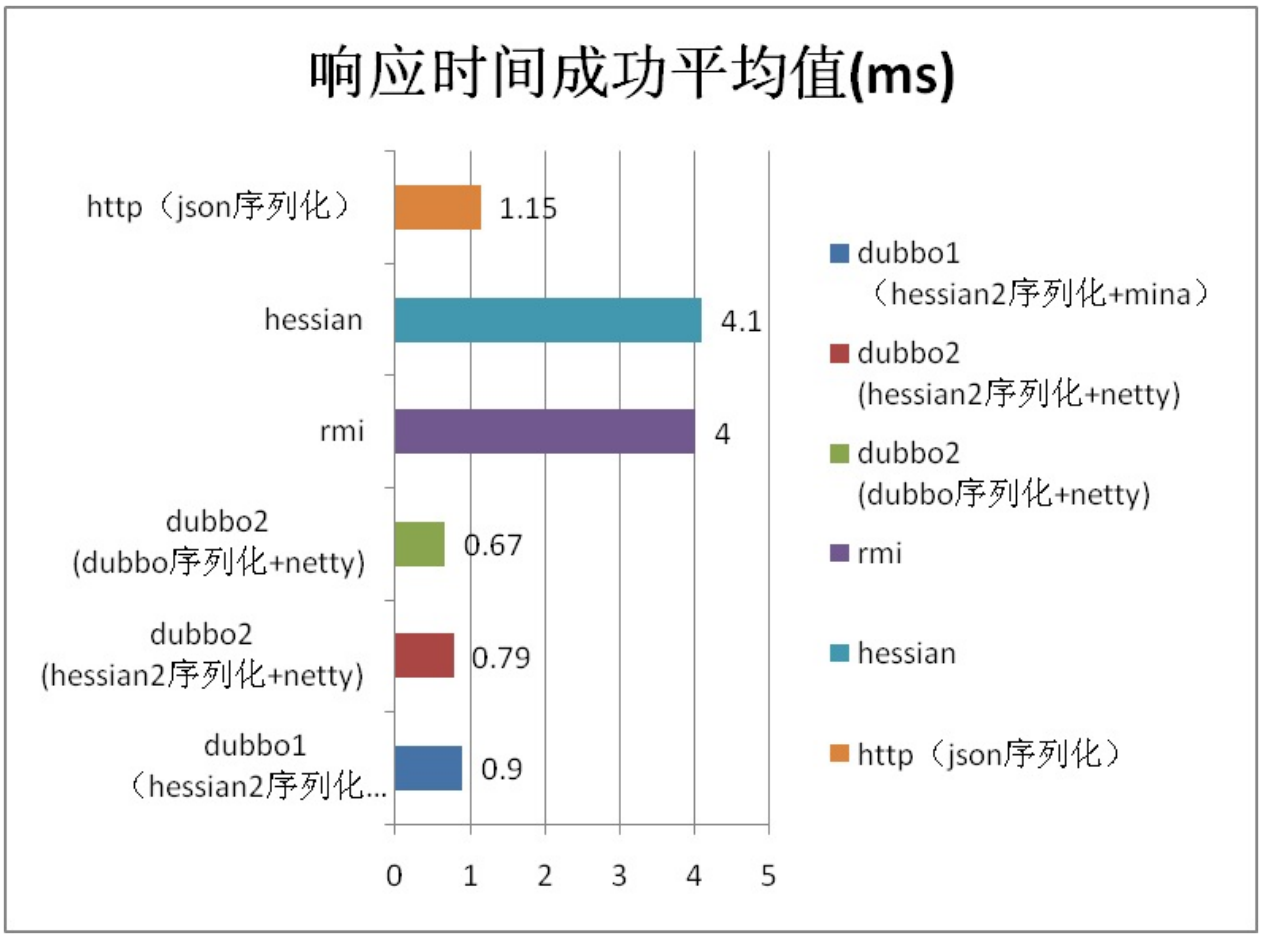
Scene name: scence POJO

	TPS success avg value	Response time avg value(ms)
dubbo1 (hessian2 serialization+mina)	10813.5	0.9
dubbo2 (hessian2 serialization+netty)	11994	0.79
dubbo2 (dubbo serialization+netty)	13620	0.67
rmi	2461.79	4
hessian	2417.7	4.1
http（json serialization）	8179.08	1.15
The default percentage of 2.0 and 1.0	10.92	-12.22
Dubbo serialization compared to the percentage of hessian2 serialization	13.56	-15.19

POJO TPS



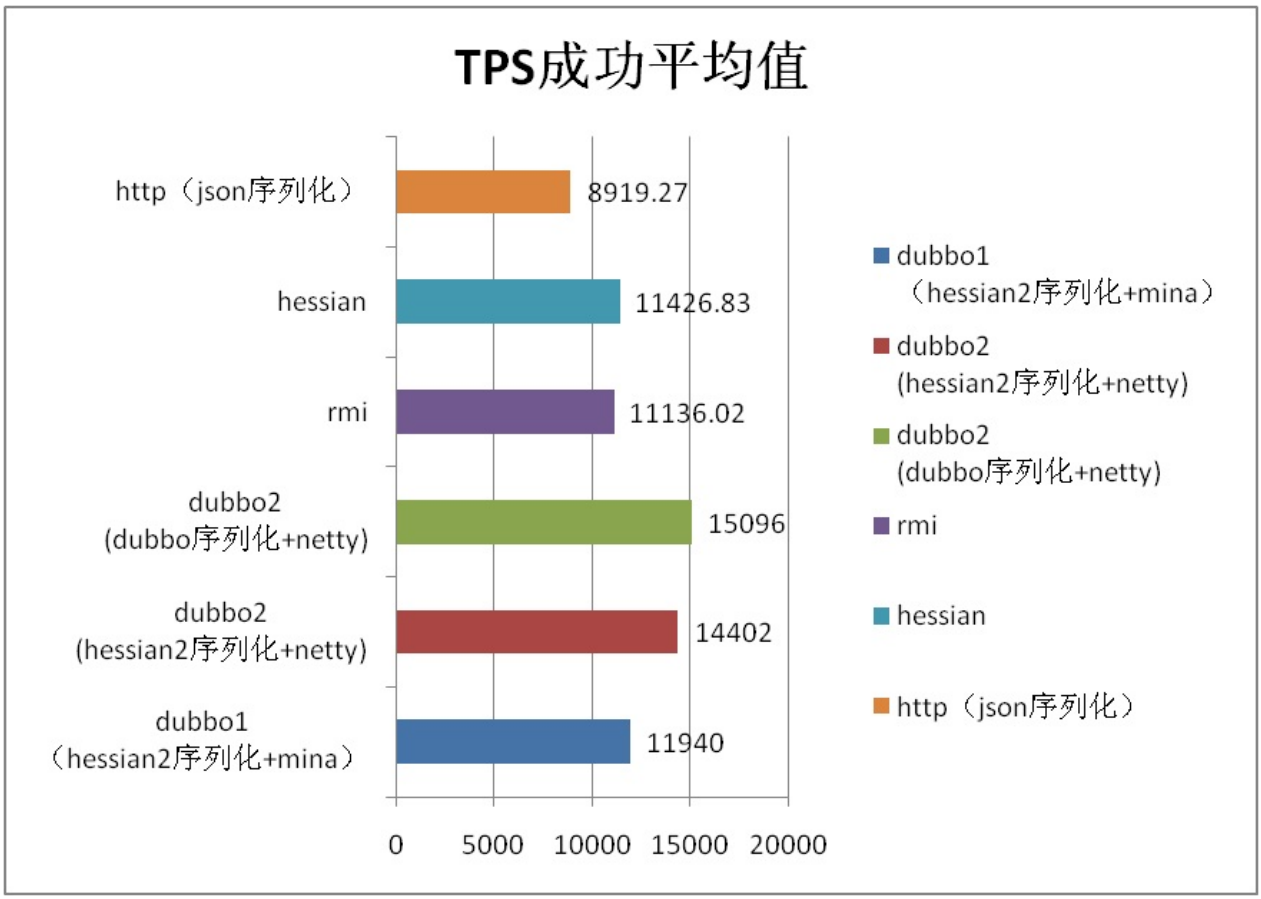
POJO Response



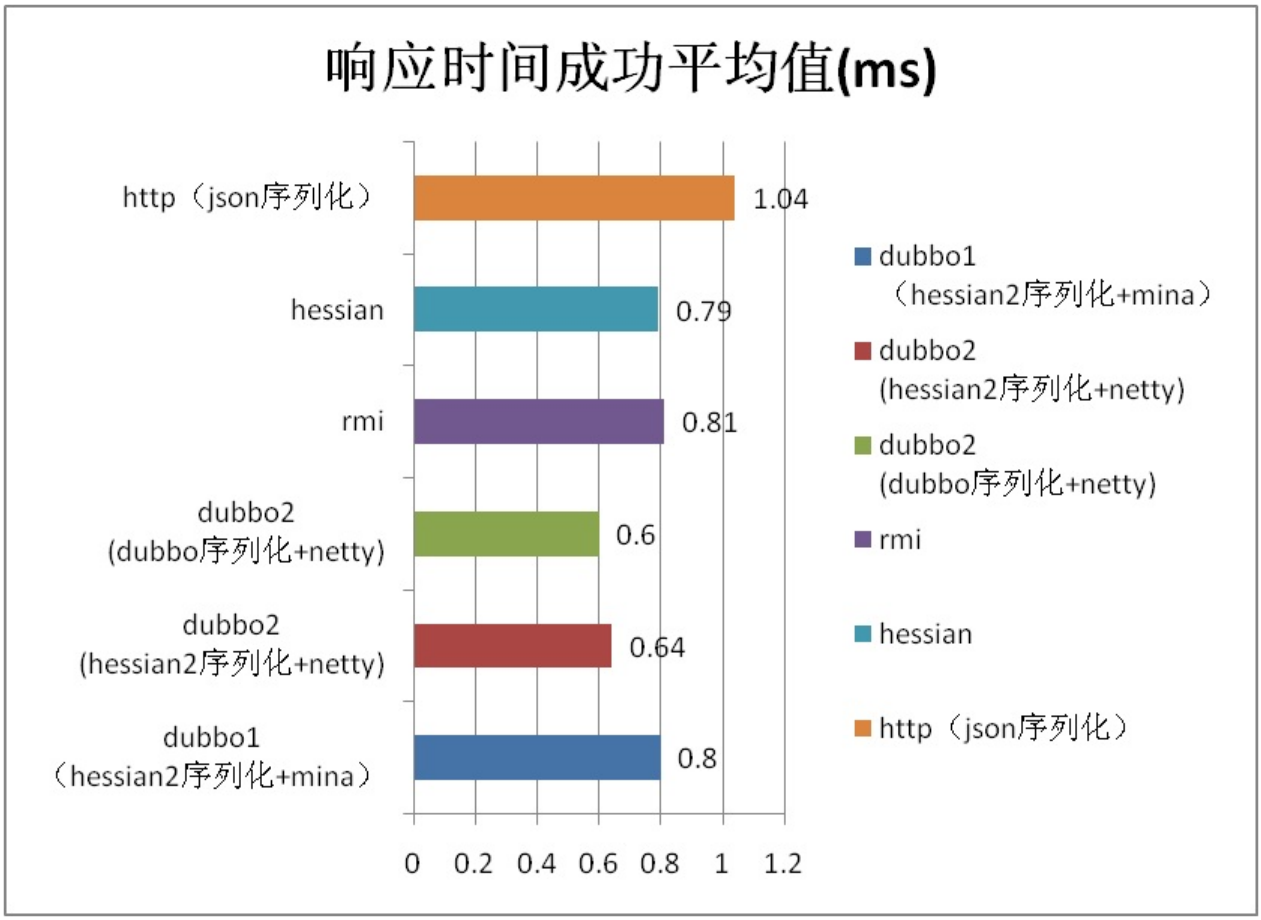
Scene name: scence 1k string

	TPS success avg value	Response time avg value(ms)
dubbo1（hessian2 serialization+mina）	11940	0.8
dubbo2 (hessian2 serialization+netty)	14402	0.64
dubbo2 (dubbo serialization+netty)	15096	0.6
rmi	11136.02	0.81
hessian	11426.83	0.79
http（json serialization）	8919.27	1.04
The default percentage of 2.0 and 1.0	20.62	-20.00
Dubbo serialization compared to the percentage of hessian2 serialization	4.82	-6.25

1k TPS


TPS成功平均值
http（json序列化）
8919.27
hessian
11426.83
rmi
11136.02
dubbo2
(dubbo序列化+netty)
15096
dubbo2
(hessian2序列化+netty)
14402
dubbo1
（hessian2序列化+mina）
11940
0
5000
10000
15000
20000
dubbo1
（hessian2序列化+mina）
dubbo2
(hessian2序列化+netty)
dubbo2
(dubbo序列化+netty)
rmi
hessian
http（json序列化）

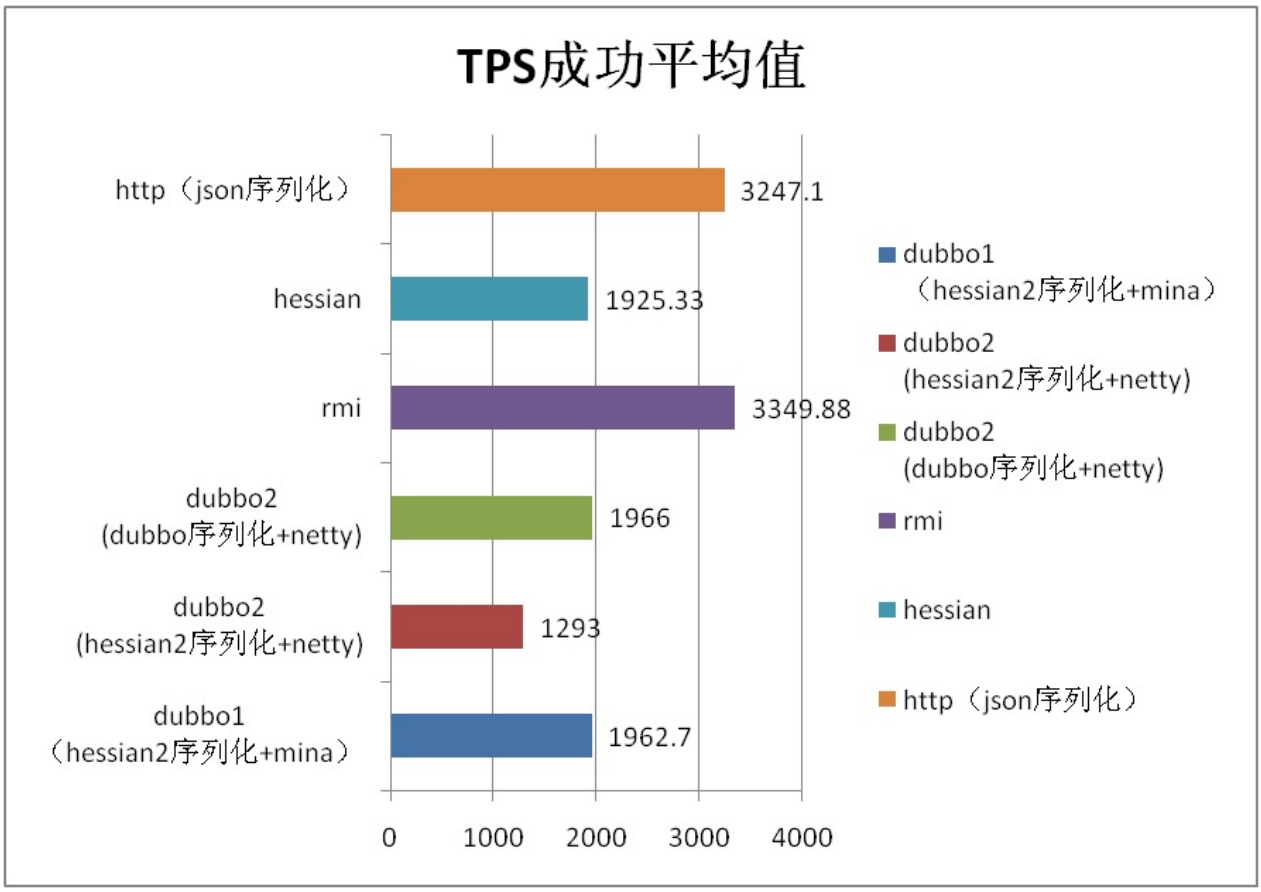
1k Response


响应时间成功平均值(ms)
http（json序列化）
1.04
hessian
0.79
rmi
0.81
dubbo2
(dubbo序列化+netty)
0.6
dubbo2
(hessian2序列化+netty)
0.64
dubbo1
（hessian2序列化+mina）
0.8
0
0.2
0.4
0.6
0.8
1
1.2
dubbo1
（hessian2序列化+mina）
dubbo2
(hessian2序列化+netty)
dubbo2
(dubbo序列化+netty)
rmi
hessian
http（json序列化）

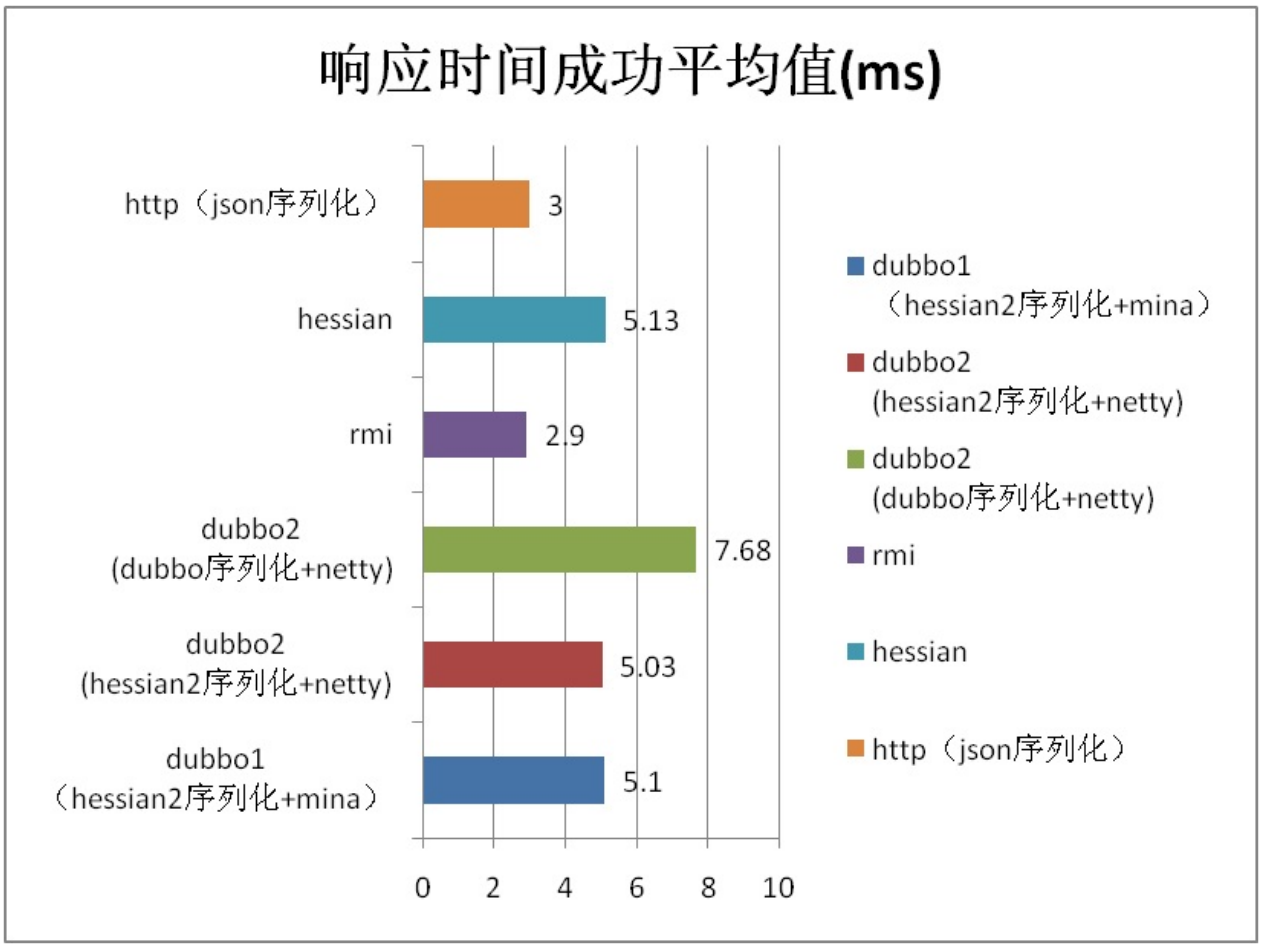
Scene name: scence 50k string

TPS success avg value	Response time avg value(ms)
dubbo1（hessian2 serialization+mina	1962.7
dubbo2 (hessian2 serialization+netty)	1293
dubbo2 (dubbo serialization+netty)	1966
rmi	3349.88
hessian	1925.33
http（json serialization）	3247.1
The default percentage of 2.0 and 1.0	-34.12
Dubbo serialization compared to the percentage of hessian2 serialization	52.05

50K TPS



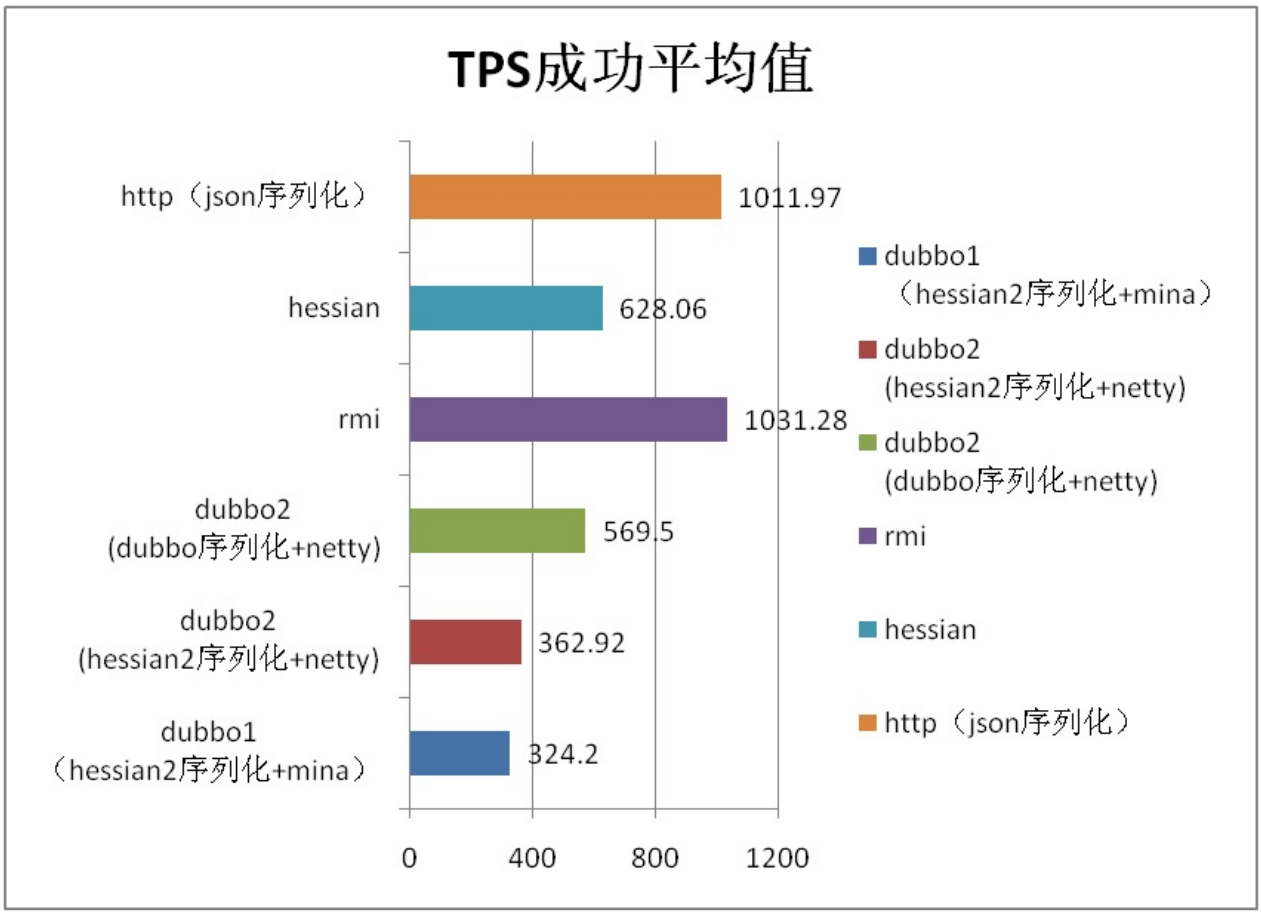
50K Response



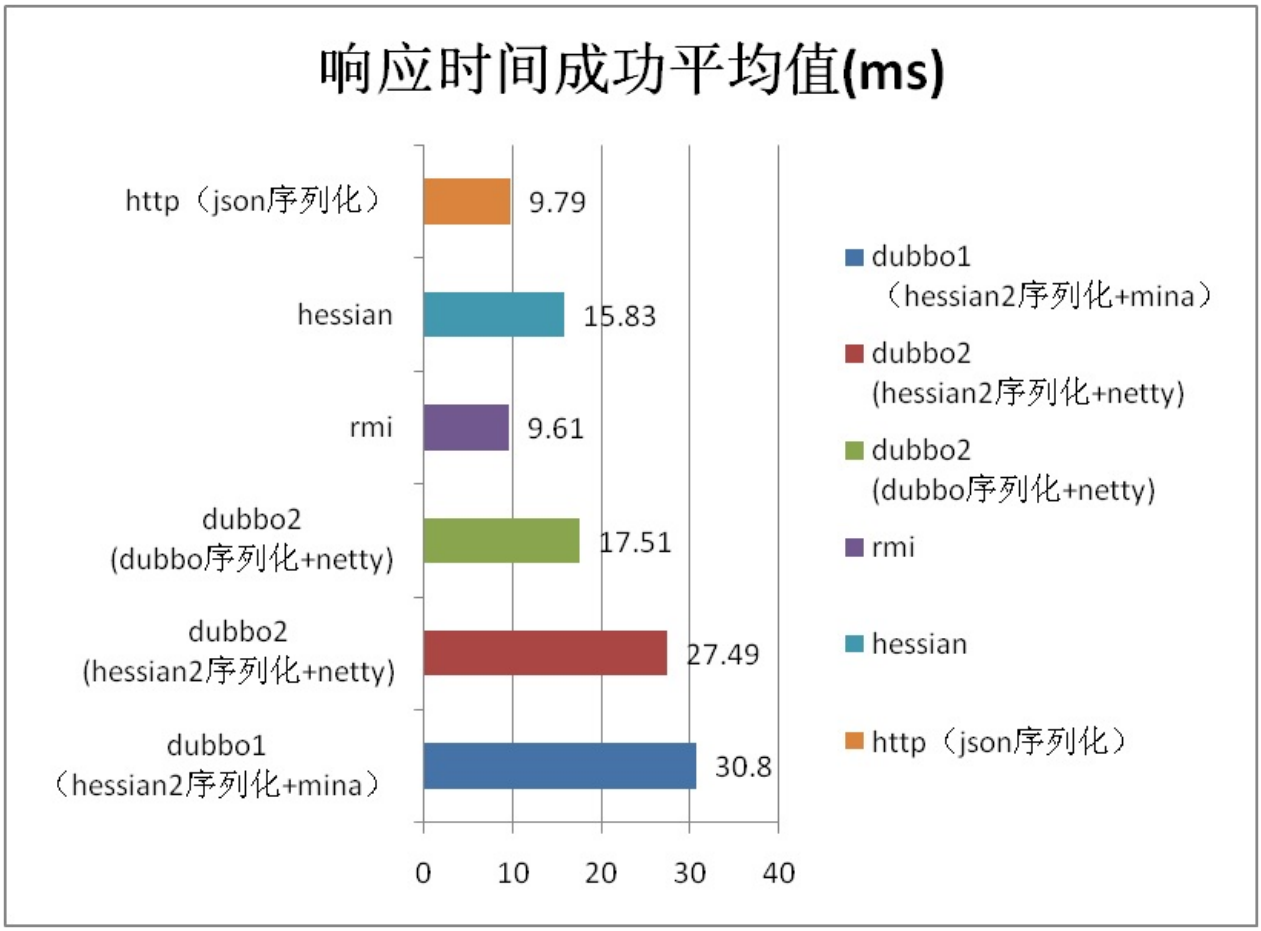
Scene name: scence 200k string

TPS success avg value	Response time avg value(ms)
dubbo1（hessian2 serialization+mina）	324.2
dubbo2 (hessian2 serialization+netty)	362.92
dubbo2 (dubbo serialization+netty)	569.5
rmi	1031.28
hessian	628.06
http（json serialization）	1011.97
The default percentage of 2.0 and 1.0	11.94
Dubbo serialization compared to the percentage of hessian2 serialization	56.92

200K TPS



200K Response



Test analysis

Performance analysis and evaluation

The performance test conclusion of Dubbo 2 has been improved and improved from performance, memory footprint and stability. Because of its memory management, the change of Mina into netty greatly reduces the 1 version of the large memory sawtooth in high concurrency and large data.

Performance comparison analysis (new and old environment, different data magnitude, etc.)

The performance of Dubbo 2 is compared with that of Dubbo 1, which is all hessian2 serialization. The performance is improved (except for 50K String). See the performance data of the fifth chapter in detail.

For compatibility default serialization and 1 consistent with hessian2, such as have higher requirements on the performance of Dubbo serialization can be used, which is in the process of complicated object, can be obtained in 50% large data upgrade (but it is not recommended for use Dubbo protocol).

The purpose of Dubbo is to meet the RPC calls with high concurrent and small data volume. The performance is not good under large data volume. It is recommended to use RMI or HTTP protocol.

Test limitation analysis (optional)

This performance test examines the performance of the Dubbo itself, and the performance of the actual use needs to be verified.

Because the performance of Dubbo itself is in millisecond and the base number is small, performance improvement may not change the performance of the application as a whole.

All the monitoring charts are not listed because of the limit of length.

Test coverage report

Based on version `2.0.12` , Statistics on 2012-02-03

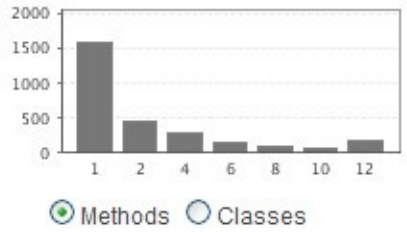
Lines of code
35,406
54,469 lines
17,315 statements
423 files

Classes
445
90 packages
2,762 methods
+347 accessors

Comments
9.2%
3,575 lines
32.1% docu. API
1,839 undocu. API
52 commented LOCs

Duplications
3.8%
2,077 lines
99 blocks
47 files

Complexity
3.8 /method
23.5 /class
24.7 /file
Total: 10,467



Code coverage
40.0%
43.4% line coverage
34.1% branch coverage

Test success
100.0%
0 failures
0 errors
817 tests
+102 skipped
55.9 sec

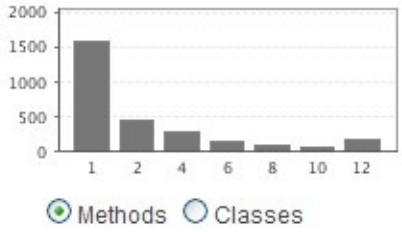
Lines of code
35,406
54,469 lines
17,315 statements
423 files

Classes
445
90 packages
2,762 methods
+347 accessors

Comments
9.2%
3,575 lines
32.1% docu. API
1,839 undocu. API
52 commented LOCs

Duplications
3.8%
2,077 lines
99 blocks
47 files

Complexity
3.8 /method
23.5 /class
24.7 /file
Total: 10,467



Code coverage
40.0%
43.4% line coverage
34.1% branch coverage

Test success
100.0%
0 failures
0 errors
817 tests
+102 skipped
55.9 sec

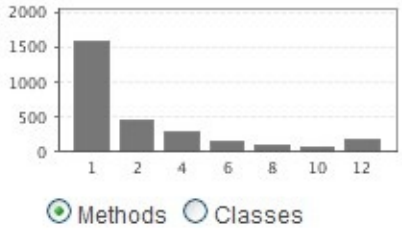
Lines of code
35,406
54,469 lines
17,315 statements
423 files

Classes
445
90 packages
2,762 methods
+347 accessors

Comments
9.2%
3,575 lines
32.1% docu. API
1,839 undocu. API
52 commented LOCs

Duplications
3.8%
2,077 lines
99 blocks
47 files

Complexity
3.8 /method
23.5 /class
24.7 /file
Total: 10,467



Code coverage
40.0%
43.4% line coverage
34.1% branch coverage

Test success
100.0%
0 failures
0 errors
817 tests
+102 skipped
55.9 sec

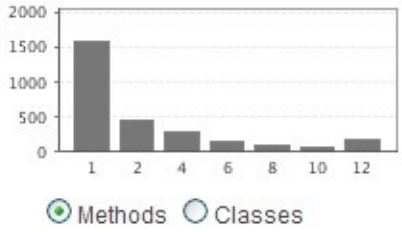
Lines of code
35,406
54,469 lines
17,315 statements
423 files

Classes
445
90 packages
2,762 methods
+347 accessors

Comments
9.2%
3,575 lines
32.1% docu. API
1,839 undocu. API
52 commented LOCs

Duplications
3.8%
2,077 lines
99 blocks
47 files

Complexity
3.8 /method
23.5 /class
24.7 /file
Total: 10,467



Code coverage
40.0%
43.4% line coverage
34.1% branch coverage

Test success
100.0%
0 failures
0 errors
817 tests
+102 skipped
55.9 sec

Violations
2,996

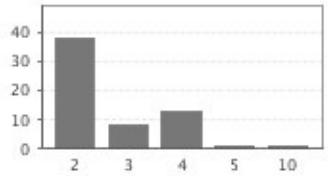
Rules compliance
80.0%

Blocker	0	
Critical	232	
Major	1,590	
Minor	1,145	
Info	29	

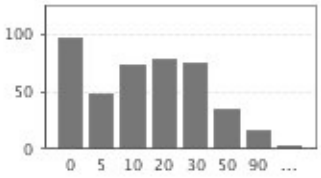
Package tangle index
4.1%
> 11 cycles

Dependencies to cut
7 between packages
14 between files

LCOM4
1.2 /class
14.4% files having LCOM4>1



RFC
25 /class



Violations
2,996

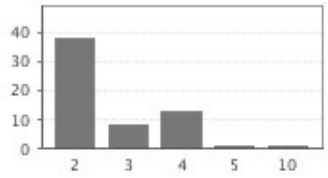
Rules compliance
80.0%

Blocker	0	
Critical	232	
Major	1,590	
Minor	1,145	
Info	29	

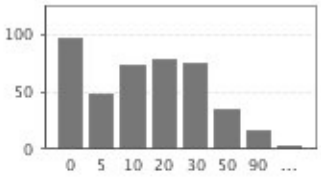
Package tangle index
4.1%
> 11 cycles

Dependencies to cut
7 between packages
14 between files

LCOM4
1.2 /class
14.4% files having LCOM4>1



RFC
25 /class



Violations
2,996

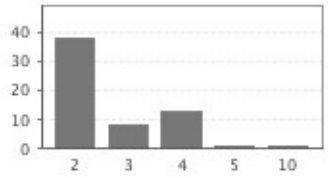
Rules compliance
80.0%

Blocker	0	
Critical	232	
Major	1,590	
Minor	1,145	
Info	29	

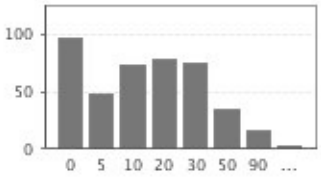
Package tangle index
4.1%
> 11 cycles

Dependencies to cut
7 between packages
14 between files

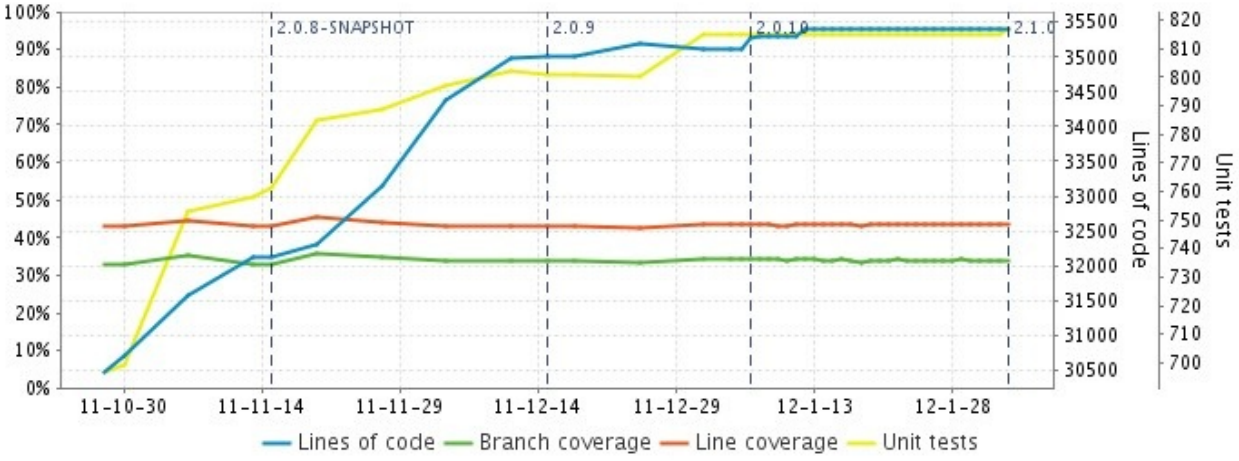
LCOM4
1.2 /class
14.4% files having LCOM4>1



RFC
25 /class



Name	Rules compliance	Line coverage	Branch coverage
Dubbo Common Module	69.9%	58.1%	43.3%
Dubbo Remoting Module	82.6%	25.3%	20.7%
Dubbo Netty Remoting Module	79.9%	75.7%	53.6%
Dubbo Mina Remoting Module	81.2%	69.1%	42.7%
Dubbo Grizzly Remoting Module	79.6%	0.0%	0.0%
Dubbo P2P Remoting Module	85.1%	0.0%	0.0%
Dubbo Http Remoting Module	99.0%	0.0%	0.0%
Dubbo RPC Module	89.8%	23.9%	17.1%
Dubbo Default RPC Module	89.0%	61.3%	47.7%
Dubbo InJVM RPC Module	92.6%	76.2%	50.0%
Dubbo RMI RPC Module	88.3%	48.5%	29.8%
Dubbo Hessian RPC Module	87.9%	72.2%	39.1%
Dubbo Cluster Module	85.1%	66.3%	51.9%
Dubbo Registry Module	85.6%	1.7%	0.6%
Dubbo Default Registry Module	84.2%	48.1%	10.0%
Dubbo Multicast Registry Module	93.5%	65.8%	50.0%
Dubbo Zookeeper Registry Module	81.3%	12.9%	6.7%
Dubbo Monitor Module	93.9%	60.7%	37.5%
Dubbo Default Monitor Module	63.4%	47.1%	17.7%
Dubbo Config Module	88.6%	55.3%	46.5%
Dubbo Container Module	86.7%	15.5%	8.7%
Dubbo All In One			
Dubbo Simple Registry Module	98.7%	5.2%	0.0%
Dubbo Simple Monitor Module	87.2%	3.0%	0.6%
Dubbo Demo Module	100.0%		
Dubbo Demo Consumer Module	66.7%	0.0%	0.0%
Dubbo Demo Provider Module	72.7%	0.0%	



Dependency | Suspect dependency (cycle) | - uses > | - uses >

Dubbo Simple Registry Module	-																								
Dubbo Simple Monitor Module		-																							
Dubbo Demo Consumer Module			-																						
Dubbo Demo Provider Module				-																					
Dubbo All In One	1	1	1	1	-																				
Dubbo Mina Remoting Module					1	-																			
Dubbo P2P Remoting Module					1		-																		
Dubbo Grizzly Remoting Module					1			-																	
Dubbo Config Module					1				-																
Dubbo Default Monitor Module					1				1	-															
Dubbo Default Registry Module					1				1		-														
Dubbo Multicast Registry Module					1							-													
Dubbo Zookeeper Registry Module					1								-												
Dubbo Registry Module									1		1	1	1	-											
Dubbo Default RPC Module					1				1		1				-										
Dubbo Netty Remoting Module					1				1		1				1	-									
Dubbo Container Module														1	1		-								
Dubbo Monitor Module									1	1								-							
Dubbo Cluster Module														1					-						
Dubbo Hessian RPC Module					1															-					
Dubbo RMI RPC Module					1				1												-				
Dubbo InJVM RPC Module					1																	-			
Dubbo RPC Module															1			1	1	1	1	1	-		
Dubbo Http Remoting Module					1															1				-	
Dubbo Demo Module			1	1																					-
Dubbo Remoting Module						1	1	1	1						1	1									-
Dubbo Common Module																	1						1	1		1	-