The current version support two styles rpc.

- RPC based on interface,when u use this style,u should implement an interface,then will call this implemention class directly.
- RPC based on reflection,when u use this style,client use proxy to call server method,the framework will use reflection to call approviate instance and method.

Use this rpc framework,u can realize rpc very simply.

Mina RPC

Direct Call RPC

Server

- Implement processor

```
public class MyProcessor implements ServerProcessor{
    public Object handle(Object request) throws Exception {
      return (String)request + " return by server";
    }
}
```

- Register processor & start server

```
public class MyMinaServer{
   public static void main(String[] args) throws Exception{
       Server server = new MinaServer();
       server.registerProcessor(SimpleProcessorProtocol.TYPE,String.class.getName(),new
MyProcessor());
       ThreadFactory tf = new NamedThreadFactory("BUSINESSTHREADPOOL");
       ExecutorService threadPool = new ThreadPoolExecutor(20, 100,
                   300, TimeUnit.SECONDS, new SynchronousQueue<Runnable>(), tf);
       server.start(18888, threadPool);
    }
}
```

Then your server will listen at 18888 and wait for request.

Client

If u want to call MyProcessor,u can follow below steps.

- Get Client instance

```
// clientNums mean how many connections to server
// we appreciate single connection to the same serverIP & serverPort
Client client =
MinaClientFactory.getInstance().get(serverIP,serverPort,connectTimeOut,clientNums);
```

- Use Client to call

```
// rpcTimeout mean how long u will wait here for server return
// datatype mean serialize/deserialize
//       current support Java or Hessian or Protobuf
// if u use Protobuf,pls call PBDecoder.addMessage first.
String
result=client.invokeSync("hello",rpcTimeout,Codecs.HESSIAN_CODEC,SimpleProcessorProtocol.TYPE);
```

Reflection RPC

Server

- Your processor

```
public interface HelloWorldService{
    public String sayHello(String word);
}
public class HelloWorldComponent implements HelloWorldService{
    public String sayHello(String word){
        return word + " return by server";
    }
}
```

- Register processor instance & start server

```
public class MyMinaServer{
    public static void main(String[] args) throws Exception{
        Server server = new MinaServer();
        server.registerProcessor(RPCProtocol.TYPE,"helloworld",new
HelloWorldComponent());
        ThreadFactory tf = new NamedThreadFactory("BUSINESSTHREADPOOL");
        ExecutorService threadPool = new ThreadPoolExecutor(20, 100,
                        300, TimeUnit.SECONDS, new SynchronousQueue<Runnable>(), tf);
        server.start(18888, threadPool);
    }
}
```

Client

- Create client proxy

```
Map<String, Integer> methodTimeouts = new HashMap<String, Integer>();
// so u can specialize some method timeout
methodTimeouts.put("*", timeout);
List<InetSocketAddress> servers = new ArrayList<InetSocketAddress>();
servers.add(new InetSocketAddress(serverIP, serverPort));
```

```
        // Protocol also support Protobuf & Java,if u use Protobuf,u need call
PBDecoder.addMessage first.
        int codectype = Codecs.HESSIAN_CODEC;
        HelloWorldService service = (HelloWorldService) Proxy.newProxyInstance(
                        this.getClass().getClassLoader(),
                        new Class<?>[] { HelloWorldService.class },
                        new MinaClientInvocationHandler(servers, clientNums,connectTimeout,
"helloworld", methodTimeouts,codectype,RPCProtocol.TYPE));
```

- Use proxy to call

```
        String result = service.sayHello("hello");
```

Netty RPC

It's very similiar to Mina RPC,u only need make below changes:

- MinaServer to NettyServer;
- MinaClientFactory to NettyClientFactory;
- MinaClientInvocationHandler to NettyClientInvocationHandler;

Grizzly RPC

It's very similiar to Mina RPC,u only need make below changes:

- MinaServer to GrizzlyServer;
- MinaClientFactory to GrizzlyClientFactory;
- MinaClientInvocationHandler to GrizzlyClientInvocationHandler;