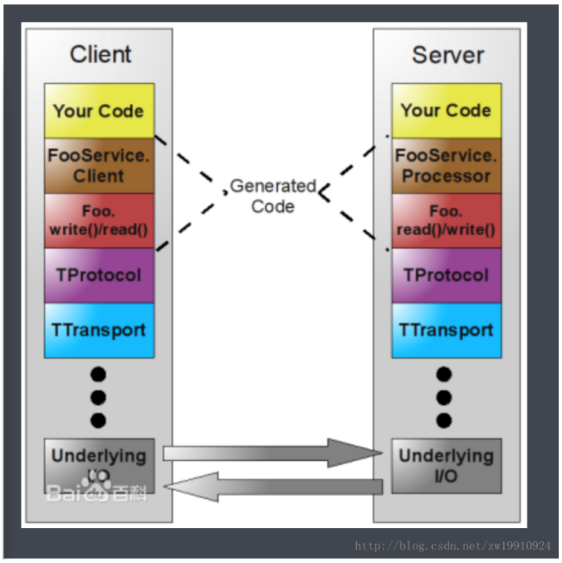
下面是thrift的客户端和服务端交互的一个原理图：



如上图：客户端在进行远程方法调用时，首先是通过Thrift的编译器生成的客户端，将调用信息（方法名，参数信息）以指定的协议进行封装，而传输层TTransport是对协议层的封装进行处理（比如封装成帧frame），并通过网络发送出去。服务端这边流程跟客户端相反，收到客户端发过来的数据后，首先经过传输层对传过来的数据进行处理，然后使用特定协议（跟客户端一一对应的）进行解析，然后再通过生成的processor调用用户编写的代码，如果有返回值的话，返回值以逆向的顺序，即通过协议层封装，然后传输层处理对数据进行发送给，到了客户端那边就是对服务端返回的数据进行处理，使用特定协议进行解析，然后得到一个调用结果

Thrift的传输格式：

thrift之所以被称为一种高效的RPC框架，其中一个重要的原因就是它提供了高效的数据传输。

以下是Thrift的传输格式种类：

- TBinaryProtocol：二进制格式。效率显然高于文本格式
- TCompactPtotocol: 压缩格式。在二进制基础上进一步压缩。
- TJSONProtocol：JSON格式
- TSimpleJSONProtocol：提供Json只写格式，生成的文件很容易用脚本语言解析。
- TDebugProtocol：使用易懂的刻度文本格式，以便于调试。

以上可以看到，在线上环境，使用TCompactProtocol格式效率最高的，同等数据传输占用网络带宽是最少的。

Thrift的数据传输方式（传输层）

- TSocket：阻塞式socket
- TFramedTransport：以frame为单位进行传输，非阻塞式服务中使用
- TFileTransport：以文件形式进行传输
- TMemoryTransport：将内存用于I/O。
- TZlibTransport：使用zlib进行压缩，与其他传输方式联合使用。

Thrift的服务模型

TSimpleServer

简单的单线程服务模型，常用于测试。只在一个单独的线程中以阻塞I/O的方式来提供服务。所以它只能服务一个客户端连接，其他所有客户端在被服务器端接受之前都只能等待。

TNonblockingServer

它使用了非阻塞式I/O，使用了java.nio.channels.Selector，通过调用select()，它使得程序阻塞在多个连接上，而不是单一的一个连接上。TNonblockingServer处理这些连接的时候，要么接受它，要么从它那读数据，要么把数据写到它那里，然后再次调用select()来等待下一个准备好的可用的连接。通用这种方式，server可同时服务多个客户端，而不会出现一个客户端把其他客户端全部“饿死”的情况。缺点是所有消息是被调用select()方法的同一个线程处理的，服务端同一时间只会处理一个消息，并没有实现并行处理。

THsHaServer（半同步半异步server）

针对TNonblockingServer存在的问题，THsHaServer应运而生。它使用一个单独的线程专门负责I/O，同样使用java.nio.channels.Selector，通过调用select()。然后再利用一个独立的worker线程池来处理消息。只要有空闲的worker线程，消息就会被立即处理，因此多条消息能被并行处理。效率进一步得到了提高。

TThreadedSelectorServer

它与THsHaServer的主要区别在于，TThreadedSelectorServer允许你用多个线程来处理网络I/O。它维护了两个线程池，一个用来处理网络I/O，另一个用来进行请求的处理。

TThreadPoolServer

它使用的是一种多线程服务模型，使用标准的阻塞式I/O。它会使用一个单独的线程来接收连接。一旦接受了一个连接，它就会被放入ThreadPoolExecutor中的一个worker线程里处理。worker线程被绑定到特定的客户端连接上，直到它关闭。一旦连接关闭，该worker线程就又回到了线程池中。

这意味着，如果有1万个并发的客户端连接，你就需要运行1万个线程。所以它对系统资源的消耗不像其他类型的server一样那么“友好”。此外，如果客户端数量超过了线程池中的最

大线程数，在有一个worker线程可用之前，请求将被一直阻塞在那里。
如果提前知道了将要连接到服务器上的客户端数量，并且不介意运行大量线程的话，TThreadPoolServer可能是个很好的选择

总结

Thrift是一个跨语言的RPC框架，如果有跨语言交互的业务场景，Thrift可能是一个很好的选择。如果使用恰当，thrift将是一个非常高效的一个RPC框架。开发时应根据具体场景选择合适的协议，传输方式以及服务模型。缺点就是Thrift并没有像dubbo那样提供分布式服务的支持，如果要支持分布式，需要开发者自己去开发集成。