
Table of Contents

Introduction	1.1
1 How To Build	1.2
2 Architecture	1.3
3 How SPI Works	1.4
4 Init, Process, Protocols	1.5
5 SPI Extensions	1.6
5.1 Protocol	1.6.1
5.2 Filter	1.6.2
5.3 InvokerListener	1.6.3
5.4 ExporterListener	1.6.4
5.5 Cluster	1.6.5
5.6 Router	1.6.6
5.7 LoadBalance	1.6.7
5.8 Merger	1.6.8
5.9 Registry	1.6.9
5.10 Monitor	1.6.10
5.11 ExtensionFactory	1.6.11
5.12 ProxyFactory	1.6.12
5.13 Compiler	1.6.13
5.14 Dispatcher	1.6.14
5.15 Threadpool	1.6.15
5.16 Serialization	1.6.16
5.17 Remoting	1.6.17
5.18 Exchanger	1.6.18
5.19 Networker	1.6.19
5.20 TelnetHandler	1.6.20
5.21 StatusChecker	1.6.21

5.22 Container	1.6.22
5.23 PageHandler	1.6.23
5.24 Cache	1.6.24
5.25 Validation	1.6.25
5.26 LoggerAdapter	1.6.26
6 Contract	1.7
7 Code Style	1.8
9 Versions	1.9
10 Contribution	1.10
11 Checklist	1.11
12 Code Smell	1.12
13 TCK	1.13

dubbo-dev-book

This book dives into the design principles of dubbo, mainly covers the following topics: extension, coding styles, versio, build, etc.

Source Code Build

Checkout

checkout the latest project source code with commands below:

```
git clone https://github.com/alibaba/dubbo dubbo
```

Branches

We use `master` as the major branch for new feature development, and use other branches for maintenance. Tags for all versions can be checked via <https://github.com/alibaba/dubbo/tags>.

Building

Dubbo relies on `maven` as the building tool.

Requirements:

- Java above 1.5 version
- Maven version 2.2.1 or above

The following `MAVEN_OPTS` should be configured before building:

```
export MAVEN_OPTS=-Xmx1024m -XX:MaxPermSize=512m
```

build with below command:

```
mvn clean install
```

skip testing using below building command:

```
mvn install -Dmaven.test.skip
```

Building jar package of source code

build Dubbo source code jar package with below command, which can debug Dubbo source code.

```
mvn clean source:jar install -Dmaven.test.skip
```

IDE support

use below command to generate IDE.

IntelliJ Idea

```
mvn idea:idea
```

Eclipse

```
mvn eclipse:eclipse
```

Importing into eclipse

Firstly, a maven repository needs to be configured in eclipse. Define `M2_REPO` and point it to the local maven repository by clicking `Preferences -> Java -> Build Path -> Classpath` .

Use the following maven command as well:

```
mvn eclipse:configure-workspace -Declipse.workspace=/path/to/the  
/workspace/
```

1: view the source code through <https://github.com/alibaba/dubbo> 2: path under UNIX is `${HOME}/.m2/repository`, path under Windows is `C:\Documents and Settings\m2\repository`

Framework Design

Overall design

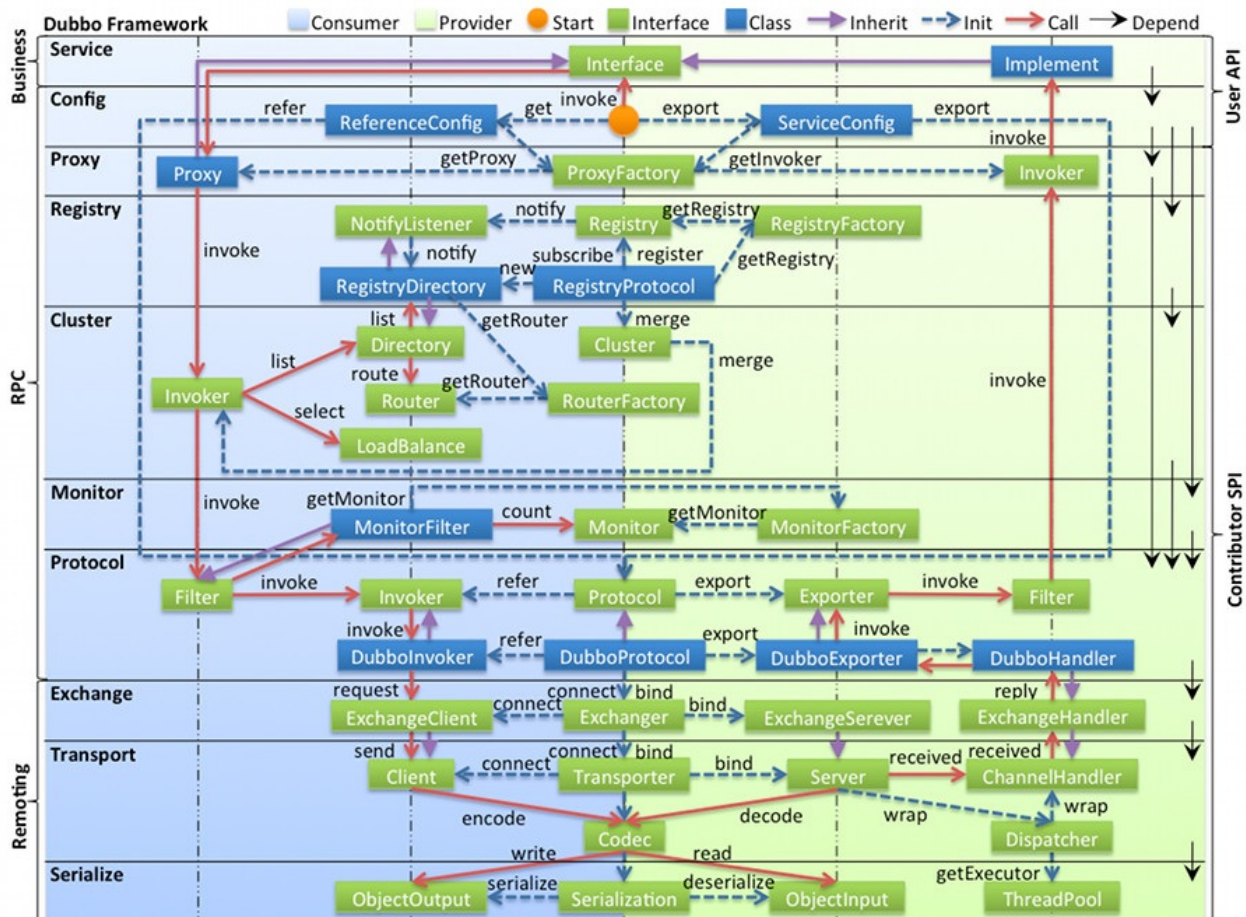


Image description:

- Left area with light blue background shows service consumer interfaces, Right area with light green background shows service provider interfaces, center area shows both side interfaces.
- The image is divided into 10 layers from the bottom to the top, and the layers are one-way dependence. The black arrow on the right represents the dependency between layers, and each layer can be stripped from the upper layer to be reused, the Service and Config layers are API, and the other layers are SPI.
- Green boxes are extension interfaces, blue boxes are implementation classes, image only shows implementation class of associated layers.

- The blue dashed line is the initialization process, which is assembly chain when starting, red line for the method call process, which is calling chain when running, purple triangle arrow is inherited, can treat subclass as the same node of parent class, text of lines are the method invocation.

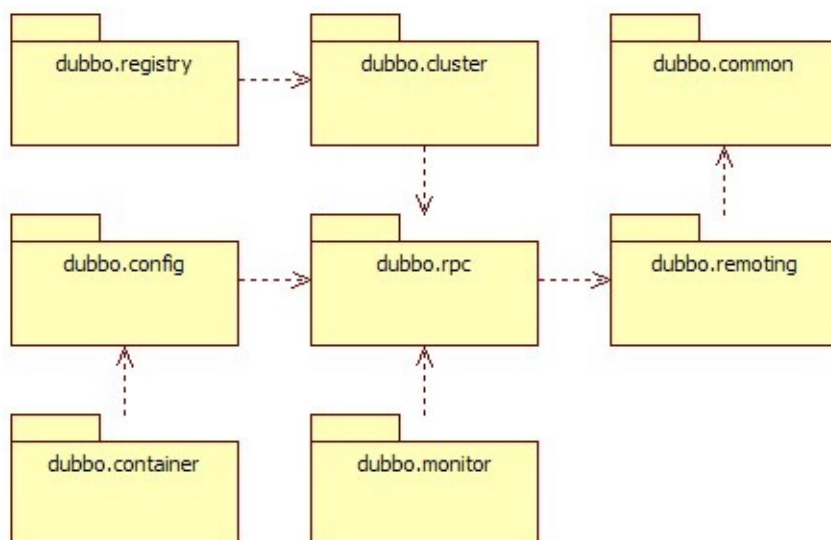
Layer description

- **config layer**: external config interface, `ServiceConfig` and `ReferenceConfig` is the center of the layer, you can directly initialize config class, also can generate config class by spring.
- **proxy layer**: transparent proxy of service interface, generate client Stub of service and server Skeleton of service, `ServiceProxy` is the center, extension interface is `ProxyFactory` .
- **registry layer**: encapsulation of service registry and discovery, service URL is the center, extension interfaces are `RegistryFactory` , `Registry` and `RegistryService` .
- **cluster layer**: encapsulation of cluster of multiple providers and load balance, and bridging registration center, `Invoker` is the center, extension interfaces are `Cluster` , `Directory` , `Router` , `LoadBalance` .
- **monitor layer**: monitor of RPC call times and call execute time, `Statistics` is the center, extension interface are `MonitorFactory` , `Monitor` , `MonitorService` .
- **protocol layer**: encapsulation of RPC, `Invocation` and `Result` are the center, extension interfaces are `Protocol` , `Invoker` , `Exporter` .
- **exchange layer**: encapsulation of request and response, synchronous transfer asynchronous, `Request` and `Response` are the center, extension interfaces are `Exchanger` , `ExchangeChannel` , `ExchangeClient` , `ExchangeServer` .
- **transport layer**: abstraction of mina and netty, `Message` is the center, extension interfaces are `Channel` , `Transporter` , `Client` , `Server` , `Codec` .
- **serialize layer**: reusable tools, extension interfaces are `Serialization` , `ObjectInput` , `ObjectOutput` , `ThreadPool` .

Relationship description

- In RPC, Protocol is the core layer, it means that you can complete RPC calling by Protocol + Invoker + Exporter, then filter at the main process of Invoker.
- Consumer and Provider are abstraction concepts, just want you have a more intuitive understanding of which classes belong to the client and server side, the reason not use Client and Server is that Dubbo uses Provider, Consumer, Registry, Monitor divide logical topology node in many scenes, keep the concept of unity.
- Cluster is external concept, the purpose of Cluster is that make various Invoker disguise to one Invoker, so that we just pay attention to the Invoker in Protocol layer, adding Cluster or removing Cluster will not affect other layers, because we don't need Cluster when only have one provider.
- The Proxy layer encapsulates the transparent proxy for all interfaces, and in other layers with Invoker as the center, turn Invoker into interface, or turn interface implementation into Invoker by Proxy only when exposing to user. RPC still work even removing Proxy layer, but not so transparent, making remote service calling don't look like local service calling.
- Remoting is the implemetation of Dubbo protocols, you can remove Remoting if choosing RMI. The Remoting is divided into Transport layer and Exchange layer, Transport layer is responsible for one-way message transmission, it's abstraction of Mina, Netty, Grizzly, it also can extend UDP transmission. The Exchange layer encapsulates the Request-Response semantics over the transport layer.
- Actually Registry and Monitor are not at the same layer, they are independent nodes, draw them together by layer just for global view.

Modules packaging



Modules description:

- **dubbo-common module:** includes Util classes and common modules.
- **dubbo-remoting module:** is Dubbo protocol implementation, no need to use this module if using RMI for RPC.
- **dubbo-rpc module:** abstraction of various protocols, and dynamic proxy, only one to one call, not concerned with the management of cluster.
- **dubbo-cluster module:** disguise many service providers as one provider, including load balancing, fault tolerance, routing, etc. the address list of clusters can be either static or send by registry.
- **dubbo-registry module:** Based on the cluster of registry send address and the abstraction of various registry centers.
- **dubbo-monitor module:** statistics of service call times, call time, call chain tracked services.
- **dubbo-config module:** is Dubbo external API, users use Dubbo by Config, hide Dubbo details.
- **dubbo-container module:** is a Standlone container, just use Main method to load Spring, because usually service no need Tomcat/JBoss features.

Package dividing according to the layer structure, and the difference with layer dividing:

- container is service container, for service running deployment, not showed in the image.
- protocol layer and proxy layer are placed in RPC module, they are the core of

RPC module, you can use these 2 layers complete RPC call when only have 1 provider.

- transport layer and exchange layer are placed in remoting module, for RPC call base communication.
- serialize layer is placed in common module, for reuse.

Dependence relationship

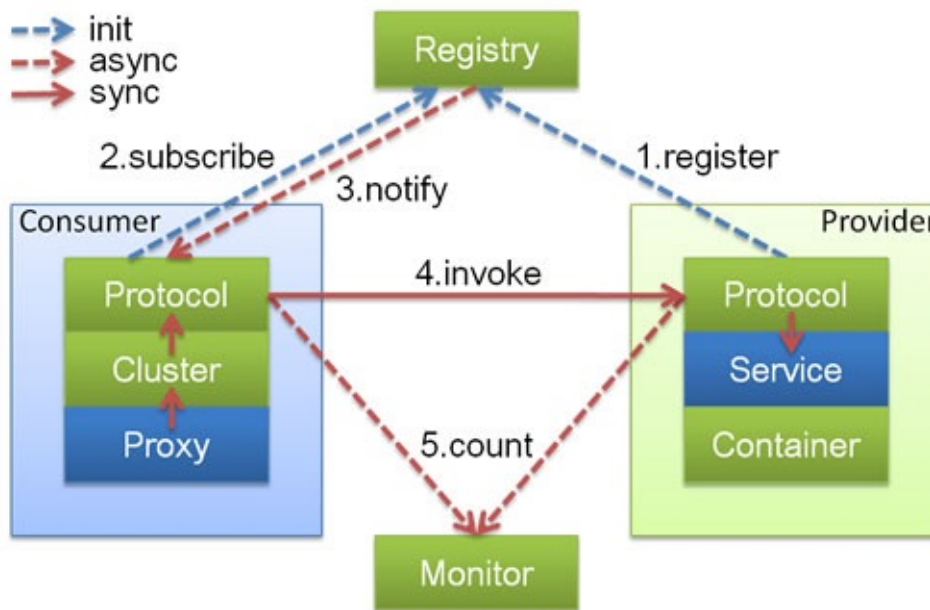
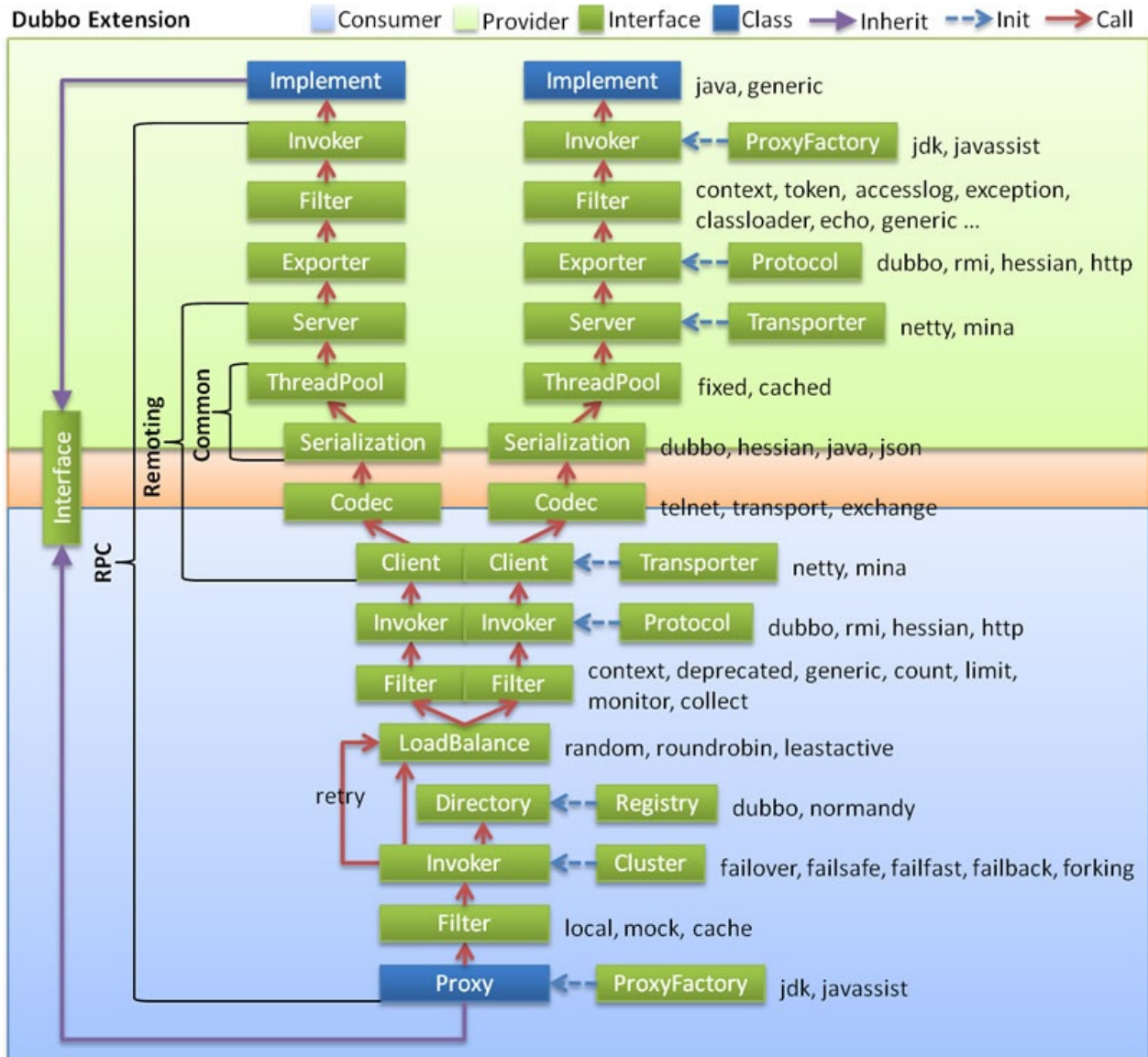


Image description:

- The boxes in image, Protocol, Cluster, Proxy, Service, Container, Registry, Monitor represent layer or module, the blues mean interactive with business, the greens mean only internal interactive with Dubbo.
- The background boxes in image, Consumer, Provider, Registry, Monitor represent deployment logical topology node.
- The blue dashed line in the image is called for initialization, the red dashed line is called asynchronously for runtime, and the red line is called synchronously for runtime.
- The image only includes RPC layer, don't includes Remoting layer, the whole Remoting hides in Protocol layer.

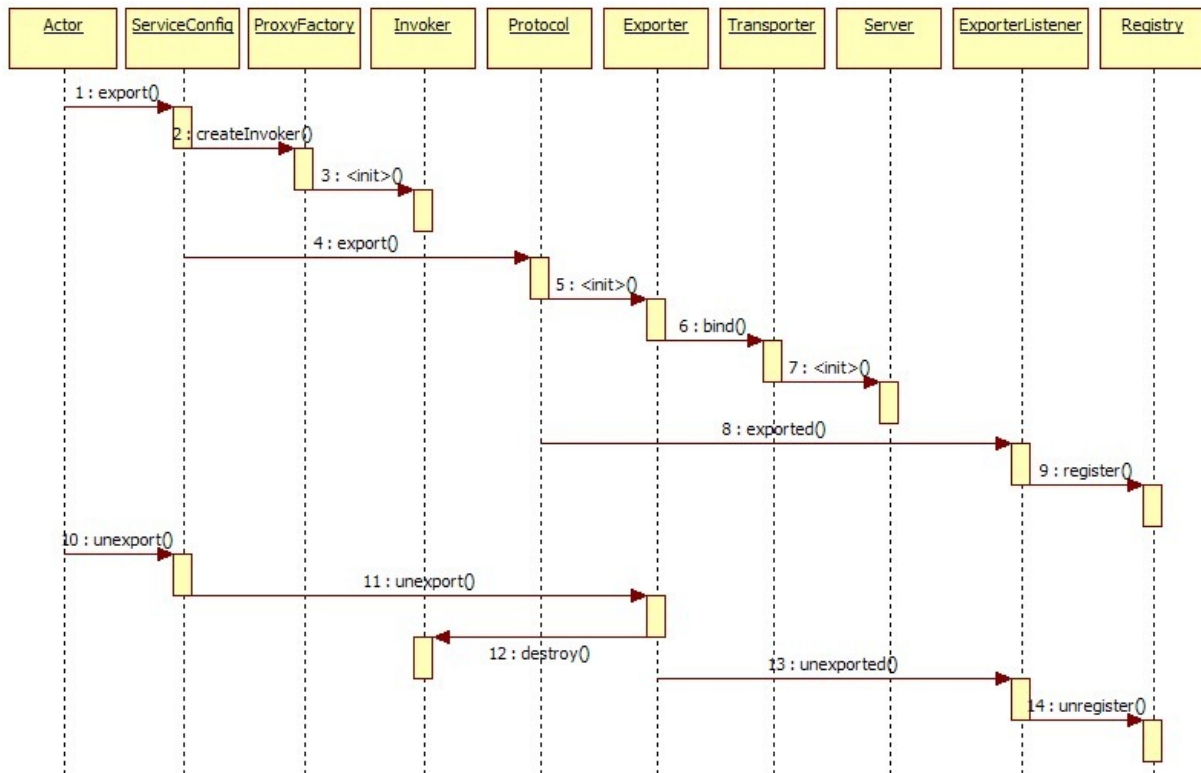
Call chain

Expand the red call chain of the overall design map:



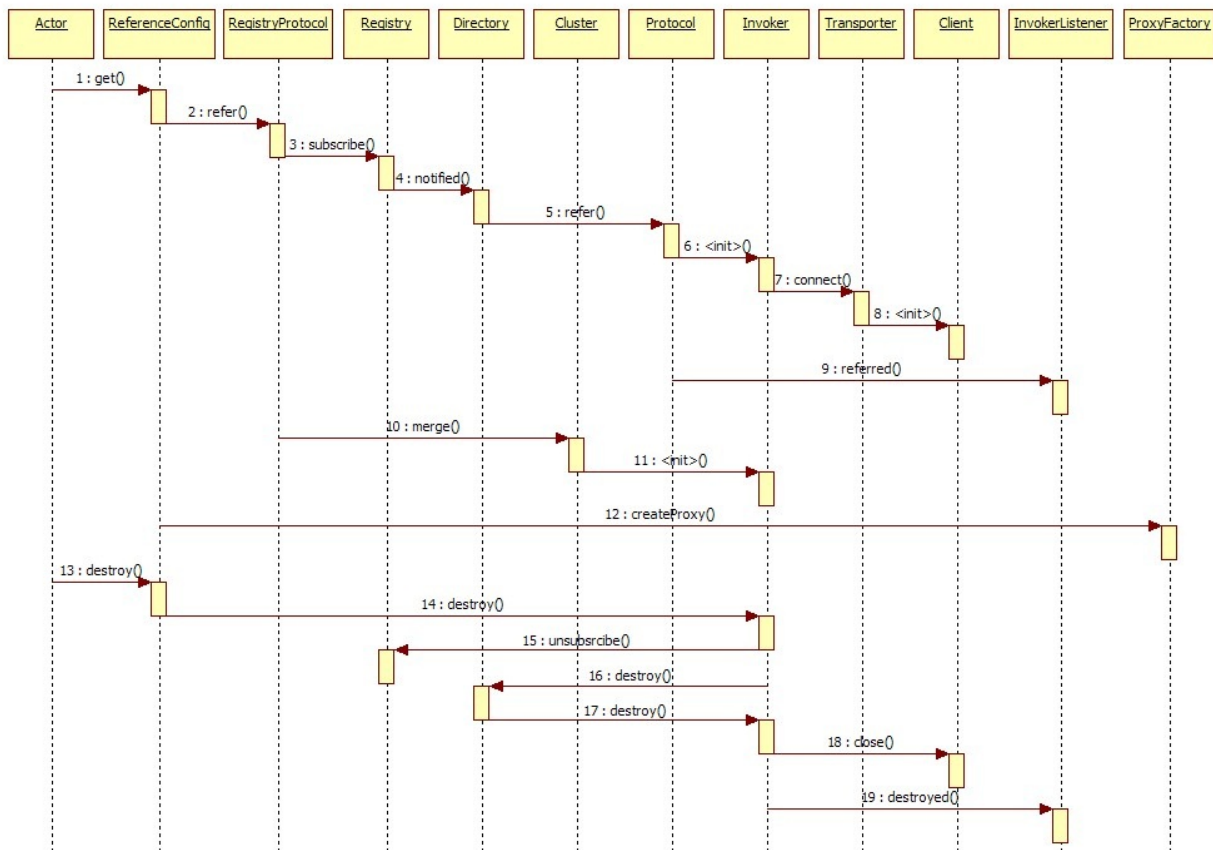
Expose service sequence

Expand the initialization chain of service provider exposes service which at the left of the overall design map, the sequence diagram shows below:



Reference service sequence

Expand the initialization chain of service consumer references service which at the right of the overall design map, the sequence diagram shows below:



Domain model

The core domain models of Dubbo:

- Protocol is service domain, it is the main function entrance of Invoker exposure and reference, it is responsible for the life cycle management of Invoker.
- Invoker is entity domain, it is the core model of Dubbo, all the other models are disturbed, or converted to it, it represents an executable, you can call it by calling invoke, it can be a local implementation, a remote implementation, or a cluster implementation.
- Invocation is session domain, it holds variables in the calling process, such as the method name, the parameters, and so on.

Base design principle

- Use Microkernel + Plugin design pattern , Microkernel only responsible for assembly Plugin, the functions of Dubbo are implemented by extension

points, it means that all functions of Dubbo can be replaced by self defined extension by user.

- Use URL to be the startdard format of config information, all extension points transfer config information by URL.

More design principles refer to: [Framework design principle](#)

SPI Loading

SPI Config

Source :

Dubbo SPI is inherited from standard JDK SPI(Service Provider Interface) and makes it more powerful.

Dubbo fixed below issues of the standard JDK SPI :

- the standard JDK SPI will load and instantize all implementation at once. It will be a waste of resources if the implementation is timecosted ,but never be used.
- We can't acquire the SPI name,if loading the SPI implementation is failed.For example : standard JDK ScriptEngine , get script type by invoking method getName(). RubyScriptEngine class will load failed if the depenency jar jruby.jar is missing , and the real error info will be lost. When user executes ruby scripts , program throws exception , telling not support ruby, but it is not the real cause.
- Enhance the SPI functionality by supporting IoC and AOP , one SPI can be easily injected by another SPI simply using setter.

Appointment :

In the jar file containing extension class ¹ , places a config file `META-INF/dubbo/full interface name` , file content pattern : `SPI name=the fully qualified name for the extension class` , use new line seperator for multiple implementation.

Example :

To extend Dubbo Protocol , placee a text file in the extension jar file : `META-INF/dubbo/com.alibaba.dubbo.rpc.Protocol` , content :


```
xxx=com.alibaba.xxx.XxxProtocol
```

content of the implementation ² :

```
package com.alibaba.xxx;

import com.alibaba.dubbo.rpc.Protocol;

public class XxxProtocol implements Protocol {
    // ...
}
```

Configuration in config module

In Dubbo config module , all SPI points have related attributes or labels , we can choose the specific SPI implementation by using its name. Like :

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

SPI Features

SPI Auto Wrap

Auto wrap the SPI's Wrapper class ° `ExtensionLoader` loads the SPI implementation , if the SPI has a copy instructor ,it will be regarded as the SPI's Wrapper class °

Wrapper class content :

```

package com.alibaba.xxx;

import com.alibaba.dubbo.rpc.Protocol;

public class XxxProtocolWrapper implements Protocol {
    Protocol impl;

    public XxxProtocol(Protocol protocol) { impl = protocol; }

    //after interface method is executed, the method in extension
    //will be executed
    public void refer() {
        //... some operation
        impl.refer();
        // ... some operation
    }

    // ...
}

```

Wrapper class also implements the same SPI interface, but Wrapper is not the real implementation. It is used to wrap the real implementation returned from the `ExtensionLoader`. The real returned instance by `ExtensionLoader` is the Wrapper class instance, Wrapper holds the real SPI implementation class.

There can be many Wrapper for one SPI, simply add one if you need.

By Wrapper class, you will be able to move some logics into Wrapper for all SPIs. Newly added Wrapper class adds external logics for all SPIs, looks like AOP, Wrapper acts as a proxy for SPI.

SPI Auto Load

When loading the SPI, Dubbo will auto load the dependency SPI. When one SPI implementation contains an attribute which is also an SPI of another type, `ExtensionLoader` will automatically load the dependency SPI. `ExtensionLoader` knows all the members of the specific SPI by scanning the setter method of all implementation class.

Demo : two SPI `CarMaker` (car maker) 、 `WheelMaker` (wheel maker)

Interfaces look like :

```
public interface CarMaker {  
    Car makeCar();  
}  
  
public interface WheelMaker {  
    Wheel makeWheel();  
}
```

`CarMaker` implementation :

```
public class RaceCarMaker implements CarMaker {  
    WheelMaker wheelMaker;  
  
    public setWheelMaker(WheelMaker wheelMaker) {  
        this.wheelMaker = wheelMaker;  
    }  
  
    public Car makeCar() {  
        // ...  
        Wheel wheel = wheelMaker.makeWheel();  
        // ...  
        return new RaceCar(wheel, ...);  
    }  
}
```

when `ExtensionLoader` loading `CarMaker` implementation `RaceCar` , `setWheelMaker` needs paramType `WheelMaker` which is also an SPI, It will be automatically loaded .

This brings a new question:How `ExtensionLoader` determines which implementation to use when load the injected SPI . As for this demo, when existing multi `WheelMaker` implementation, which one should the `ExtensionLoader` chooses.

Good question ,we will explain it in the following chapter: SPI Auto Adaptive.

SPI Auto Adaptive

The dependency SPI that `ExtensionLoader` injects is an instance of `Adaptive` , the real spi implementation is known until the adaptive instance is be executed.

Dubbo use URL (containing Key-Value) to pass the configuration.

The SPI method invocation has the URL parameter (Or Entity that has URL attribute)

In this way depended SPI can get configuration from URL , after config all SPI key needed, configuration information will be passed from outer by URL.URL act as a bus when pass the config information.

Demo : two SPI `CarMaker` 、 `WheelMaker`

interface looks like :

```
public interface CarMaker {  
    Car makeCar(URL url);  
}  
  
public interface WheelMaker {  
    Wheel makeWheel(URL url);  
}
```

`CarMaker` implementation :

```
public class RaceCarMaker implements CarMaker {
    WheelMaker wheelMaker;

    public setWheelMaker(WheelMaker wheelMaker) {
        this.wheelMaker = wheelMaker;
    }

    public Car makeCar(URL url) {
        // ...
        Wheel wheel = wheelMaker.makeWheel(url);
        // ...
        return new RaceCar(wheel, ...);
    }
}
```

when execute the code above

```
// ...
Wheel wheel = wheelMaker.makeWheel(url);
// ...
```

， the injected `Adaptive` object determine which `WheelMaker` 's `makeWheel` method will be executed by predefined Key. ◦ Such as `wheel.type` , key `url.get("wheel.type")` will determine `wheelMake` implementation. ◦ The logic of `Adaptive` instance of fixed , getting the predefined Key of the URL , dynamically creating the real implementation and execute it.

For Dubbo , the SPI `Adaptive` implementation in `ExtensionLoader` is dynamically created when dubbo is loading the SPI. ◦ Get the Key from URL, the Key will be provided through `@Adaptive` annotation for the interface method definition.

Below is Dubbo Transporter SPI codes :

```
public interface Transporter {  
    @Adaptive({"server", "transport"})  
    Server bind(URL url, ChannelHandler handler) throws Remoting  
    Exception;  
  
    @Adaptive({"client", "transport"})  
    Client connect(URL url, ChannelHandler handler) throws Remot  
    ingException;  
}
```

for the method bind(), Adaptive will firstly search `server` key , if no Key were founded then will search `transport` key , to determine the implementation that the proxy represent for.

SPI Auto Activation

As for Collections SPI, such as : `Filter` , `InvokerListener` , `ExportListener` , `TelnetHandler` , `StatusChecker` etc , multi implementations can be loaded at one time. User can simplify configuration by using auto activation , Like :

```
import com.alibaba.dubbo.common.extension.Activate;  
import com.alibaba.dubbo.rpc.Filter;  
  
@Activate // Active for any condition  
public class XxxFilter implements Filter {  
    // ...  
}
```

Or :

```
import com.alibaba.dubbo.common.extension.Activate;
import com.alibaba.dubbo.rpc.Filter;

@Activate("xxx") // when configed xxx parameter and the parameter has a valid value, the SPI is activated, for example configed cache="lru", auto activate CacheFilter.
public class XxxFilter implements Filter {
    // ...
}
```

Or :

```
import com.alibaba.dubbo.common.extension.Activate;
import com.alibaba.dubbo.rpc.Filter;

@Activate(group = "provider", value = "xxx") // only activate for provider, group can be "provider" or "consumer"
public class XxxFilter implements Filter {
    // ...
}
```

¹. Note : The config file here is in your own jar file, not in dubbo release jar file, Dubbo will scan all jar files for the same filename in ClassPath and merge together then ↩

². Note : SPI will be loaded in singleton pattern (Please ensure thread safety), cached in `ExtensionLoader` ↩

Implementation details

Initialization details

Service parsing

Based on `META-INF/spring.handlers` config in `dubbo.jar`, Spring calls `DubboNamespaceHandler` when meeting dubbo namespace.

All Dubbo tags are parsed by `DubboBeanDefinitionParser`, based on one to one attribute mapping, the XML label is parsed as a Bean object.

Transfer Bean object to URL, and transfer all attributes of Bean to URL parameters when `ServiceConfig.export()` or `ReferenceConfig.get()` initialization.

Then parse URL to [Protocol extension point](#), based on [Extension point adaptive mechanism](#) of extension point, processing service exposure or reference for different protocols according to URL protocol header.

Service Exposure

1. Only expose service port:

Direct exposing to provider when have not Registry, ¹, the URL format which parsing by `ServiceConfig`: `dubbo://service-host/com.foo.FooService?version=1.0.0`.

Based on extension point adaptive mechanism, call `export()` method of `DubboProtocol` and open server port by identifying `dubbo://` protocol header of URL.

2. Expose to Registry:

Expose provider address to Registry², the URL format which parsing by

```
ServiceConfig : registry://registry-  
host/com.alibaba.dubbo.registry.RegistryService?  
export=URL.encode("dubbo://service-host/com.foo.FooService?  
version=1.0.0") ,
```

Based on extension point adaptive mechanism, call `export()` method of `RegistryProtocol` by identifying `registry://` protocol header, register the provider URL parameter of `export` to Registry.

Resend to `Protocol` extension point to do exposure: `dubbo://service-host/com.foo.FooService?version=1.0.0`, then based on extension point adaptive mechanism, call `export()` method of `DubboProtocol` and open server port by identifying `dubbo://` protocol header of provider URL.

Service Reference

1. Direct connect service

Direct connect provider when have not Registry³, the URL format which parsing by `ReferenceConfig` : `dubbo://service-host/com.foo.FooService?version=1.0.0` .

Based on extension point adaptive mechanism, call `refer()` method of `DubboProtocol` by identifying `dubbo://` protocol header of URL, and return provider reference.

2. Service Registry discovery

Discover provider address by Registry⁴, the URL format which parsing by

```
ReferenceConfig : registry://registry-  
host/com.alibaba.dubbo.registry.RegistryService?  
refer=URL.encode("consumer://consumer-host/com.foo.FooService?  
version=1.0.0") .
```

Based on extension point adaptive mechanism, call `refer()` method of `RegistryProtocol` by identifying `registry://` protocol header of URL, then based on the condition of parameter of `refer` to search provider URL, for example: `dubbo://service-host/com.foo.FooService?version=1.0.0` .

Then based on extension point adaptive mechanism, call `refer()` method of `DubboProtocol` to get provider reference by identifying `dubbo://` protocol header of provider URL.

Then `RegistryProtocol` disguise various provider references to single provider by `Cluster` extension point and return.

Service Filter

Based on extension point adaptive mechanism, all `Protocol` extension points are auto wrapped `wrapper` class.

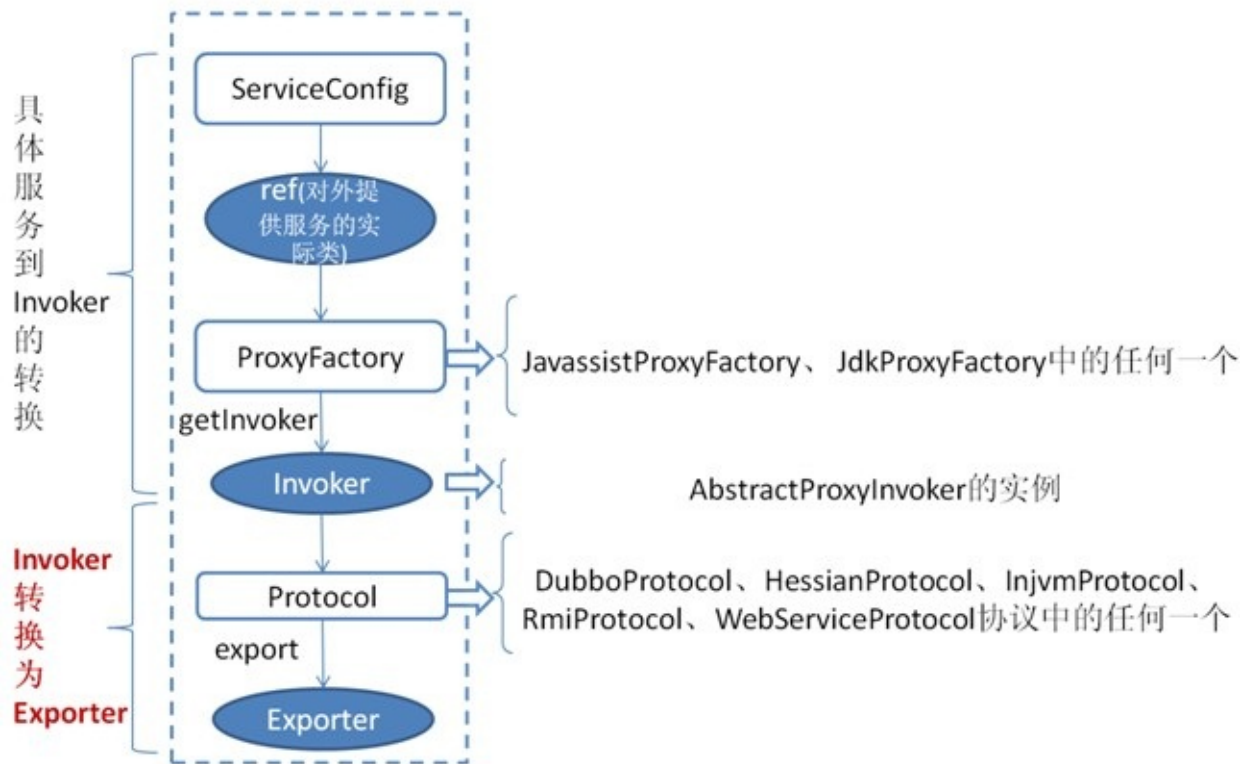
Based on `ProtocolFilterWrapper` class, make all `Filter` as chain, call the real reference at the end of the chain.

Based on `ProtocolListenerWrapper` class, make all `InvokerListener` and `ExporterListener` as list, perform call back before and after exposure and reference.

All additional functions would be implemented by `Filter`, including Monitor.

RPC details

The detail process of exposing service by service provider



The above image shows the main process of exposing service by service provider:

First `ServiceConfig` class get the actual class `ref` that provides service(e.g. 如: `HelloWorldImpl`), then generating a `AbstractProxyInvoker` instance by the `getInvoker` method of `ProxyFactory` class, to this step, complete the transformation of specific service to `Invoker`, next is the process of converting `Invoker` to `Exporter`.

The key of Dubbo processing service exposure is the process of converting `Invoker` to `Exporter`, the red part in the above image. Here we introduce the implementation of the two typical protocols, Dubbo and RMI:

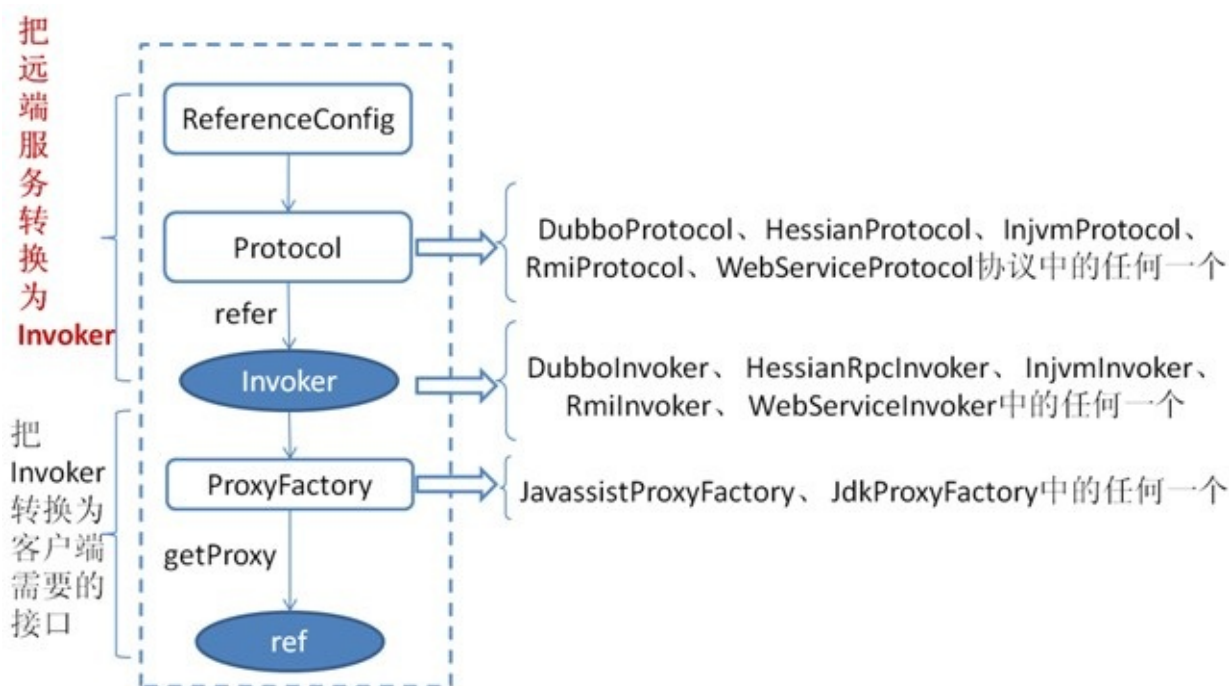
Dubbo implementation

The transformation from `Invoker` of Dubbo protocol to `Exporter` takes place in the `export` method of `DubboProtocol` class, it mainly opens the socket to listen service and receive all kinds of requests sent by the client, and the communication details are implemented by Dubbo itself.

RMI implementation

The transformation from `Invoker` of RMI protocol to `Exporter` takes place in the `export` method of `RmiProtocol` class, the RMI service is implemented by Spring, Dubbo or JDK, and the communication details are implemented by JDK, which saves a lot of work.

The detail process of serving service for service consumer



The above image is the main process of service consumption:

First, the `init` method of `ReferenceConfig` class calls the `refer` method of `Protocol` to generate `Invoker` instance (such as the red part in the above image), which is the key of service consumption. Then the `Invoker` is converted to the interface required by the client (such as: `HelloWorld`).

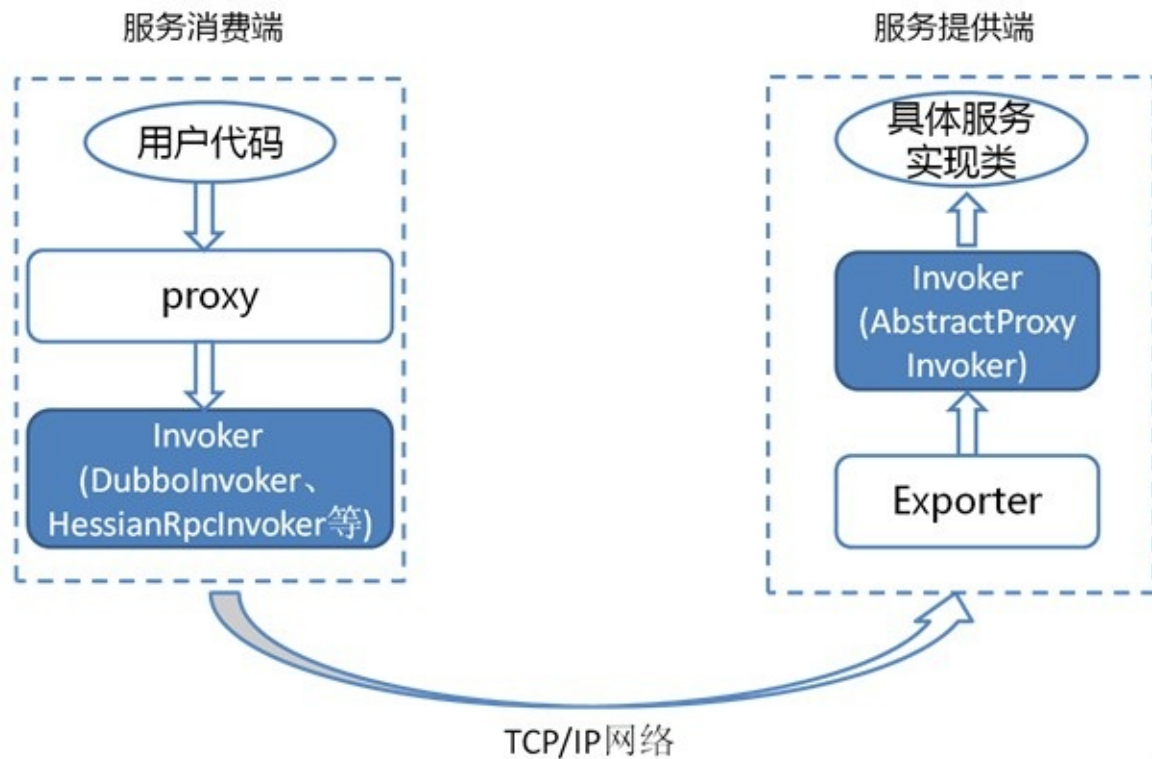
For each protocol such as RMI/Dubbo/Web service, the details they call `refer` method generate `Invoker` instance are similar to the previous section.

Invoker everywhere

Because of `Invoker` is a very important concept in the Dubbo domain model, many of the design ideas are close to it. This makes `Invoker` permeate the entire implementation code, and it's really easy to mix up for people who have just

started Dubbo.

Let's use a simple image below to describe the 2 important `Invoker` : service provider `Invoker` and service consumer `Invoker` :



To better explain the above image, we provide the below code examples of service consumption and providers:

Service consumer code:

```
public class DemoClientAction {

    private DemoService demoService;

    public void setDemoService(DemoService demoService) {
        this.demoService = demoService;
    }

    public void start() {
        String hello = demoService.sayHello("world" + i);
    }

}
```

The `DemoService` in above code is the proxy of service consumer in above image, user can call `Invoker`⁵ which implementate the real RPC by the proxy.

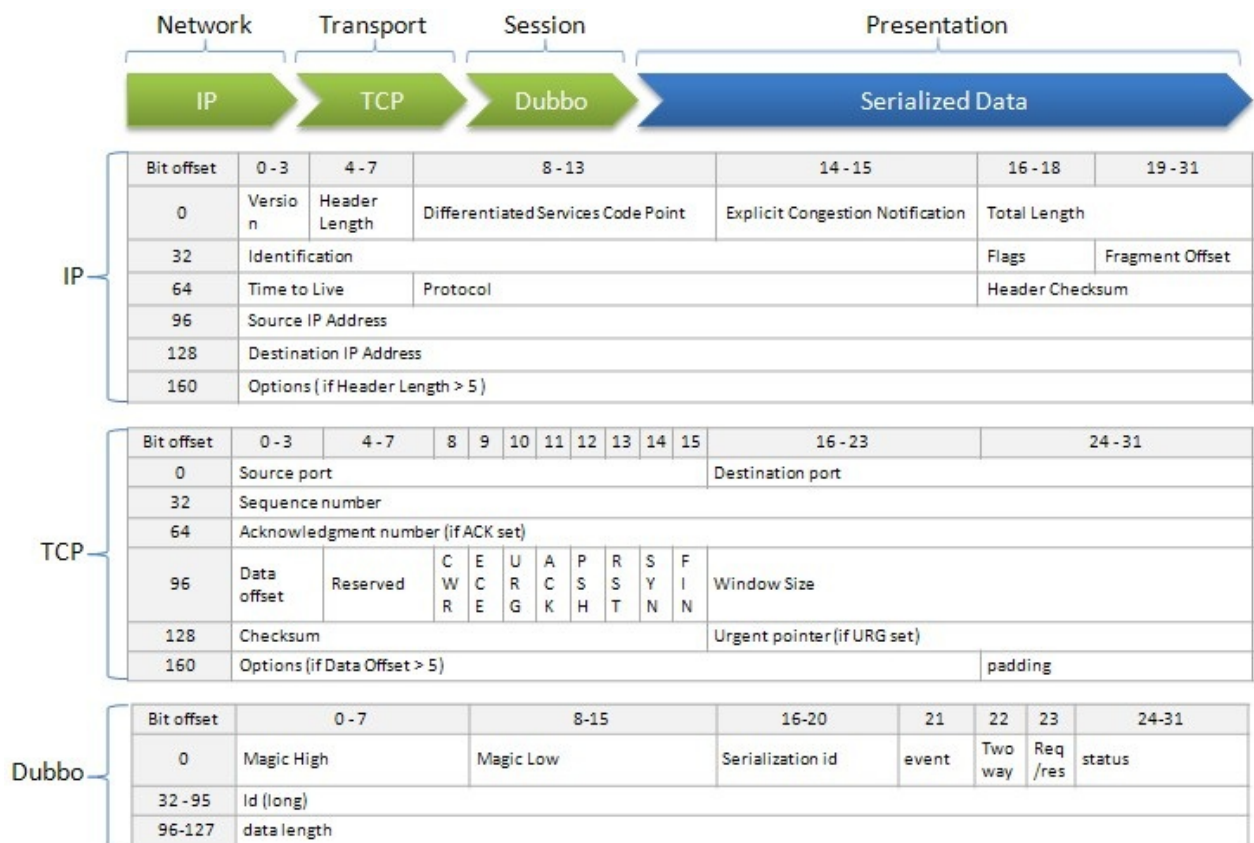
Service provider code:

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

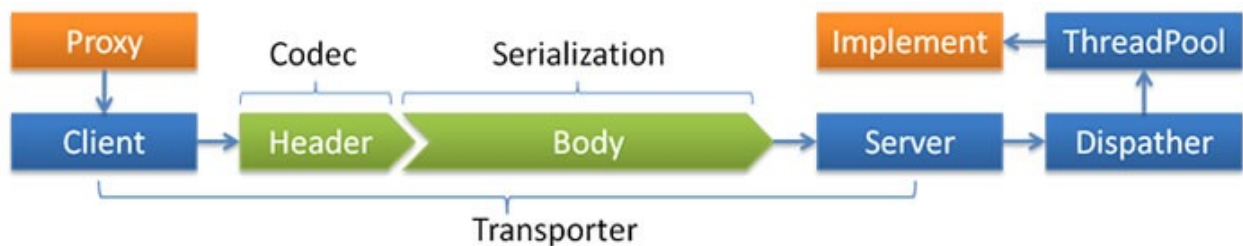
The above class would be encapsulated to be a `AbstractProxyInvoker` instance, and create a new `Exporter` instance, then find corresponding `Exporter` instance and call its corresponding `AbstractProxyInvoker` instance when network communication layer recieve request, so that real call service provider code. There are some other `Invoker` classes, but the above 2 are the most important.

Remote communication details

Protocol header agreement



Thread dispatch model



- Dispatcher: all, direct, message, execution, connection
- ThreadPool: fixed, cached

1. is `<dubbo:service registry="N/A" />` or `<dubbo:registry address="N/A" />` ↩

2. is `<dubbo:registry address="zookeeper://10.20.153.10:2181" />` ↩

3. is `<dubbo:reference url="dubbo://service-host/com.foo.FooService?version=1.0.0" />` ↩

4. is `<dubbo:registry address="zookeeper://10.20.153.10:2181" />` ↩

5

5. is one of `DubboInvoker` , `HessianRpcInvoker` , `InjvmInvoker` ,
`RmiInvoker` , `WebServiceInvoker` [↩](#)

SPI Extension Implementations

SPI extension interface is used for system integration, it's also useful for dubbo contributor to extend dubbo functionality.

Protocol Extension

Summary

Extension to RPC protocol, hide details of remote call.

Contract:

- When user calls `invoke()` method of `Invoker` object which's returned from `refer()` call, the protocol needs to correspondingly execute `invoke()` method of `Invoker` object passed from remote `export()` method associated with the same URL.
- Moreover, it's protocol's responsibility to implement `Invoker` which's returned from `refer()`. Generally speaking, protocol sends remote request in the `Invoker` implementation, but needs not to care about the `Invoker` passed into `export()` since the framework will implement the logic and pass in the instance.

Notes:

- Protocol does not need to care about the proxy of the business interface. The upper layer of the framework will convert `Invoker` into business interface.
- It is not a requirement that the protocol must use TCP for network communication. It could be file-sharing, IPC, or others.

Extension Interface

- `com.alibaba.dubbo.rpc.Protocol`
- `com.alibaba.dubbo.rpc.Exporter`
- `com.alibaba.dubbo.rpc.Invoker`

```
public interface Protocol {  
    /**  
     * Export remote service: <br>  
     * 1. Should save address info for the request when the protocol receives it: RpcContext.getContext().setRemoteAddress();<br>
```

```

>
    * 2. export() must be implemented as idempotent, that is, i
t should not introduce side effect when the implementation gets
called with the same Invoker for more than once.
    * 3. Invoker is passed by the framework, and the protocol s
hould not care about it. <br>
    *
    * @param <T> Service type
    * @param invoker Service invoker
    * @return exporter The reference of service exporter, used
for cancelling service export.
    * @throws RpcException throw when there's any error during
service export, e.g. the port is occupied
    */
    <T> Exporter<T> export(Invoker<T> invoker) throws RpcExcepti
on;

    /**
    * Reference remote service: <br>
    * 1. When user calls `invoke()` method of `Invoker` object
which's returned from `refer()` call, the protocol needs to corr
espondingly execute `invoke()` method of `Invoker` object passed
from remote `export()` method associated with the same URL. <br>
>
    * 2. It's protocol's responsibility to implement `Invoker`
which's returned from `refer()`. Generally speaking, protocol se
nds remote request in the `Invoker` implementation. <br>
    * 3. When there's check=false set in URL, the implementatio
n must not throw exception but try to recover when connection fa
ils.
    *
    * @param <T> Service type
    * @param type Service type
    * @param url URL address for the remote service
    * @return invoker service's local proxy
    * @throws RpcException throw when there's any error while c
onnecting to the service provider
    */
    <T> Invoker<T> refer(Class<T> type, URL url) throws RpcExcep
tion;

```

```
}
```

Extension Configuration

```
<!-- declare protocol, if id is not set, then use the value of name for id -->
<dubbo:protocol id="xxx1" name="xxx" />
<!-- reference protocol, if protocol's attribute is not set, the
n protocol configuration will be scanned automatically from Application
Context -->
<dubbo:service protocol="xxx1" />
<!-- default value for referenced protocol, it will be used if protocol
attribute is not configured in <dubbo:service> -->
<dubbo:provider protocol="xxx1" />
```

Existing Protocol

- `com.alibaba.dubbo.rpc.injvm.InjvmProtocol`
- `com.alibaba.dubbo.rpc.dubbo.DubboProtocol`
- `com.alibaba.dubbo.rpc.rmi.RmiProtocol`
- `com.alibaba.dubbo.rpc.http.HttpProtocol`
- `com.alibaba.dubbo.rpc.http.hessian.HessianProtocol`

Extension Guide

Directory layout:

```
src
|-main
  |-java
    |-com
      |-xxx
        |-XxxProtocol.java (Protocol implementation)
        |-XxxExporter.java (Exporter implementation)
        |-XxxInvoker.java (Invoker implementation)
  |-resources
    |-META-INF
      |-dubbo
        |-com.alibaba.dubbo.rpc.Protocol (plain text file with the content: xxx=com.xxx.XxxProtocol)
```

XxxProtocol.java :

```
package com.xxx;

import com.alibaba.dubbo.rpc.Protocol;

public class XxxProtocol implements Protocol {
    public <T> Exporter<T> export(Invoker<T> invoker) throws RpcException {
        return new XxxExporter(invoker);
    }
    public <T> Invoker<T> refer(Class<T> type, URL url) throws RpcException {
        return new XxxInvoker(type, url);
    }
}
```

XxxExporter.java :

```
package com.xxx;

import com.alibaba.dubbo.rpc.support.AbstractExporter;

public class XxxExporter<T> extends AbstractExporter<T> {
    public XxxExporter(Invoker<T> invoker) throws RemotingException{
        super(invoker);
        // ...
    }
    public void unexport() {
        super.unexport();
        // ...
    }
}
```

XxxInvoker.java :

```
package com.xxx;

import com.alibaba.dubbo.rpc.support.AbstractInvoker;

public class XxxInvoker<T> extends AbstractInvoker<T> {
    public XxxInvoker(Class<T> type, URL url) throws RemotingException{
        super(type, url);
    }
    protected abstract Object doInvoke(Invocation invocation) throws Throwable {
        // ...
    }
}
```

META-INF/dubbo/com.alibaba.dubbo.rpc.Protocol :

```
xxx=com.xxx.XxxProtocol
```


Filter Extension

Summary

Extension for intercepting the invocation for both service provider and consumer, furthermore, most of functions in dubbo are implemented base on the same mechanism. Since every time when remote method is invoked, the filter extensions will be executed too, the corresponding penalty should be considered before more filters are added.

Contract:

- User defined filters are executed after built-in filters by default.
- Special value `default` is introduced to represent the default extension location. For example: for `filter="xxx,default,yyy"` , `xxx` is before default filter, and `yyy` is after the default filter.
- Special value `-` means delete. For example: `filter="-foo1"` excludes `foo1` extension. For example, `filter="-default"` excludes all default filters.
- When provider and service have filter configured at the same moment, all filters are accumulated together instead of override, for example: for `<dubbo:provider filter="xxx,yyy"/>` and `<dubbo:service filter="aaa,bbb" />` , `xxx` , `yyy` , `aaa` , `bbb` are all count as filters. In order to change to override, use: `<dubbo:service filter="-xxx, -yyy,aaa,bbb" />`

Extension Interface

```
com.alibaba.dubbo.rpc.Filter
```

Extension Configuration


```
<!-- filter for consumer -->
<dubbo:reference filter="xxx,yyy" />
<!-- default filter configuration for the consumer, will interce
pt for all references -->
<dubbo:consumer filter="xxx,yyy"/>
<!-- filter for provider -->
<dubbo:service filter="xxx,yyy" />
<!-- default filter configuration for the provider, will interce
pt for all services -->
<dubbo:provider filter="xxx,yyy"/>
```

Existing Extension

- `com.alibaba.dubbo.rpc.filter.EchoFilter`
- `com.alibaba.dubbo.rpc.filter.GenericFilter`
- `com.alibaba.dubbo.rpc.filter.GenericImplFilter`
- `com.alibaba.dubbo.rpc.filter.TokenFilter`
- `com.alibaba.dubbo.rpc.filter.AccessLogFilter`
- `com.alibaba.dubbo.rpc.filter.CountFilter`
- `com.alibaba.dubbo.rpc.filter.ActiveLimitFilter`
- `com.alibaba.dubbo.rpc.filter.ClassLoaderFilter`
- `com.alibaba.dubbo.rpc.filter.ContextFilter`
- `com.alibaba.dubbo.rpc.filter.ConsumerContextFilter`
- `com.alibaba.dubbo.rpc.filter.ExceptionFilter`
- `com.alibaba.dubbo.rpc.filter.ExecuteLimitFilter`
- `com.alibaba.dubbo.rpc.filter.DeprecatedFilter`

Extension Guide

Directory layout:

```
src
|-main
  |-java
    |-com
      |-xxx
        |-XxxFilter.java (Filter implementation)
  |-resources
    |-META-INF
      |-dubbo
        |-com.alibaba.dubbo.rpc.Filter (plain text file
with the content: xxx=com.xxx.XxxFilter)
```

XxxFilter.java :

```
package com.xxx;

import com.alibaba.dubbo.rpc.Filter;
import com.alibaba.dubbo.rpc.Invoker;
import com.alibaba.dubbo.rpc.Invocation;
import com.alibaba.dubbo.rpc.Result;
import com.alibaba.dubbo.rpc.RpcException;

public class XxxFilter implements Filter {
    public Result invoke(Invoker<?> invoker, Invocation invocation) throws RpcException {
        // before filter ...
        Result result = invoker.invoke(invocation);
        // after filter ...
        return result;
    }
}
```

META-INF/dubbo/com.alibaba.dubbo.rpc.Filter :

```
xxx=com.xxx.XxxFilter
```


InvokerListener Extension

Summary

Fire event when there's any service referenced.

Extension Interface

```
com.alibaba.dubbo.rpc.InvokerListener
```

Extension Configuration

```
<!-- 引用服务监听 -->
<!-- service reference listener -->
<dubbo:reference listener="xxx,yyy" />
<!-- default service reference listener -->
<dubbo:consumer listener="xxx,yyy" />
```

Existing Extension

```
com.alibaba.dubbo.rpc.listener.DeprecatedInvokerListener
```

Extension Guide

Directory layout:

```
src
|-main
  |-java
    |-com
      |-xxx
        |-XxxInvokerListener.java (InvokerListener implementation)
      |-resources
        |-META-INF
          |-dubbo
            |-com.alibaba.dubbo.rpc.InvokerListener (plain text file with the content: xxx=com.xxx.XxxInvokerListener)
```

XxxInvokerListener.java :

```
package com.xxx;

import com.alibaba.dubbo.rpc.InvokerListener;
import com.alibaba.dubbo.rpc.Invoker;
import com.alibaba.dubbo.rpc.RpcException;

public class XxxInvokerListener implements InvokerListener {
    public void referred(Invoker<?> invoker) throws RpcException {
        // ...
    }
    public void destroyed(Invoker<?> invoker) throws RpcException {
        // ...
    }
}
```

META-INF/dubbo/com.alibaba.dubbo.rpc.InvokerListener :

```
xxx=com.xxx.XxxInvokerListener
```


ExporterListener Extension

Summary

Fire events when there's any service exported.

Extension Interface

```
com.alibaba.dubbo.rpc.ExporterListener
```

Extension Configuration

```
<!-- service exporter listener -->  
<dubbo:service listener="xxx,yyy" />  
<!-- default exporter listener for service provider -->  
<dubbo:provider listener="xxx,yyy" />
```

Existing Extension

```
com.alibaba.dubbo.registry.directory.RegistryExporterListener
```

Extension Guide

Directory layout:

```
src
|-main
  |-java
    |-com
      |-xxx
        |-XxxExporterListener.java (ExporterListener implementation)
      |-resources
        |-META-INF
          |-dubbo
            |-com.alibaba.dubbo.rpc.ExporterListener (plain text file with the content: xxx=com.xxx.XxxExporterListener)
```

XxxExporterListener.java :

```
package com.xxx;

import com.alibaba.dubbo.rpc.ExporterListener;
import com.alibaba.dubbo.rpc.Exporter;
import com.alibaba.dubbo.rpc.RpcException;

public class XxxExporterListener implements ExporterListener {
    public void exported(Exporter<?> exporter) throws RpcException {
        // ...
    }
    public void unexported(Exporter<?> exporter) throws RpcException {
        // ...
    }
}
```

META-INF/dubbo/com.alibaba.dubbo.rpc.ExporterListener :

```
xxx=com.xxx.XxxExporterListener
```


Cluster Extension

Summary

Group service providers in a cluster, and treat them as one single provider.

Extension Interface

```
com.alibaba.dubbo.rpc.cluster.Cluster
```

Extension Configuration

```
<dubbo:protocol cluster="xxx" />
<!-- default configuration, will take affect if cluster attribute is not configured in <dubbo:protocol> -->
<dubbo:provider cluster="xxx" />
```

Existing Extensions

- `com.alibaba.dubbo.rpc.cluster.support.FailoverCluster`
- `com.alibaba.dubbo.rpc.cluster.support.FailfastCluster`
- `com.alibaba.dubbo.rpc.cluster.support.FailsafeCluster`
- `com.alibaba.dubbo.rpc.cluster.support.FailbackCluster`
- `com.alibaba.dubbo.rpc.cluster.support.ForkingCluster`
- `com.alibaba.dubbo.rpc.cluster.support.AvailableCluster`

Extension Guide

Directory layout:

```
src
|-main
  |-java
    |-com
      |-xxx
        |-XxxCluster.java (Cluster implementation)
  |-resources
    |-META-INF
      |-dubbo
        |-com.alibaba.dubbo.rpc.cluster.Cluster (plain text
        ext file with the content: xxx=com.xxx.XxxCluster)
```

XxxCluster.java :

```
package com.xxx;

import com.alibaba.dubbo.rpc.cluster.Cluster;
import com.alibaba.dubbo.rpc.cluster.support.AbstractClusterInvoker;
import com.alibaba.dubbo.rpc.cluster.Directory;
import com.alibaba.dubbo.rpc.cluster.LoadBalance;
import com.alibaba.dubbo.rpc.Invoker;
import com.alibaba.dubbo.rpc.Invocation;
import com.alibaba.dubbo.rpc.Result;
import com.alibaba.dubbo.rpc.RpcException;

public class XxxCluster implements Cluster {
    public <T> Invoker<T> merge(Directory<T> directory) throws RpcException {
        return new AbstractClusterInvoker<T>(directory) {
            public Result doInvoke(Invocation invocation, List<Invoker<T>>
invokers, LoadBalance loadbalance) throws RpcException
n {
                // ...
            }
        };
    }
}
```

META-INF/dubbo/com.alibaba.dubbo.rpc.cluster.Cluster :

```
xxx=com.xxx.XxxCluster
```

Router Extension

Summary

Pick one from service providers and fire the invocation.

Extension Interface

- `com.alibaba.dubbo.rpc.cluster.RouterFactory`
- `com.alibaba.dubbo.rpc.cluster.Router`

Existing Extension

- `com.alibaba.dubbo.rpc.cluster.router.ScriptRouterFactory`
- `com.alibaba.dubbo.rpc.cluster.router.FileRouterFactory`

Extension Guide

Directory layout:

```
src
|-main
|   |-java
|       |-com
|           |-xxx
|               |-XxxRouterFactory.java (LoadBalance implementat
ion)
|   |-resources
|       |-META-INF
|           |-dubbo
|               |-com.alibaba.dubbo.rpc.cluster.RouterFactory (p
lain text file with the content: xxx=com.xxx.XxxRouterFactory)
```

XxxRouterFactory.java :

```
package com.xxx;

import com.alibaba.dubbo.rpc.cluster.RouterFactory;
import com.alibaba.dubbo.rpc.Invoker;
import com.alibaba.dubbo.rpc.Invocation;
import com.alibaba.dubbo.rpc.RpcException;

public class XxxRouterFactory implements RouterFactory {
    public <T> List<Invoker<T>> select(List<Invoker<T>> invokers
, Invocation invocation) throws RpcException {
        // ...
    }
}
```

META-INF/dubbo/com.alibaba.dubbo.rpc.cluster.RouterFactory :

```
xxx=com.xxx.XxxRouterFactory
```

LoadBalance Extension

Summary

Pick one from service providers and fire the invocation.

Extension Interface

```
com.alibaba.dubbo.rpc.cluster.LoadBalance
```

Extension Configuration

```
<dubbo:protocol loadbalance="xxx" />
<!-- default configuration, will take effect when loadbalance is
not configured in <dubbo:protocol> -->
<dubbo:provider loadbalance="xxx" />
```

Existing Extension

- `com.alibaba.dubbo.rpc.cluster.loadbalance.RandomLoadBalance`
- `com.alibaba.dubbo.rpc.cluster.loadbalance.RoundRobinLoadBalance`
- `com.alibaba.dubbo.rpc.cluster.loadbalance.LeastActiveLoadBalance`

Extension Guide

Directory layout:

```
src
|-main
  |-java
    |-com
      |-xxx
        |-XxxLoadBalance.java (LoadBalance implementation)
  |-resources
    |-META-INF
      |-dubbo
        |-com.alibaba.dubbo.rpc.cluster.LoadBalance (plain text file with the content: xxx=com.xxx.XxxLoadBalance)
```

XxxLoadBalance.java :

```
package com.xxx;

import com.alibaba.dubbo.rpc.cluster.LoadBalance;
import com.alibaba.dubbo.rpc.Invoker;
import com.alibaba.dubbo.rpc.Invocation;
import com.alibaba.dubbo.rpc.RpcException;

public class XxxLoadBalance implements LoadBalance {
    public <T> Invoker<T> select(List<Invoker<T>> invokers, Invocation invocation) throws RpcException {
        // ...
    }
}
```

META-INF/dubbo/com.alibaba.dubbo.rpc.cluster.LoadBalance :

```
xxx=com.xxx.XxxLoadBalance
```


Merger Extension

Summary

Merge strategy for return result aggregation in group.

Extension Interface

```
com.alibaba.dubbo.rpc.cluster.Merger
```

Extension Configuration

```
<dubbo:method merger="xxx" />
```

Existing Extension

- `com.alibaba.dubbo.rpc.cluster.merger.ArrayMerger`
- `com.alibaba.dubbo.rpc.cluster.merger.ListMerger`
- `com.alibaba.dubbo.rpc.cluster.merger.SetMerger`
- `com.alibaba.dubbo.rpc.cluster.merger.MapMerger`

Extension Guide

Directory layout:

```
src
|-main
  |-java
    |-com
      |-xxx
        |-XxxMerger.java (Merger implementation)
  |-resources
    |-META-INF
      |-dubbo
        |-com.alibaba.dubbo.rpc.cluster.Merger (plain text file with the content: xxx=com.xxx.XxxMerger)
```

XxxMerger.java :

```
package com.xxx;

import com.alibaba.dubbo.rpc.cluster.Merger;

public class XxxMerger<T> implements Merger<T> {
    public T merge(T... results) {
        // ...
    }
}
```

META-INF/dubbo/com.alibaba.dubbo.rpc.cluster.Merger :

```
xxx=com.xxx.XxxMerger
```

Registry Extension

Summary

Registry extension is used for service registration and discovery.

Extension Interface

- `com.alibaba.dubbo.registry.RegistryFactory`
- `com.alibaba.dubbo.registry.Registry`

Extension Configuration

```
<!-- config registry server -->
<dubbo:registry id="xxx1" address="xxx://ip:port" />
<!-- reference registry server, if registry attribute is not specified, then ApplicationContext will be scanned to find if there's any -->
<dubbo:service registry="xxx1" />
<!-- default configuration for referencing registry server, it will take effect if there's no registry attribute specified in <dubbo:service> -->
<dubbo:provider registry="xxx1" />
```

Extension Contract

RegistryFactory.java :

```
public interface RegistryFactory {  
    /**  
     * Connect to registry server  
     *  
     * The contract for connecting to registry server: <br>  
     * 1. Will not check connection when check=false is set, otherwise exception will be thrown if connection fails. <br>  
     * 2. Support authorizing against username:password in the URL <br>  
     * 3. Support registry server backup with backup=10.20.153.10 <br>  
     * 4. Support cache on local disk with file=registry.cache <br>  
     * 5. Support timeout setup with timeout=1000 <br>  
     * 6. Support session expiration setup with session=60000 <br>  
     *  
     * @param url registry server address, null is not allowed  
     * @return reference to registry server, never return null  
     */  
    Registry getRegistry(URL url);  
}
```

RegistryService.java :

```
public interface RegistryService { // Registry extends RegistryService  
    /**  
     * Register service.  
     *  
     * Contract for registering service: <br>  
     * 1. Registration failure will be ignored and kept retrying if check=false is set in URL, otherwise exception will be thrown <br>  
     * 2. Persistence is required if dynamic=false is set in URL, otherwise, the registration info will be removed automatically when register quits accidentally <br>  
     * 3. Persistent by category if category=overrides is set in URL, default category is providers. It is possible to notify by
```

```

category. <br>
    * 4. Data lost is not tolerant when registry server reboots
    or network jitter happens. <br>
    * 5. It is not allowed to override each other when URLs hav
e same URI part but different parameters <br>
    *
    * @param url registration info, null is not allowed, e.g.:
    dubbo://10.20.153.10/com.alibaba.foo.BarService?version=1.0.0&ap
    plication=kylin
    */
    void register(URL url);

/**
    * Unregister service.
    *
    * Contract for unregistering service: <br>
    * 1. IllegalStateException should be thrown when registrati
    on info which's supposed to be persistent (with dynamic=false se
    t) cannot be found, otherwise it should be ignored. <br>
    * 2. To cancel one service, extract match on its URL will b
    e honored <br>
    *
    * @param url registration info, null is not allowed, e.g.:
    dubbo://10.20.153.10/com.alibaba.foo.BarService?version=1.0.0&ap
    plication=kylin
    */
    void unregister(URL url);

/**
    * 订阅服务.
    * Subscribe service.
    *
    * Contract for subscribing service: <br>
    * 1. Subscription failure will be ignored and kept retrying
    if check=false is set in URL <br>
    * 2. Only the specified category will be notified if catego
    ry=overrides is set in URL. Categories are seperated with comma,
    and all categorized data will be subscribed when wildcard "*" i
    s specified. <br>
    * 3. Allow to query by interface, group, version, classifie

```

```

r, e.g.: interface=com.alibaba.foo.BarService&version=1.0.0<br>
    * 4. Allow to query with wildcard "*" to subscribe all versions under all categories for all interfaces, e.g.: interface=*&group=*&version=*&classifier=*<br>
    * 5. Subscription will be automatically recovered when registry server reboots or network jitter happens. <br>
    * 6. It is not allowed to override each other when URLs have same URI part but different parameters <br>
    * 7. Subscription procedure will not return until the first notification happens. <br>
    *
    * @param url URL for subscription, null isn't allowed, e.g.: consumer://10.20.153.10/com.alibaba.foo.BarService?version=1.0.0&application=kylin
    * @param listener notification listener, null is not allowed
    */
    void subscribe(URL url, NotifyListener listener);

    /**
     * Unsubscribe service.
     *
     * Contract for unsubscribing service: <br>
     * 1. Simply ignore if not subscribe <br>
     * 2. Unsubscribe with URL exact match <br>
     *
     * @param url URL for unsubscription, null is not allowed, e.g.: consumer://10.20.153.10/com.alibaba.foo.BarService?version=1.0.0&application=kylin
     * @param listener notification listener, null is not allowed
     */
    void unsubscribe(URL url, NotifyListener listener);

    /**
     * 查询注册列表，与订阅的推模式相对应，这里为拉模式，只返回一次结果。
     * Lookup subscription list. Compared to push mode for subscription, this is pull mode and returns result only once.
     *
     * @see com.alibaba.dubbo.registry.NotifyListener#notify(Lis

```

```
t)
    * @param url URL for query, null is not allowed, e.g.: consumer://10.20.153.10/com.alibaba.foo.BarService?version=1.0.0&application=kylin
    * @return subscription list, could be null, has the same meaning as the parameters in {@link com.alibaba.dubbo.registry.NotifyListener#notify(List<URL>)}.
    */
    List<URL> lookup(URL url);

}
```

NotifyListener.java :

```

public interface NotifyListener {
    /**
     * Fire event when receive service change notification.
     *
     * Contract for notify: <br>
     * 1. Always notify with the whole data instead of partial data from the perspective of service interface and data type. In this way, user needs not compare with the previous result. <br>
     * 2. First notification for subscription must contain the full set of data for one particular service <br>
     * 3. It is allowed to separate the different type of data in the upcoming notifications, e.g.: it is legal to only notify one of types among providers, consumers, routes or overrides each time, but pls. note for this particular type, the data must be a full set. <br>
     * 4. If the data for one particular type is empty, need to notify with a special URL which has empty as its protocol and has category parameter for this particular type.
     * 5. Notifier (usually it is monitor center) needs to guarantee the notification sequence by, for say: single thread push, queuing in order, versioning, etc. <br>
     *
     * @param urls subscription list, always not empty, equivalent to the return result of {@link com.alibaba.dubbo.registry.RegistryService#lookup(URL)}.
     */
    void notify(List<URL> urls);
}

```

Existing Extension

```
com.alibaba.dubbo.registry.support.dubbo.DubboRegistryFactory
```

Extension Guide

Directory structure:


```
src
|-main
  |-java
    |-com
      |-xxx
        |-XxxRegistryFactoryjava (RegistryFactory implementation)
        |-XxxRegistry.java (Registry implementation)
  |-resources
    |-META-INF
      |-dubbo
        |-com.alibaba.dubbo.registry.RegistryFactory (plain text file with the content: xxx=com.xxx.XxxRegistryFactory)
```

XxxRegistryFactory.java :

```
package com.xxx;

import com.alibaba.dubbo.registry.RegistryFactory;
import com.alibaba.dubbo.registry.Registry;
import com.alibaba.dubbo.common.URL;

public class XxxRegistryFactory implements RegistryFactory {
    public Registry getRegistry(URL url) {
        return new XxxRegistry(url);
    }
}
```

XxxRegistry.java :

```
package com.xxx;

import com.alibaba.dubbo.registry.Registry;
import com.alibaba.dubbo.registry.NotifyListener;
import com.alibaba.dubbo.common.URL;

public class XxxRegistry implements Registry {
    public void register(URL url) {
        // ...
    }
    public void unregister(URL url) {
        // ...
    }
    public void subscribe(URL url, NotifyListener listener) {
        // ...
    }
    public void unsubscribe(URL url, NotifyListener listener) {
        // ...
    }
}
```

META-INF/dubbo/com.alibaba.dubbo.registry.RegistryFactory :

```
xxx=com.xxx.XxxRegistryFactory
```

Monitor Extension

Summary

Extension to monitor service invocation times and time taken for each service invocation.

Extension Interface

- `com.alibaba.dubbo.monitor.MonitorFactory`
- `com.alibaba.dubbo.monitor.Monitor`

Extension Configuration

```
<!-- configure monitor center -->  
<dubbo:monitor address="xxx://ip:port" />
```

Existing Extension

`com.alibaba.dubbo.monitor.support.dubbo.DubboMonitorFactory`

Extension Guide

Directory layout:

```
src
|-main
  |-java
    |-com
      |-xxx
        |-XxxMonitorFactoryjava (MonitorFactory implemen
tation)
          |-XxxMonitor.java (Monitor implementation)
    |-resources
      |-META-INF
        |-dubbo
          |-com.alibaba.dubbo.monitor.MonitorFactory (plai
n text file with the format: xxx=com.xxx.XxxMonitorFactory)
```

XxxMonitorFactory.java :

```
package com.xxx;

import com.alibaba.dubbo.monitor.MonitorFactory;
import com.alibaba.dubbo.monitor.Monitor;
import com.alibaba.dubbo.common.URL;

public class XxxMonitorFactory implements MonitorFactory {
    public Monitor getMonitor(URL url) {
        return new XxxMonitor(url);
    }
}
```

XxxMonitor.java :

```
package com.xxx;

import com.alibaba.dubbo.monitor.Monitor;

public class XxxMonitor implements Monitor {
    public void count(URL statistics) {
        // ...
    }
}
```

META-INF/dubbo/com.alibaba.dubbo.monitor.MonitorFactory :

```
xxx=com.xxx.XxxMonitorFactory
```

ExtensionFactory Extension

Summary

Factory to load dubbo extensions.

Extension Interface

```
com.alibaba.dubbo.common.extension.ExtensionFactory
```

Extension Configuration

```
<dubbo:application compiler="jdk" />
```

Existing Extension

- `com.alibaba.dubbo.common.extension.factory.SpiExtensionFactory`
- `com.alibaba.dubbo.config.spring.extension.SpringExtensionFactory`

Extension Guide

Directory layout:

```
src
|-main
  |-java
    |-com
      |-xxx
        |-XxxExtensionFactory.java (ExtensionFactory implementation)
      |-resources
        |-META-INF
          |-dubbo
            |-com.alibaba.dubbo.common.extension.ExtensionFactory (plain text file with the content: xxx=com.xxx.XxxExtensionFactory)
```

XxxExtensionFactory.java :

```
package com.xxx;

import com.alibaba.dubbo.common.extension.ExtensionFactory;

public class XxxExtensionFactory implements ExtensionFactory {
    public Object getExtension(Class<?> type, String name) {
        // ...
    }
}
```

META-INF/dubbo/com.alibaba.dubbo.common.extension.ExtensionFactory :

```
xxx=com.xxx.XxxExtensionFactory
```

ProxyFactory Extension

Summary

Convert `Invoker` into business interface.

Extension Interface

```
com.alibaba.dubbo.rpc.ProxyFactory
```

Extension Configuration

```
<dubbo:protocol proxy="xxx" />
<!-- default configuration, it will take effect when proxy attribute is not configured in <dubbo:protocol> -->
<dubbo:provider proxy="xxx" />
```

Existing Extension

- `com.alibaba.dubbo.rpc.proxy.JdkProxyFactory`
- `com.alibaba.dubbo.rpc.proxy.JavassistProxyFactory`

Extension Guide

Directory layout:


```
src
|-main
  |-java
    |-com
      |-xxx
        |-XxxProxyFactory.java (ProxyFactory implementat
ion)
      |-resources
        |-META-INF
          |-dubbo
            |-com.alibaba.dubbo.rpc.ProxyFactory (plain text
file with the content: xxx=com.xxx.XxxProxyFactory)
```

XxxProxyFactory.java :

```
package com.xxx;

import com.alibaba.dubbo.rpc.ProxyFactory;
import com.alibaba.dubbo.rpc.Invoker;
import com.alibaba.dubbo.rpc.RpcException;

public class XxxProxyFactory implements ProxyFactory {
    public <T> T getProxy(Invoker<T> invoker) throws RpcExceptio
n {
        // ...
    }
    public <T> Invoker<T> getInvoker(T proxy, Class<T> type, URL
url) throws RpcException {
        // ...
    }
}
```

META-INF/dubbo/com.alibaba.dubbo.rpc.ProxyFactory :

```
xxx=com.xxx.XxxProxyFactory
```


Compiler Extension

Summary

Java compiler, used for byte code dynamic generation for RPC invocation.

Extension Interface

```
com.alibaba.dubbo.common.compiler.Compiler
```

Extension Configuration

No configuration required, the extension will be automatically discovered and loaded.

Existing Extensions

- `com.alibaba.dubbo.common.compiler.support.JdkCompiler`
- `com.alibaba.dubbo.common.compiler.support.JavassistCompiler`

Extension Guide

Directory layout:

```
src
|-main
  |-java
    |-com
      |-xxx
        |-XxxCompiler.java (Compiler implementation)
  |-resources
    |-META-INF
      |-dubbo
        |-com.alibaba.dubbo.common.compiler.Compiler (plain text file with the content: xxx=com.xxx.XxxCompiler)
```

XxxCompiler.java :

```
package com.xxx;

import com.alibaba.dubbo.common.compiler.Compiler;

public class XxxCompiler implements Compiler {
    public Object getExtension(Class<?> type, String name) {
        // ...
    }
}
```

META-INF/dubbo/com.alibaba.dubbo.common.compiler.Compiler :

```
xxx=com.xxx.XxxCompiler
```

Dispatcher Extension

Summary

Thread pool dispatch strategy.

Extension Interface

```
com.alibaba.dubbo.remoting.Dispatcher
```

Extension Configuration

```
<dubbo:protocol dispatcher="xxx" />
<!-- default configuration, will take effect if dispatcher attri
bute is not set in <dubbo:protocol> -->
<dubbo:provider dispatcher="xxx" />
```

Existing Extensions

- `com.alibaba.dubbo.remoting.transport.dispatcher.all.AllDispatcher`
- `com.alibaba.dubbo.remoting.transport.dispatcher.direct.DirectDispatcher`
- `com.alibaba.dubbo.remoting.transport.dispatcher.message.MessageOnlyDispatcher`
- `com.alibaba.dubbo.remoting.transport.dispatcher.execution.ExecutionDispatcher`
- `com.alibaba.dubbo.remoting.transport.dispatcher.connection.ConnectionOrderedDispatcher`

Extension Guide

Directory layout:

```
src
|-main
  |-java
    |-com
      |-xxx
        |-XxxDispatcher.java (Dispatcher implementation)
  |-resources
    |-META-INF
      |-dubbo
        |-com.alibaba.dubbo.remoting.Dispatcher (plain text
        ext file with the content: xxx=com.xxx.XxxDispatcher)
```

XxxDispatcher.java :

```
package com.xxx;

import com.alibaba.dubbo.remoting.Dispatcher;

public class XxxDispatcher implements Dispatcher {
    public Group lookup(URL url) {
        // ...
    }
}
```

META-INF/dubbo/com.alibaba.dubbo.remoting.Dispatcher :

```
xxx=com.xxx.XxxDispatcher
```

ThreadPool Extension

Summary

Thread pool strategy extension for service provider. When server receives one request, it needs a thread from thread pool to execute business logic in service provider.

Extension Interface

```
com.alibaba.dubbo.common.threadpool.ThreadPool
```

Extension Configuration

```
<dubbo:protocol threadpool="xxx" />
<!-- default configuration, it will take effect when threadpool
attribute is not specified in <dubbo:protocol> -->
<dubbo:provider threadpool="xxx" />
```

Existing Extension

- `com.alibaba.dubbo.common.threadpool.FixedThreadPool`
- `com.alibaba.dubbo.common.threadpool.CachedThreadPool`

Extension Guide

Directory layout:

```
src
|-main
  |-java
    |-com
      |-xxx
        |-XxxThreadPool.java (ThreadPool implementation)
  |-resources
    |-META-INF
      |-dubbo
        |-com.alibaba.dubbo.common.threadpool.ThreadPool
        (plain text file with the content: xxx=com.xxx.XxxThreadPool)
```

XxxThreadPool.java :

```
package com.xxx;

import com.alibaba.dubbo.common.threadpool.ThreadPool;
import java.util.concurrent.Executor;

public class XxxThreadPool implements ThreadPool {
    public Executor getExecutor() {
        // ...
    }
}
```

META-INF/dubbo/com.alibaba.dubbo.common.threadpool.ThreadPool :

```
xxx=com.xxx.XxxThreadPool
```


Serialization Extension

Summary

Extension to serializing java object into byte code stream for transporting on the network, and vise versa.

Extension Interface

- `com.alibaba.dubbo.common.serialize.Serialization`
- `com.alibaba.dubbo.common.serialize.ObjectInput`
- `com.alibaba.dubbo.common.serialize.ObjectOutput`

Extension Configuration

```
<!-- protocol serialization style -->
<dubbo:protocol serialization="xxx" />
<!-- default configuration, will take effect if serialization is
not configured in <dubbo:protocol> -->
<dubbo:provider serialization="xxx" />
```

Existing Extension

- `com.alibaba.dubbo.common.serialize.dubbo.DubboSerialization`
- `com.alibaba.dubbo.common.serialize.hessian.Hessian2Serializati`
`on`
- `com.alibaba.dubbo.common.serialize.java.JavaSerialization`
- `com.alibaba.dubbo.common.serialize.java.CompactedJavaSerializat`
`ion`

Extension Guide

Directory layout:

```
src
|-main
  |-java
    |-com
      |-xxx
        |-XxxSerialization.java (Serialization implement
ation)
        |-XxxObjectInput.java (ObjectInput implementatio
n)
        |-XxxObjectOutput.java (ObjectOutput implementat
ion)
      |-resources
        |-META-INF
          |-dubbo
            |-com.alibaba.dubbo.common.serialize.Serializati
on (plain text file with the content: xxx=com.xxx.XxxSerializati
on)
```

XxxSerialization.java :

```
package com.xxx;

import com.alibaba.dubbo.common.serialize.Serialization;
import com.alibaba.dubbo.common.serialize.ObjectInput;
import com.alibaba.dubbo.common.serialize.ObjectOutput;

public class XxxSerialization implements Serialization {
    public ObjectOutput serialize(Parameters parameters, OutputStream output) throws IOException {
        return new XxxObjectOutput(output);
    }
    public ObjectInput deserialize(Parameters parameters, InputStream input) throws IOException {
        return new XxxObjectInput(input);
    }
}
```

META-INF/dubbo/com.alibaba.dubbo.common.serialize.Serialization :

```
xxx=com.xxx.XxxSerialization
```

Transporter Extension

Summary

Transportation extension for communication between server and client.

Extension Interface

- `com.alibaba.dubbo.remoting.Transporter`
- `com.alibaba.dubbo.remoting.Server`
- `com.alibaba.dubbo.remoting.Client`

Extension Configuration

```
<!-- server and client use the same transporter -->
<dubbo:protocol transporter="xxx" />
<!-- server and client use the different transporter -->
<dubbo:protocol server="xxx" client="xxx" />
<!-- default configuration, will take effect when transport/serv
er/client attribute is not set in <dubbo:protocol> -->
<dubbo:provider transporter="xxx" server="xxx" client="xxx" />
```

Existing Extension

- `com.alibaba.dubbo.remoting.transport.transporter.netty.NettyTransporter`
- `com.alibaba.dubbo.remoting.transport.transporter.mina.MinaTransporter`
- `com.alibaba.dubbo.remoting.transport.transporter.grizzly.GrizzlyTransporter`

Extension Guide

Directory layout:

```
src
|-main
  |-java
    |-com
      |-xxx
        |-XxxTransporter.java (Transporter implementation)
        |-XxxServer.java (Server implementation)
        |-XxxClient.java (Client implementation)
  |-resources
    |-META-INF
      |-dubbo
        |-com.alibaba.dubbo.remoting.Transporter (plain
text file with the content: xxx=com.xxx.XxxTransporter)
```

XxxTransporter.java :

```
package com.xxx;

import com.alibaba.dubbo.remoting.Transporter;

public class XxxTransporter implements Transporter {
    public Server bind(URL url, ChannelHandler handler) throws RemotingException {
        return new XxxServer(url, handler);
    }
    public Client connect(URL url, ChannelHandler handler) throws RemotingException {
        return new XxxClient(url, handler);
    }
}
```

XxxServer.java :

```
package com.xxx;

import com.alibaba.dubbo.remoting.transport.transporter.Abstract
Server;

public class XxxServer extends AbstractServer {
    public XxxServer(URL url, ChannelHandler handler) throws Rem
otingException{
        super(url, handler);
    }
    protected void doOpen() throws Throwable {
        // ...
    }
    protected void doClose() throws Throwable {
        // ...
    }
    public Collection<Channel> getChannels() {
        // ...
    }
    public Channel getChannel(InetSocketAddress remoteAddress) {
        // ...
    }
}
```

XxxClient.java :

```
package com.xxx;

import com.alibaba.dubbo.remoting.transport.transporter.Abstract
Client;

public class XxxClient extends AbstractClient {
    public XxxServer(URL url, ChannelHandler handler) throws Rem
otingException{
        super(url, handler);
    }
    protected void doOpen() throws Throwable {
        // ...
    }
    protected void doClose() throws Throwable {
        // ...
    }
    protected void doConnect() throws Throwable {
        // ...
    }
    public Channel getChannel() {
        // ...
    }
}
```

META-INF/dubbo/com.alibaba.dubbo.remoting.Transporter :

```
xxx=com.xxx.XxxTransporter
```

Exchanger Extension

Summary

Exchange message between request and response on network transport layer.

Extension Interface

- `com.alibaba.dubbo.remoting.exchange.Exchanger`
- `com.alibaba.dubbo.remoting.exchange.ExchangeServer`
- `com.alibaba.dubbo.remoting.exchange.ExchangeClient`

Extension Configuration

```
<dubbo:protocol exchanger="xxx" />
<!-- default configuration, will take effect if exchanger attrib
ute is not set in <dubbo:protocol> -->
<dubbo:provider exchanger="xxx" />
```

Existing Extension

```
com.alibaba.dubbo.remoting.exchange.exchanger.HeaderExchanger
```

Extension Guide

Directory layout:


```
src
|-main
  |-java
    |-com
      |-xxx
        |-XxxExchanger.java (Exchanger implementation)
        |-XxxExchangeServer.java (ExchangeServer implementation)
        |-XxxExchangeClient.java (ExchangeClient implementation)
      |-resources
        |-META-INF
          |-dubbo
            |-com.alibaba.dubbo.remoting.exchange.Exchanger
            (plain text file with the content: xxx=com.xxx.XxxExchanger)
```

XxxExchanger.java :

```
package com.xxx;

import com.alibaba.dubbo.remoting.exchange.Exchanger;

public class XxxExchanger implements Exchanger {
    public ExchangeServer bind(URL url, ExchangeHandler handler)
    throws RemotingException {
        return new XxxExchangeServer(url, handler);
    }
    public ExchangeClient connect(URL url, ExchangeHandler handler)
    throws RemotingException {
        return new XxxExchangeClient(url, handler);
    }
}
```

XxxExchangeServer.java :

```
package com.xxx;

import com.alibaba.dubbo.remoting.exchange.ExchangeServer;

public class XxxExchangeServer implements ExchangeServer {
    // ...
}
```

XxxExchangeClient.java :

```
package com.xxx;

import com.alibaba.dubbo.remoting.exchange.ExchangeClient;

public class XxxExchangeClient implements ExchangeClient {
    // ...
}
```

META-INF/dubbo/com.alibaba.dubbo.remoting.exchange.Exchanger :

```
xxx=com.xxx.XxxExchanger
```

Networker Extension

Summary

Extension for peer to peer network grouping.

Extension Interface

```
com.alibaba.dubbo.remoting.p2p.Networker
```

Extension Configuration

```
<dubbo:protocol networker="xxx" />
<!-- default configuration, it takes effect if networker attribute is not set in <dubbo:protocol> -->
<dubbo:provider networker="xxx" />
```

Existing Extension

- `com.alibaba.dubbo.remoting.p2p.support.MulticastNetworker`
- `com.alibaba.dubbo.remoting.p2p.support.FileNetworker`

Extension Guide

Directory layout:

```
src
|-main
  |-java
    |-com
      |-xxx
        |-XxxNetworker.java (Networker implementation)
  |-resources
    |-META-INF
      |-dubbo
        |-com.alibaba.dubbo.remoting.p2p.Networker (plain
n text file with the content: xxx=com.xxx.XxxNetworker)
```

XxxNetworker.java :

```
package com.xxx;

import com.alibaba.dubbo.remoting.p2p.Networker;

public class XxxNetworker implements Networker {
    public Group lookup(URL url) {
        // ...
    }
}
```

META-INF/dubbo/com.alibaba.dubbo.remoting.p2p.Networker :

```
xxx=com.xxx.XxxNetworker
```

TelnetHandler Extension

Summary

Extension to telnet command. All server should support telnet access for operation convenience.

Extension Interface

```
com.alibaba.dubbo.remoting.telnet.TelnetHandler
```

Extension Configuration

```
<dubbo:protocol telnet="xxx,yyy" />
<!-- default configuration, will take effect if telnet attribute
    is not specified in <dubbo:protocol> -->
<dubbo:provider telnet="xxx,yyy" />
```

Existing Extension

- `com.alibaba.dubbo.remoting.telnet.support.ClearTelnetHandler`
- `com.alibaba.dubbo.remoting.telnet.support.ExitTelnetHandler`
- `com.alibaba.dubbo.remoting.telnet.support.HelpTelnetHandler`
- `com.alibaba.dubbo.remoting.telnet.support.StatusTelnetHandler`
- `com.alibaba.dubbo.rpc.dubbo.telnet.ListTelnetHandler`
- `com.alibaba.dubbo.rpc.dubbo.telnet.ChangeTelnetHandler`
- `com.alibaba.dubbo.rpc.dubbo.telnet.CurrentTelnetHandler`
- `com.alibaba.dubbo.rpc.dubbo.telnet.InvokeTelnetHandler`
- `com.alibaba.dubbo.rpc.dubbo.telnet.TraceTelnetHandler`
- `com.alibaba.dubbo.rpc.dubbo.telnet.CountTelnetHandler`
- `com.alibaba.dubbo.rpc.dubbo.telnet.PortTelnetHandler`

Extension Guide

Directory layout:

```
src
|-main
  |-java
    |-com
      |-xxx
        |-XxxTelnetHandler.java (TelnetHandler implement
ation)
      |-resources
        |-META-INF
          |-dubbo
            |-com.alibaba.dubbo.remoting.telnet.TelnetHandle
r (plain text file with the content: xxx=com.xxx.XxxTelnetHandle
r)
```

XxxTelnetHandler.java :

```
package com.xxx;

import com.alibaba.dubbo.remoting.telnet.TelnetHandler;

@Help(parameter="...", summary="...", detail="...")

public class XxxTelnetHandler implements TelnetHandler {
    public String telnet(Channel channel, String message) throws
RemotingException {
        // ...
    }
}
```

META-INF/dubbo/com.alibaba.dubbo.remoting.telnet.TelnetHandler :

```
xxx=com.xxx.XxxTelnetHandler
```

用法

```
telnet 127.0.0.1 20880  
dubbo> xxx args
```

StatusChecker Extension

Summary

Extension to check status of resources service depends on. This status checker can be used in both telnet status command and status page.

Extension Interface

```
com.alibaba.dubbo.common.status.StatusChecker
```

Extension Configuration

```
<dubbo:protocol status="xxx,yyy" />
<!-- default configuration, will take effect if no status attrib
ute is configured in <dubbo:protocol> -->
<dubbo:provider status="xxx,yyy" />
```

Existing Extension

- `com.alibaba.dubbo.common.status.support.MemoryStatusChecker`
- `com.alibaba.dubbo.common.status.support.LoadStatusChecker`
- `com.alibaba.dubbo.rpc.dubbo.status.ServerStatusChecker`
- `com.alibaba.dubbo.rpc.dubbo.status.ThreadPoolStatusChecker`
- `com.alibaba.dubbo.registry.directory.RegistryStatusChecker`
- `com.alibaba.dubbo.rpc.config.spring.status.SpringStatusChecker`
- `com.alibaba.dubbo.rpc.config.spring.status.DataSourceStatusChecker`

Extension Guide

Directory layout:


```
src
|-main
  |-java
    |-com
      |-xxx
        |-XxxStatusChecker.java (StatusChecker implement
ation)
      |-resources
        |-META-INF
          |-dubbo
            |-com.alibaba.dubbo.common.status.StatusChecker
(plain text file with the content: xxx=com.xxx.XxxStatusChecker)
```

XxxStatusChecker.java :

```
package com.xxx;

import com.alibaba.dubbo.common.status.StatusChecker;

public class XxxStatusChecker implements StatusChecker {
    public Status check() {
        // ...
    }
}
```

META-INF/dubbo/com.alibaba.dubbo.common.status.StatusChecker :

```
xxx=com.xxx.XxxStatusChecker
```

Container Extension

Summary

Service container extension, useful for loading custom contents.

Extension Interface

```
com.alibaba.dubbo.container.Container
```

Extension Configuration

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

Existing Extensions

- `com.alibaba.dubbo.container.spring.SpringContainer`
- `com.alibaba.dubbo.container.spring.JettyContainer`
- `com.alibaba.dubbo.container.spring.Log4jContainer`

Extension Guide

Directory layout:

```
src
|-main
  |-java
    |-com
      |-xxx
        |-XxxContainer.java (Container implementation)
  |-resources
    |-META-INF
      |-dubbo
        |-com.alibaba.dubbo.container.Container (plain text
        ext file with the content: xxx=com.xxx.XxxContainer)
```

XxxContainer.java :

```
package com.xxx;

com.alibaba.dubbo.container.Container;

public class XxxContainer implements Container {
    public Status start() {
        // ...
    }
    public Status stop() {
        // ...
    }
}
```

META-INF/dubbo/com.alibaba.dubbo.container.Container :

```
xxx=com.xxx.XxxContainer
```

Page Extension

Summary

Extension for page handler

Extension Interface

```
com.alibaba.dubbo.container.page.PageHandler
```

Extension Configuration

```
<dubbo:protocol page="xxx,yyy" />
<!-- default configuration, will take effect if page attribute i
s not set in <dubbo:protocol> -->
<dubbo:provider page="xxx,yyy" />
```

Existing Extension

- `com.alibaba.dubbo.container.page.pages.HomePageHandler`
- `com.alibaba.dubbo.container.page.pages.StatusPageHandler`
- `com.alibaba.dubbo.container.page.pages.LogPageHandler`
- `com.alibaba.dubbo.container.page.pages.SystemPageHandler`

Extension Guide

Directory layout:

```
src
|-main
  |-java
    |-com
      |-xxx
        |-XxxPageHandler.java (PageHandler implementation)
  |-resources
    |-META-INF
      |-dubbo
        |-com.alibaba.dubbo.container.page.PageHandler (
plain text file with the content: xxx=com.xxx.XxxPageHandler)
```

XxxPageHandler.java :

```
package com.xxx;

import com.alibaba.dubbo.container.page.PageHandler;

public class XxxPageHandler implements PageHandler {
    public Group lookup(URL url) {
        // ...
    }
}
```

META-INF/dubbo/com.alibaba.dubbo.container.page.PageHandler :

```
xxx=com.xxx.XxxPageHandler
```

Cache Extension

Summary

Cache the return value, use request parameter as the key.

Extension Interface

```
com.alibaba.dubbo.cache.CacheFactory
```

Extension Configuration

```
<dubbo:service cache="lru" />
<!-- method level cache -->
<dubbo:service><dubbo:method cache="lru" /></dubbo:service>
<!-- 缺省值设置，当<dubbo:service>没有配置cache属性时，使用此配置 -->
<!-- default configuration, will take affect if cache attribute
isn't configured in <dubbo:service> -->
<dubbo:provider cache="xxx,yyy" />
```

Existing Extensions

- `com.alibaba.dubbo.cache.support.lru.LruCacheFactory`
- `com.alibaba.dubbo.cache.support.threadlocal.ThreadLocalCacheFactory`
- `com.alibaba.dubbo.cache.support.jcache.JCacheFactory`

Extension Guide

Directory layout:

```
src
|-main
  |-java
    |-com
      |-xxx
        |-XxxCacheFactory.java (CacheFactory implementat
ion)
      |-resources
        |-META-INF
          |-dubbo
            |-com.alibaba.dubbo.cache.CacheFactory (plain te
xt file with contents: xxx=com.xxx.XxxCacheFactory)
```

XxxCacheFactory.java :

```
package com.xxx;

import com.alibaba.dubbo.cache.CacheFactory;

public class XxxCacheFactory implements CacheFactory {
    public Cache getCache(URL url, String name) {
        return new XxxCache(url, name);
    }
}
```

XxxCacheFactory.java :

```
package com.xxx;

import com.alibaba.dubbo.cache.Cache;

public class XxxCache implements Cache {
    public Cache(URL url, String name) {
        // ...
    }
    public void put(Object key, Object value) {
        // ...
    }
    public Object get(Object key) {
        // ...
    }
}
```

META-INF/dubbo/com.alibaba.dubbo.cache.CacheFactory :

```
xxx=com.xxx.XxxCacheFactory
```


Validation Extension

Summary

Extension for parameter validation.

Extension Interface

```
com.alibaba.dubbo.validation.Validation
```

Extension Configuration

```
<dubbo:service validation="xxx,yyy" />
<!-- default configuration, it will take effect when there's no
validation attribute specified in <dubbo:service> -->
<dubbo:provider validation="xxx,yyy" />
```

Existing Extension

```
com.alibaba.dubbo.validation.support.jvalidation.JValidation
```

Extension Guide

Directory layout:

```
src
|-main
  |-java
    |-com
      |-xxx
        |-XxxValidation.java (Validation implementation)
  |-resources
    |-META-INF
      |-dubbo
        |-com.alibaba.dubbo.validation.Validation (plain
text file with the content: xxx=com.xxx.XxxValidation)
```

XxxValidation.java :

```
package com.xxx;

import com.alibaba.dubbo.validation.Validation;

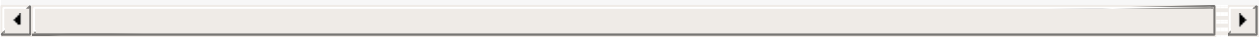
public class XxxValidation implements Validation {
    public Object getValidator(URL url) {
        // ...
    }
}
```

XxxValidator.java :

```
package com.xxx;

import com.alibaba.dubbo.validation.Validator;

public class XxxValidator implements Validator {
    public XxxValidator(URL url) {
        // ...
    }
    public void validate(Invocation invocation) throws Exception
    {
        // ...
    }
}
```



META-INF/dubbo/com.alibaba.dubbo.validation.Validation :

```
xxx=com.xxx.XxxValidation
```

LoggerAdapter Extension

Summary

Extension for adapting logger output

Extension Interface

```
com.alibaba.dubbo.common.logger.LoggerAdapter
```

Extension Configuration

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

Or:

```
-Ddubbo:application.logger=xxx
```

Existing Extension

- `com.alibaba.dubbo.common.logger.slf4j.Slf4jLoggerAdapter`
- `com.alibaba.dubbo.common.logger.jcl.JclLoggerAdapter`
- `com.alibaba.dubbo.common.logger.log4j.Log4jLoggerAdapter`
- `com.alibaba.dubbo.common.logger.jdk.JdkLoggerAdapter`

Extension Guide

Directory layout:

```
src
|-main
  |-java
    |-com
      |-xxx
        |-XxxLoggerAdapter.java (LoggerAdapter implement
ation)
      |-resources
        |-META-INF
          |-dubbo
            |-com.alibaba.dubbo.common.logger.LoggerAdapter
(plain text file with the content: xxx=com.xxx.XxxLoggerAdapter)
```

XxxLoggerAdapter.java :

```
package com.xxx;

import com.alibaba.dubbo.common.logger.LoggerAdapter;

public class XxxLoggerAdapter implements LoggerAdapter {
    public Logger getLogger(URL url) {
        // ...
    }
}
```

XxxLogger.java :

```
package com.xxx;

import com.alibaba.dubbo.common.logger.Logger;

public class XxxLogger implements Logger {
    public XxxLogger(URL url) {
        // ...
    }
    public void info(String msg) {
        // ...
    }
    // ...
}
```

META-INF/dubbo/com.alibaba.dubbo.common.logger.LoggerAdapter :

```
xxx=com.xxx.XxxLoggerAdapter
```

Public Agreement

This document is Dubbo public agreement, we expect all extension points comply with it.

URL

- All extension points must include URL parameter, design URL as a context information which throughputs the whole extension point design system.
- URL standard style: `protocol://username:password@host:port/path?key=value&key=value`

Logging

- Print `ERROR` log for unrecoverable and NEED TO ALARM situation.
- Print `WARN` log for recoverable exception or transient state inconsistency.
- Print `INFO` log for normally status.

Coding convention

Code style

The source and JavaDoc of Dubbo follow below specifications:

- [Code Conventions for the Java Programming Language](#)
- [How to Write Doc Comments for the Javadoc Tool](#)

Exception and Logging

- Log more context information as possible, such as error reason, error server address, client address, registry center address, dubbo version and so on.
- Try to put the main cause at the front, and display all other context information with key-value pairs after it.
- Log is not printed where the exception is thrown, log level is determined by the final exception handler, and must print log when discarding exception.
- `ERROR` log means NEED TO ALARM, `WARN` log means COULD AUTO RECOVERY, `INFO` log means NORMAL.
- Suggestion: config `ERROR` log in Monitor center for real-time alarm, summary and send `WARN` log weekly.
- `RpcException` is the ONLY external exception of Dubbo, all internal exceptions must be transferred to `RpcException` if need to throw out to user.
- `RpcException` CAN NOT have sub-class, all types of information are identified with ErrorCode in order to keep compatible.

Configuration and URL

- Use initials and camelCase for multiple words for object properties ¹.
- Use lowercase and split by '-' for multiple words for config properties ².
- Use lowercase and split by '.' for multiple words for URL properties ³.
- Use URL transfer parameters as possible, Don't define Map or other types, config information also transfer to URL style.

- Minimize URL nesting to keep URL simplicity.

Unit testing and integration testing

- Use JUnit and EasyMock for unit testing, use TestNG for integration testing, use DBUnit for database testing.
- Don't put large integration test case in unit testing for running speed of unit test case.
- Use `try...finally` or `tearDown` to release resource for all test cases of unit testing.
- Minimize test case that with `while` loop which need waiting response, use to make the logic in timer as function for timer and net testing.
- For fail-safe testing, unified use `LogUtil` assertion log output.

Extension point base class and AOP

- AOP class should be named as `XxxWrapper` , Base class should be named as `AbstractXxx` .
- Use AOP for combine relationship between extension points, `ExtensionLoader` only loads extension points, including AOP extension.
- Try to use loc inject dependency of extension points, Don't direct dependent on factory method of `ExtensionLoader` .
- Try to use AOP implement the common action of extension points, instead of using base class, such as the `isAvailable` checking before load balancing, which is independent of load balance. Close the URL parameters which no need to check.
- Use base class for abstraction for a variety of similar types, such as RMI, Hessian 3rd protocols which have generated interface proxy, only transfer interface proxy to `Invoker` to complete bridging, and public base class can do the logic.
- The base class is also part of the SPI, and each extension should have a convenient base class support.

Module and packaging

- Base on reusability for packaging, dividing the interface, base class and large implementation into separate modules.
- Put all interfaces under the base package of module, and put base classes in support subpackage, different implementations are placed under the subpackage named by extension point.
- Try to keep subpackage dependent on parent package, NOT reverse.

1. Java convention ↩

2. Spring convention ↩

3. Dubbo convention ↩

Versions

New feature development and **stability improvement** are equally important to product. But adding new features will affect stability, dubbo uses the following version development pattern to achieve a good balance.

Two versions evolving at the same time

- BugFix Version : low version , e.g. `2.4.x` . This is called the GA version, which can be applied in production. We are supposed only to fix bugs in this version, and increase the third version number when release.
- Feature Version : high version, e.g. `2.5.x` . We add new features to this version, so applications have opportunities try new features.

When features in `2.5.x` are proved stable enough, we will announce `2.5.x` as a beta release.

When `2.5.x` proved stable after enough test on enough applications :

- `2.5.x` , the GA Version, only do BugFix, the main version to be used. We can try to promote applications to upgrade to GA at the desired time.
- `2.4.x` , no longer maintained. When bugs appear, applications have no choice but upgrade to the latest stable version- Sunset Clause
- We create a new branch `2.6.0` based on `2.5.x` for new features.

Pros

- GA Version are promised stable:
 - only BugFix
 - GA Version got enough tests before promotion
- New features can respond quickly in Feature Version and allow applications to try that
- Significantly reduces development and maintenance costs

The responsibilities of users

Users should always keep in track with the GA Version, make sure all bugs were fixed.

There is a fake proposition: regular upgrades bring more risks. Here's the reasons:

- GA remains stable after a trial period.
- Bugs find on GA will be fixed immediately.
- Comparing with the on-need-upgrade (only upgrade when find a serious problem, and may span multiple versions), upgrade periodically can flat risk. Experienced a long cycle of large projects, students will have such an experience, the tripartite library version does not upgrade for a long time, the result of the problem had to upgrade to the new version (across multiple versions) a huge risk.

Contribution

Flow

- Direct adding new project, black-box dependent on Dubbo for function extension.
- Fork Dubbo on Github for BUG fixing or modify the framework.
- Pull Request after changing.

Tasks

Funcation	Type	Priority	Status
Translation	Document	High	Unclaimed
translation	Document	High	Unclaimed
Extension point compatibility testing	Testing	High	Claimed
Performance benchmark testing	Testing	High	Unclaimed
Functional unit testing	Testing	High	Unclaimed
JTA/XA distributed transaction	Interceptor extension	High	Unclaimed
Thrift	Protocol extension	High	Developing done
ICE	Protocol extension	High	Unclaimed
ACE	Protocol extension	Low	Unclaimed
JSON-RPC	Protocol extension	Low	Unclaimed
XML-RPC	Protocol extension	Low	Unclaimed
JSR181&CXF(WebService)	Protocol extension	High	Developing done
	Protocol		

	extension		
JMS&ActiveMQ	Protocol extension	High	Unclaimed
Protobuf	Serialization extension	High	Researchchir
Avro	Serialization extension	Low	Unclaimed
XSocket	Transmission extension	Low	Unclaimed
CGLib	Dynamic proxy extension	Low	Unclaimed
JNDI	Registry extension	High	Unclaimed
LDAP	Registry extension	Low	Unclaimed
JSR140&SLP	Registry extension	High	Unclaimed
UDDI	Registry extension	High	Unclaimed
JMX	Monitoring expansion	High	Unclaimed
SNMP	Monitoring expansion	High	Unclaimed
Cacti	Monitoring expansion	High	Unclaimed
Nagios	Monitoring expansion	High	Unclaimed
Logstash	Monitoring expansion	High	Unclaimed
JRobin	Monitoring expansion	High	Unclaimed
Maven	Package management	Low	Unclaimed
Subversion	Package management	Low	Unclaimed
	Package		

JCR/JSR283	Package management	Low	Unclaimed
SimpleDeployer	Local deployment agent	Low	Unclaimed
SimpleScheduler	Scheduler	Low	Unclaimed

Checklist

Checklist before release

- github milestones
- github change lists
- Travis CI
- test code
- find bugs

Checklist for bigfix versions

- Create a *github issue* before coding
- Create *unit test* before bugfix
- Review
- Test your code (Normal process / Abnormal process)
- Record your design on *github issue*
- Complete javadoc and comment in code
- Manager for every version, responsible for scope and check

Bad Smell

Ugly Dubbo design or implementation will be record here.

URL Conversion

1. Point to Point Service export and refer

service directly export :

```
EXPORT(dubbo://provider-address/com.xxx.XxxService?version=1.0.0")
```

service directly refer :

```
REFER(dubbo://provider-address/com.xxx.XxxService?version=1.0.0)
```

2. Export servie by registry

export service to registry :

```
EXPORT(registry://registry-address/com.alibaba.dubbo.registry.RegistrySerevice?registry=dubbo&export=ENCODE(dubbo://provider-address/com.xxx.XxxService?version=1.0.0))
```

acquire registry :

```
url.setProtocol(url.getParameter("registry", "dubbo"))
GETREGISTRY(dubbo://registry-address/com.alibaba.dubbo.registry.RegistrySerevice)
```

registry service address :

```
url.getParameterAndDecoded("export"))  
REGISTER(dubbo://provider-address/com.xxx.XxxService?version=1.0  
.0)
```

3. Refer service from registry

refer service from registry :

```
REFER(registry://registry-address/com.alibaba.dubbo.registry.Reg  
istrySerevice?registry=dubbo&refer=ENCODE(version=1.0.0))
```

acquire registry :

```
url.setProtocol(url.getParameter("registry", "dubbo"))  
GETREGISTRY(dubbo://registry-address/com.alibaba.dubbo.registry.  
RegistrySerevice)
```

subscribe service address :

```
url.addParameters(url.getParameterAndDecoded("refer"))  
SUBSCRIBE(dubbo://registry-address/com.xxx.XxxService?version=1.  
0.0)
```

notify service address :

```
url.addParameters(url.getParameterAndDecoded("refer"))  
NOTIFY(dubbo://provider-address/com.xxx.XxxService?version=1.0.0  
)
```

4. Registry push route rule

registry push route rule :

```
NOTIFY(route://registry-address/com.xxx.XxxService?router=script
&type=js&rule=ENCODE(function{...}))
```

acquire routers :

```
url.setProtocol(url.getParameter("router", "script"))
GETROUTE(script://registry-address/com.xxx.XxxService?type=js&ru
le=ENCODE(function{...}))
```

5. Load route rule from file

load route rule from file :

```
GETROUTE(file://path/file.js?router=script)
```

acquire routers :

```
url.setProtocol(url.getParameter("router", "script")).addParamet
er("type", SUFFIX(file)).addParameter("rule", READ(file))
GETROUTE(script://path/file.js?type=js&rule=ENCODE(function{...}
))
```

Invoke parameters

- path service path
- group service group
- version service version
- dubbo current dubbo release version
- token verify token
- timeout invocation timeout

SPI Loading

1. SPI Auto Adaptive

When ExtensionLoader loads SPI, It will check spi attributes(using set method) . If one attribute is SPI, ExtensionLoader will load the SPI implementation. Auto injected object is an adaptive instance (proxy) ,because the real implementation is confirmed only in execution stage. ◦ when adaptive spi is invoked, Dubbo will choose the real implementation and executes it. Dubbo choose the right implementation according to the parameters that the mehod defines.

All the inner SPIs that Dubbo defines have the URL parameter defined for the method invocation. Adaptive SPI uses URL to determine which implementation is needed. One specific Key and Value in the URL confirms the usage of the specific implementation, All these is done by adding `@Adaptive` annotation.

```
@Extension
public interface Car {
    @Adaptive({"http://10.20.160.198/wiki/display/dubbo/car.type"
, "http://10.20.160.198/wiki/display/dubbo/transport.type"})
    public run(URL url, Type1 arg1, Type2 arg2);
}
```

For the rules above , ExtensionLoader will create a adaptive instance for each SPI injected.

ExtensionLoader generated adaptive classes look like :

```

package <package name for SPI interface>;

public class <SPI interface name>$Adaptive implements <SPI inter
face> {
    public <contains @Adaptive annotation method>(<parameters>)
    {
        if(parameters containing URL Type?) using URL parameter
        else if(method returns URL) using the return URL
        # <else throw exception,inject SPI fail!>

        if(URL acquired == null) {
            throw new IllegalArgumentException("url == null");
        }

        According to the Key order from @Adaptive annotation, ge
t the Value from the URL as the real SPI name
        if no value is found then use the default SPI implementa
tion. If no SPI point, throw new IllegalStateException("Fail to
get extension");

        Invoke the method using the spi and return the result.
    }

    public <method having annotation @Adaptive>(<parameters>) {
        throw new UnsupportedOperationException("is not adaptive
method!");
    }
}

```

@Adaptive annotation usage :

If no value is configed for those Keys in URL , default SPI implementation is used . For example , String[] {"key1", "key2"} , firstly Dubbo will look up value for key1 and use it as SPI name;if key1 value is not founded then look up for key2 , if value of key2 is also not found ,then use default spi implementation. If no default implementation is configed, then the method will throw IllegalStateException . if not configed , then default implement is lower case of the interface class full

package name. For Extension interface

```
com.alibaba.dubbo.xxx.YyyInvokerWrapper , default value is new  
String[] {"yyy.invoker.wrapper"}
```

Callback Function

1. Parameter Callback

main theory : in the persistent connection for one consumer->provider , export a service in Consumer side , provider side can reversely call the instance in consumer side.

Implement details :

- For exchanging interface instance in transmittion, auto export and auto refer is implemented in DubboCodec . Need to seperate business logic and codec logic.
- you will need to judge whether needing callback when getting exporter from invocation , if needed, get the callback instance id from the attachments. By using this method, consumer side can implement the callback interface with different implementations.

2. Event Notification

main theory : when Consumer executing invoke method , judging if any configuration for onreturn/onerror... put the method for onreturn to the callback list of the async invocation.

Implement details : parameters is passed using URL , but string-object is not supported for URL, so the method is stored in staticMap , it needs to be optimized.

Lazy Connection

DubboProtocol specific features, default disabled

When client creating proxy for server, do not establish TCP persistent connection at first, only init the connecton when data is needing transmsion.

This feather will disable the connection retry policy , resend the data again(if connection is lost when sending data ,try to establish a new connection to send data)

Share Connection

DubboProtocol specific features, default enabled ◦

JVM A export many services , JVM B refer more than one services of A , Share Connection means those different services invocations between A and B uses the same TCP connection to transmit data, reducing server connections.

Implement details : when using share connection for the same address , you need pay more attention to the invoker's destroy action.on one hand, you should close the connection when all the invokers refering the same address is destroyed, on another hand ,you should not close the connection when not all of the invokers are destroyed. In design implementation, we uses a strategy called reference count , we create a connection called Lazy connection for exceptions not affacting business when closing the connection just in case.

sticky policy

when existing many providers and configing the sticky policy , invocation will be sent to the same provider as last invocation. Sticky Policy opens the lazy attribute of connection, for avoiding open useless connectons.

provider selecting logic

1. existing multi providers , firstly select by Loadbalance ◦ If the selected provider is available ,then just doing the invocation
2. If the selected provider is not available in stage 1, then choose from the remaining ,if available then doing the inovation
3. If all providers are not available , rescan the list(not choosen invoker

first),juding if any provider is available, if existing,doing the invocatiion.

4. If no available provider in stage 3, then the next invoker of the invoker of stage 1 will be choosen(if not the last one),avoiding collision.

Compatibility test

Dubbo's protocol, communication, serialization, registry, load balancing and other SPI all offer alternative strategies for different application scenarios while our test cases are very scattered. Ours is always uncertain whether it can satisfy the complete contract of the extension point when users need to add a new implementation.

Thus we need to use TCK (Technology Compatibility Kit) for the core extension points. When users add a new implementation, compatibility with the rest of the framework can be ensured with TCK. This can effectively improve the overall health and also facilitate the access of the third party extenders, which accelerates the maturity of the open source community.

Xingzhi from the open source community is already working on this part. His preliminary idea is to build a TCK framework for Dubbo drawing on the CDI-TCK of JBoss first, then realize the TCK implementing case of Dubbo.

Reference : <http://docs.jboss.org/cdi/tck/reference/1.0.1-Final/html/introduction.html>

Anyone interested is welcomed to work on this together.

Protocol TCK

TODO

Registry TCK

TODO