**Linkage**

* Purpose And Background

When establish a big Web Site, it's inevitable to use the distribution technology. After we divide the whole system into distributed parts, we could call them as services each other. Each service could provide the specific function like Sending Message to the user when Login, Writting the records into the database.

But how to call a distributed service between application and service, service and service? Is there a need that we establish a protocol for each service calling. Answer is definitely not, we will design a unified middleware which is used between the service and the service caller.

Linkage is such a middleware which could bond all of the distributed services into one whole system. Chart below, applications, services, db, cache are placed at distributed computers. Between each is linkage. We use linkage to deal with the communication with services and applications.

Application1

Application2

Application3

Service 1

Service 2

Service 3

DB

File System

Cache

* Overall Design

Chart below show the main structure of the linkage. Client side & Service side will communicate with each other by different type of io. NIO is now most widely used, so we use this type of io. But still keep the extension for other type of io. The wapper layer placed above the io layer. In this layer, we will wrap/unwrap messages with the io protocol. The serialization/deserialization layer placed above the wrapper layer, it uses the serialization protocol. Above this layer is the service access/provider layer. In this layer, we could also define the calling and being called protocol.

Communication

Message Wrapper

Message Unwrapper

Communication

Serialization/Deserialization

ServiceAccess

Service Provider

Serialization/Deserialization

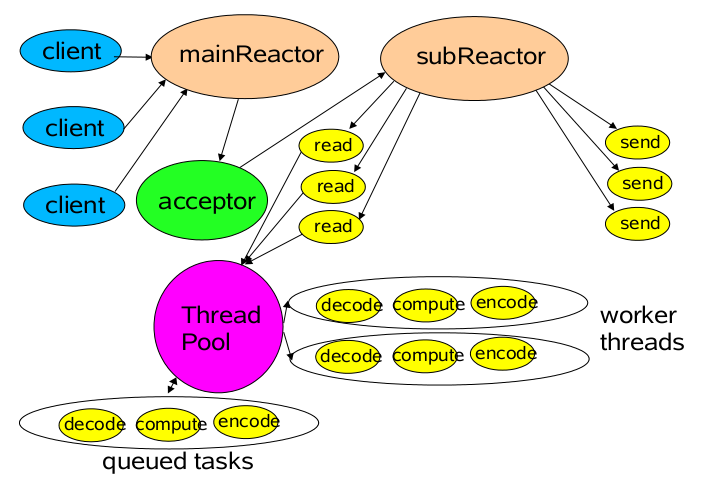
Client Side

Service Side

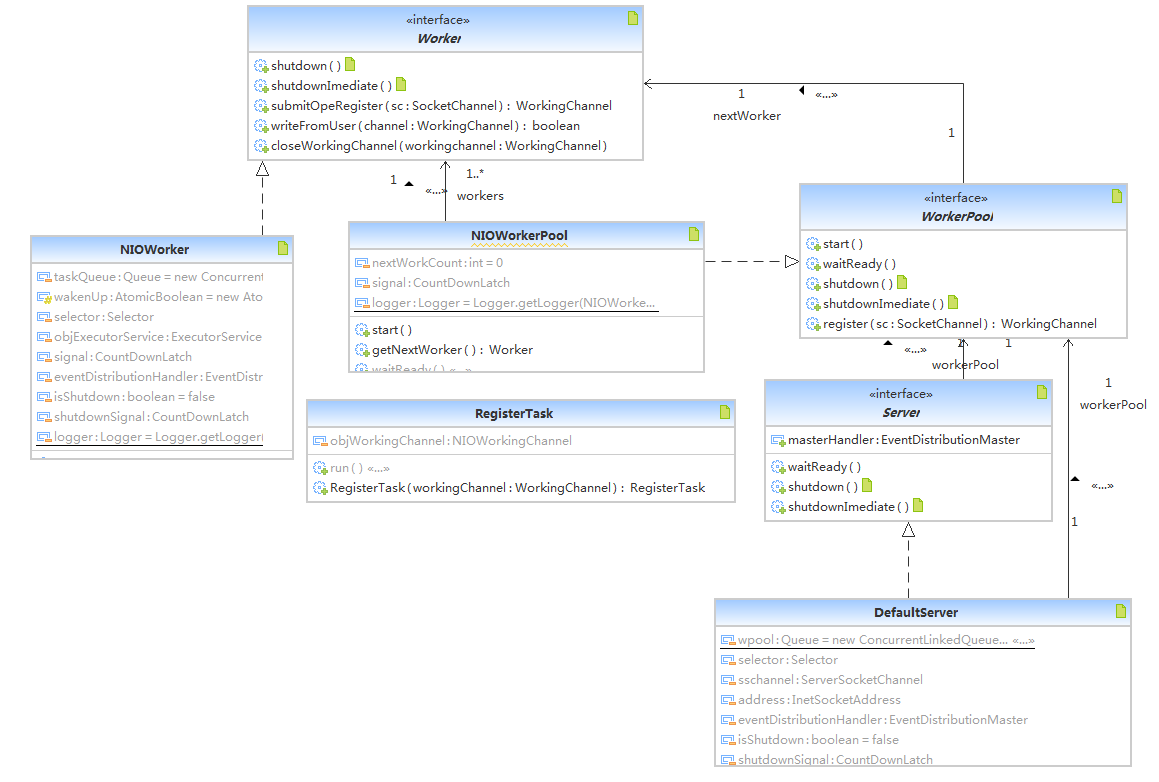
Linkage

* Communication Layer

Below chart shows the classic NIO model. There is one main reactor accepting all connections then the acceptor will forward the connection to the sub reactor. Sub reactor will deal with the read&write operation of the connection. Netty actually use this model for its nio case. We would use this model as well in order to deal with more concurrent requrest.



Below chart is class diagram of nio part, the server act as the main reactor which will accept all of the connection request and register to the worker pool. The worker pool retains the worker and the worker is actually the sub reactor. Server will listen the port, once there is new connection coming. The server will register the channel to the worker through the worker pool.



* Wrap&Unwrap Layer

In this layer, all the messages should be wrapped before sent to the client or server. In the other side the messages will be unwrapped. According to tcp/ip protocol, one connection is like a river, messages are like the water. When sending two messages, the messages will arrived one by one or together or part of them. Therefore we cannot point where is the border of one message. We must use one mechanism to control it. Using protocol is effective mean. When we receive some data and put into the buffer, we could then check if the buffer contains a complete message according to the protocol. If not wait until the data in buffer contains one message or more. Then we extract the message from the buffer.

* **IO Procotol**

In this procotol, we define the packet. We usually call it message anyway. The packet would contain header&body. Below chart is one packet structure. From the body length, we could know the length of the body. Therefore, we know the border of the packet. The Header Start is the packet start, the max length body character is the packet end. Class IOProtocol Define this protocol, we could decrease the number of characters in the Header Start & Header End in order to improve the transmission performance. But it may cause some unexpected issue if the body contains the same characters. Anyway, special means could be used here. Please note that the Body Length part is also fixed length which means if you have body length of 7 then the Body Length part could be 0007. The premises is the fixed length of body length is 4. We will fill the blanks with 0.

Example: $#####$0007\*#####\*MESSAGE

Header Start

Header End

Body Length

Body

* Serialization/Deserialization

Of course all of the request will be serialized before sent though the net. We need to define an unified Serialization&Deserialization protocol. For simple use, we will use xml to define the request xml and the response xml. This will occupy more bandwidth, however xml is an extendable format. We could change it later if the performance become a headache.

Below is the request & response xml format.

<request>

<requestid>10000</requestid>

<serviceName>&lt;testServiceNmae&gt;</serviceName>

<methodName>test23%^Method&amp;</methodName>

<version>@@!#$test.1.0</version>

<group>testGroup</group>

<list>

<arg>arg1</arg><arg>arg2&amp;\*^%</arg><arg>arg3</arg>

</list>

</request>

<response>

<requestid>100001212</requestid>

<result>sdsjdlfkj$@^!\*#!4457@$$</result>

</response>

* Service Access&Service Provider

In this layer, we provide the service access and service provider. Client would setup information of how to access the service, the server side setup the service. We conclude the calling and being called as procotol.

* Service Access

# Configure the client

client.service5.id=calculator

client.service5.name=calculator

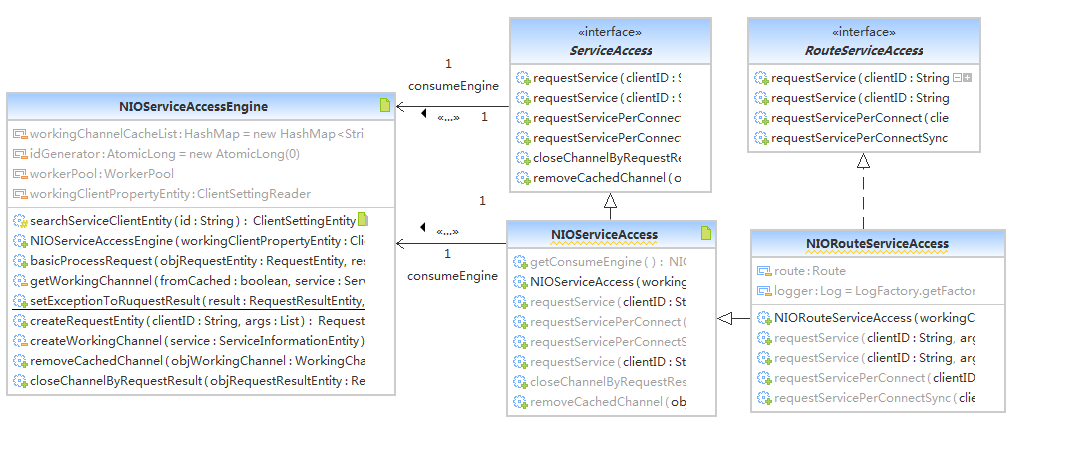
client.service5.method=add

client.service5.version=version

client.service5.group=test

Above is the client setting for the service. User could use the id to request the service. Below chart shows the class diagram, We define an engine for the nio access. In this engine we define the basic request function for the nio request. In this engine we would optionally establish the channel by caching or not. And close the channel after the request optionally.

We could also define other type of engine like bio, http as well. But the access class is required to extends the same interface(ServiceAccess, RouteServiceAccess). RouteServiceAccess is use for the Linkage center. We would talk it later on the Linkage Center paragraph.



* Service Provider

# Configure the service for the server

service.service2.name=calculator

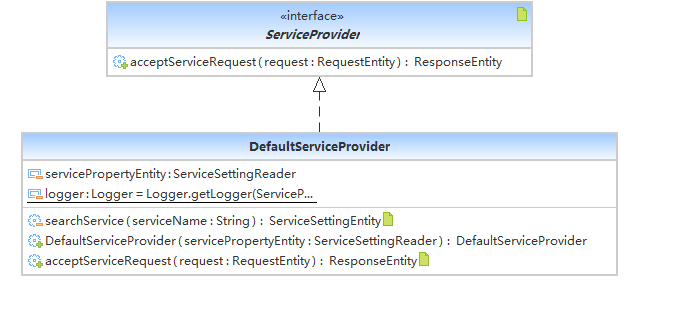
service.service2.interface=test.service.Calculator

service.service2.version=version

service.service2.group=test

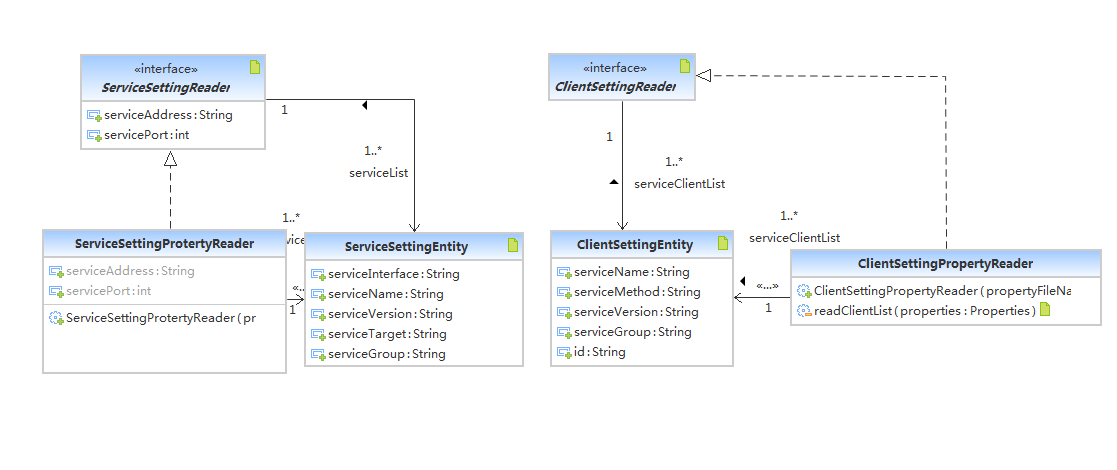
service.service2.target=test.service.CalculatorImpl

Above is the setting for service in the server. Below is the class diagram. It's easy to understand.



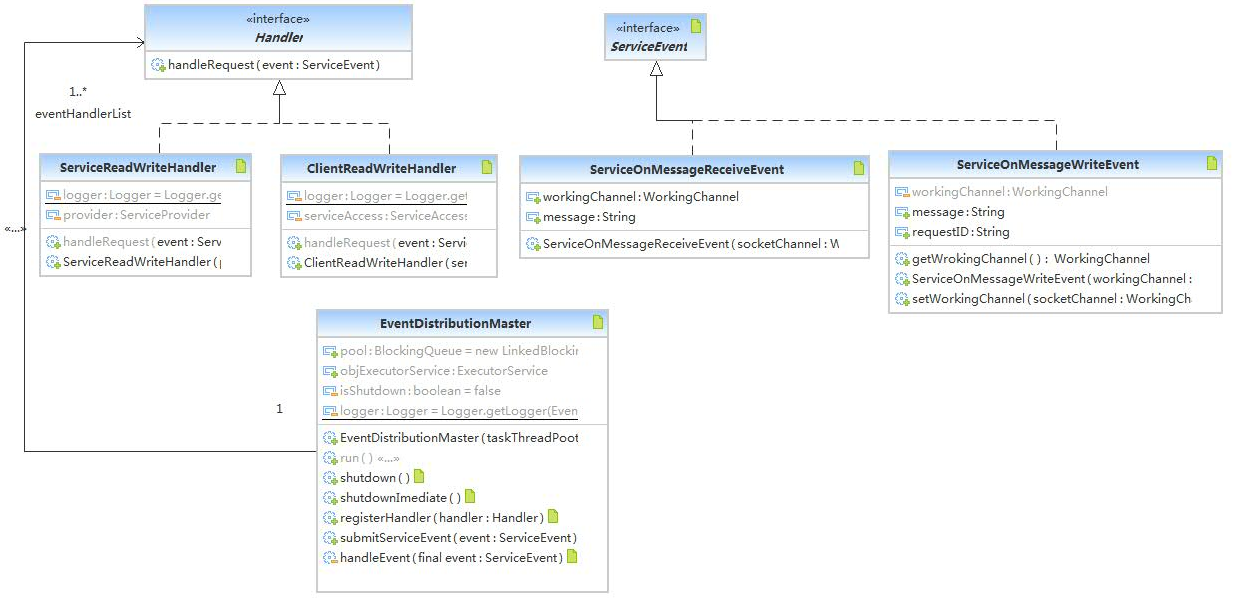
* Service Setting Reader

Client and Service Setting Reader will be responsible for reading the client and service setting. The setting could be read from different ways. We now define a property read which could read from the property file. We could also read from xml or text file.



* Event Driven Model

In the IO layer, when the message received, IO layer will forward the events to the master. IO layer won't care the event handling. It only puts the events into the event pool in the master. There are threads running background which will deal with the events and submit the job into thread pool. The events are handling by the handlers which are registered to the master when start. Threads running background will fetch the event from the event pool then submit a job into the thread pool. The job is looping all of the handlers and call the handleRequest method by passing event as parameter. By using this model, concurrence will be improved a lot. Messages receiving/sending and dealing with the message are separated.



* Linkage Center

Sometimes, there are services cluster. However we will pick up one service from the service list. When the service is available, the services will be regitered to the Linkage Center. When the service is off, the service will be unregistered then. Client does not aware the service exists or not. It will get the service list from the Linkage Center and choose one the use.

Linkage Center

Service

Service

Register/Unregister

Client

Fetch Service List

Service

* Future Plan
* Startup

This middleware can be started from different ways. Different way can have different problems.

1. Web Startup

We could setup a servlet and init the servlet with the linkage starting when start.

1. Server Startup

In this way, the middleware is starting together with the server middleware like weblogic. We may need to define a new class loader to avoid the classes confliction.

* Cloud Storage

Doing a file storage system by using linkage like hadoop.

* Test

Cover all of the module by junit test case.

* Performance Index

Define more performance indexs for linkage.