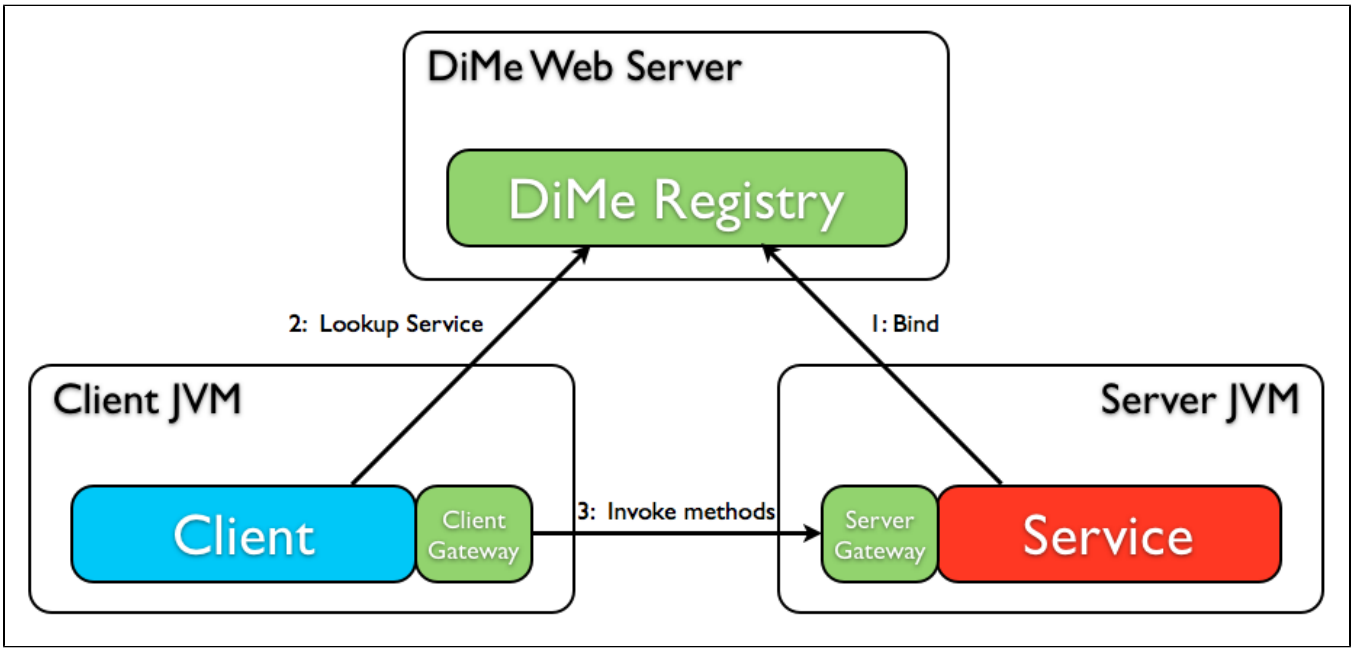
DistributeMe HowTo

DistributeMe HowTo

Distributed service workflow



Simplified workflow description:

1. Service JVM: Binding service to the DistributeMe Registry.

 1.1 Create and register in the local RMI registry LifecycleSupportServer and EventServiceRMIBridgeServer (background features)

 1.2 Create and bind local instance of the service implementation to the local RMI registry.

 1.3 Bind service descriptor/reference/link to the DistributeMe Registry via HTTP request.

2. Client JVM: Get generated service stub (Client Gateway), that implements service interface. Stub on init is requesting DistributeMe Registry via HTTP for the service descriptor by it's name.

3. Client JVM: Invoking service interface methods via service stub, who is redirecting it via RMI to the service instance on the Service JVM, directly (without Registry mediation).

DistributeMe Registry

DistributeMe Registry, in a few worlds, is just a Web Application for storing distributed services descriptors. It should be online when any client want to find any distributed remote service for self needs.

Distributed services XML list can be accessed by url's like: "http://some.server:9229/distributeme/registry/list"

Registry XML format

If you call your registry you will get something like this:

```
<services>
...
<service serviceId="org_distributeme_test_echo_EchoService" host="192.168.200.107" port="9250" protocol="rmi"
         instanceId="awodzrtvcp" globalId="rmi://org_distributeme_test_echo_EchoService" registrationString="
rmi://org_distributeme_test_echo_EchoService.awodzrtvcp@192.168.200.107:9250"/>
...
</services>
```

You will find a service entry for each registered service. The values are explained in following table:

Value	Explanation	Example
serviceId	The id of the service which is derived from the source interface	org_distributeme_test_echo_EchoService
host	Host on which the service is running. This is typically the first entry in the /etc/hosts	192.168.200.107
port	TCP port on which embedded rmiregistry is listening	Default range is 9250 - 9299
protocol	Protocol, that the service is speaking	rmi, corba or webservice
instanceId	Unique instance id, which changes on each restart	awodzrtvcp - 10 digits generated literal string
globalId	Used by the client to obtain service instance	rmi://org_distributeme_test_echo_EchoService
registration String	Complete service definition which is sent by the server instance to the registry for the registration. Uniquely describes a service instance	rmi://org_distributeme_test_echo_EchoService. awodzrtvcp@192.168.200.107:9250

Distributed service lookup

It is quite simple to lookup service you need in the client code:

1. Add remote alias and generated class factory to the MetaFactory:

```
MetaFactory.addAlias(EchoService.class, Extension.REMOTE);
try {
        MetaFactory.addFactoryClass(EchoService.class, Extension.REMOTE,
                        (Class<ServiceFactory<EchoService>>) Class.forName("lu.netservices.cdate.business.echo.
generated.RemoteEchoServiceFactory"));
} catch (ClassNotFoundException e) {
        log.fatal("Couldn't load factory class " + factoryClassName + " for service: " + EchoService.class);
}
```

2. Get service (actually, service stub) via MetaFactory and now can start use it:

```
try {
        echoService = MetaFactory.get(EchoService.class);
} catch (MetaFactoryException e) {
        throw new RuntimeException("Service intialization fail", e);
}
```

Remote service methods calls

There are no differences for the client code, between remote service methods calls and any other interfaces calls (all calls "remoting" is absolutely transparent), the difference is only in the interface instance initialization (see above):

```
long result = echoService.echo(2000);
```

On the background layer, remote calls are going through the next path: Client code -> Client Gateway (service DiMe generated stub) -> RMI Proxy on client side -> RMI -> Sever Gateway (RMI Proxy on server side) -> Service code on server side

The role of the generated code

For the "remoting" needs, for the @DestributeMe annotated services interfaces, some additional java code have to be generated, see table below:

Artifact	Features / Responsibilities	Class Name Example
Server	- Contains main() method - Used to run service on the server side - Performs all necessary service creations and registration/binding procedures	lu.netservices.cdate.business.echo.generated.EchoServer
Remote Service	- Interface, that is copy of the original service interface, plus some remote features - Used	lu.netservices.cdate.business.echo.generated.RemoteEchoService
Remote Service Stub	- Client side service interface implementation - Looking for service descriptor in the DiMe Registry - Delegates client calls to the RMI proxy on the client side	lu.netservices.cdate.business.echo.generated. RemoteEchoServiceStub
Remote Service Factory	- Create instances of the Remote Service Stub	lu.netservices.cdate.business.echo.generated. RemoteEchoServiceFactory
Remote Service Skeleton	- Server side service interface implementation - Wraps service instance, used, as example, for adding service Moskito monitoring	lu.netservices.cdate.business.echo.generated. RemoteEchoServiceSkeleton
Service Constants	- Contains some constants for service needs, e.g. ServiceId	lu.netservices.cdate.business.echo.generated. EchoServiceConstants