**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 part, 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 record into the database.

But how to call a distributed service ? Is there a need that we establish a protocol for each service calling. Answer is definitely not, we need to design a unified middleware 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. See the 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 wrapper/unwrapper message with the io protocol. The serialization/deserialization layer placed above the wrapper layer, it will use 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

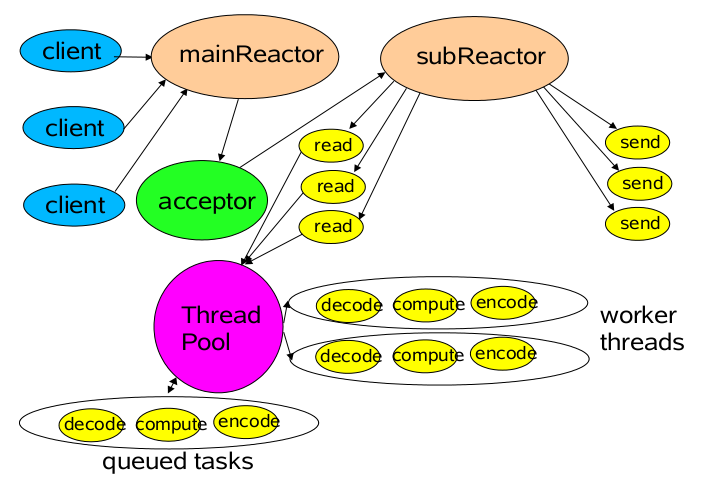
Client Side

Service Side

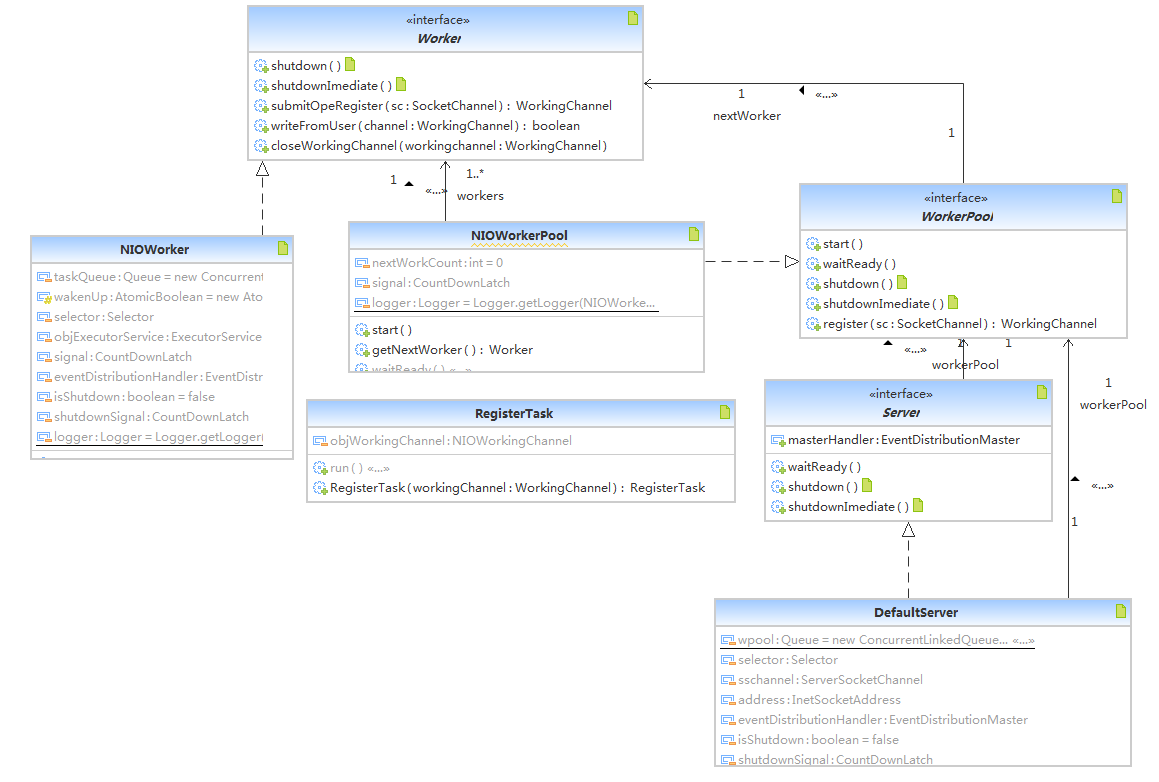
Linkage

* Communication Layer

Below chart shows the classic NIO model. There is one main reactor accept all connection then the acceptor will post the connect 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 too 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. The 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 message will be unwrapped for using. For the tcp/ip protocol, one connection is like a river, the messages are 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 one effective mean. When we receive some data, we could check if it is a complete message according to the protocol. If not we need to put the data to the buffer, then 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 packer 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 also be used here.

Header Start

Header End

Body Length

Body

* [Performance](javascript:void(0);) [index](javascript:void(0);)

Concurrence and presure test