# 编解码技术

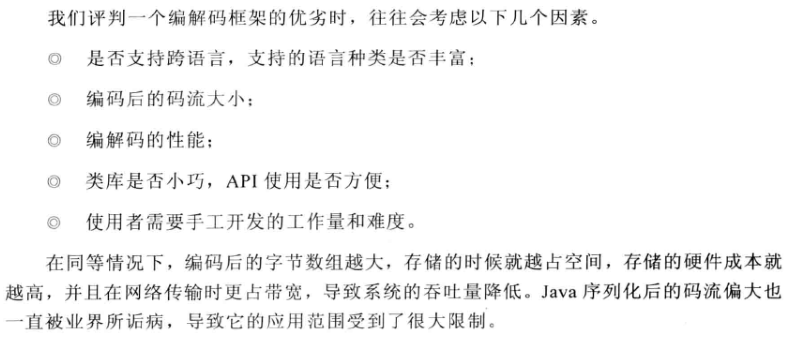
基于java提供的对象输入/输出流ObjectInputStream和ObjectOutputStream，可以直接把Java对象作为可存储的字节数组写入文件，也可以传输到网络上。

基于JDK默认的序列化机制可避免操作底层的字节数组，从而提升开发效率。

Java序列化的目的：

* 网络传输
* 对象持久化

当进行远程跨进程服务调用时，需要把被传输的java对象编码为字节数组或者ByteBuffer对象。当远程服务读取到ByteBuffer或字节数组时，需要将其解码为发送时的Java对象。这被称为java对象的编解码技术。



## Java序列化的缺点

Java序列化： 实现java.io.Serializable并生成序列ID。内部私有协议。

1. 无法跨语言

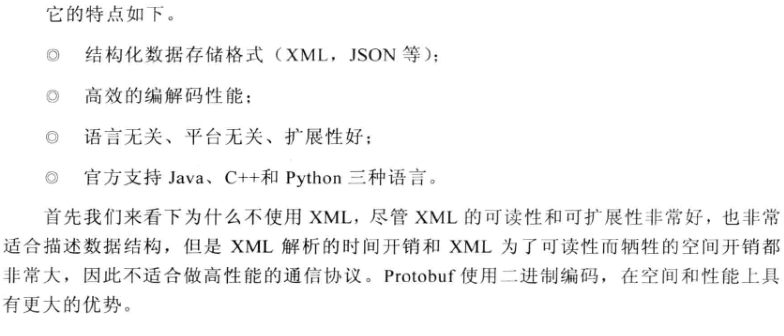
对于跨进程服务调用，服务提供者可能使用其他语言。需要和异构语言进程交互时，java序列化难以胜任，其他语言无法反序列化。

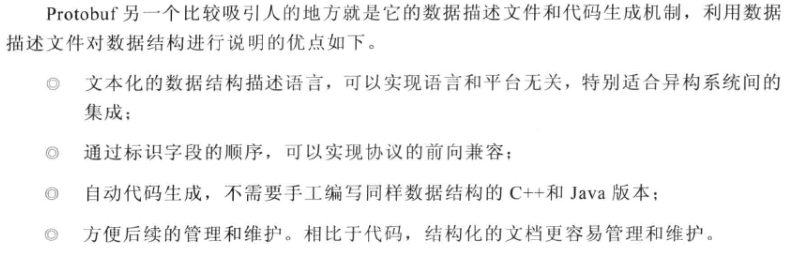
1. 序列化后的码流太大
2. 序列化性能太低

## 业界主流的编解码框架

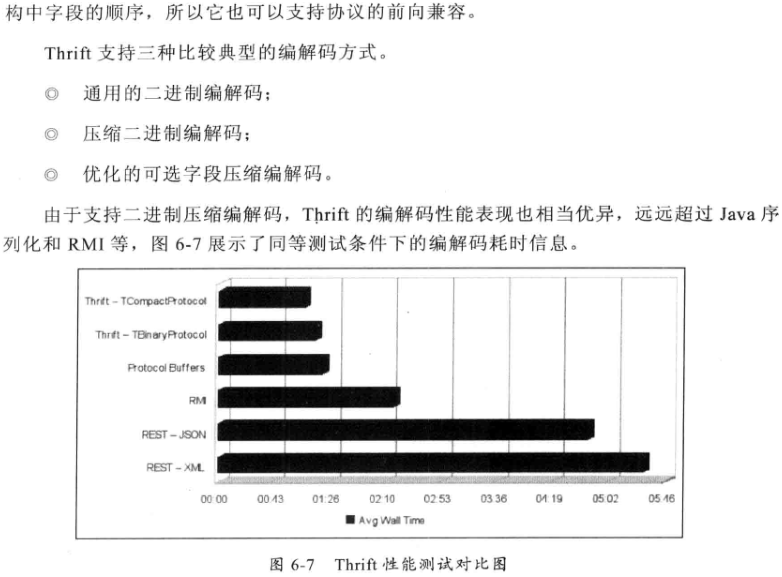
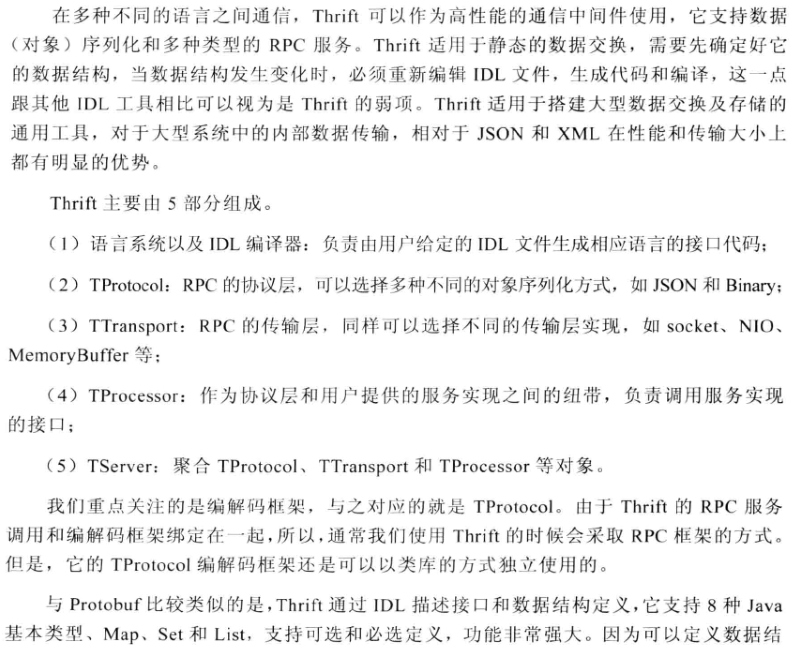
1. Google的Protobuf：（Google Protocol Buffers）

它将数据结构以.proto文件进行描述，通过代码生成共工具可生成对应数据结构的POJO对象和Protobuf相关的方法和属性。

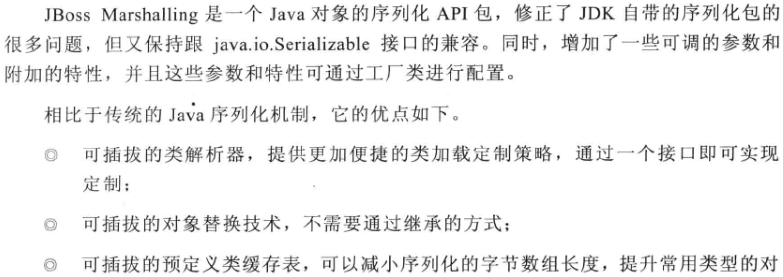


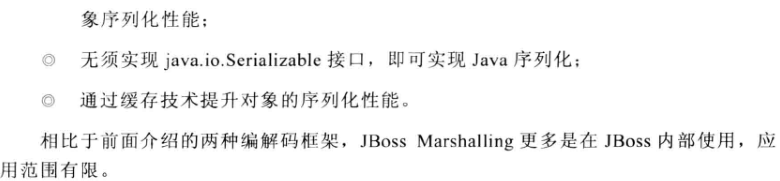


1. Facebook的Thrift



1. JBoss Marshalling





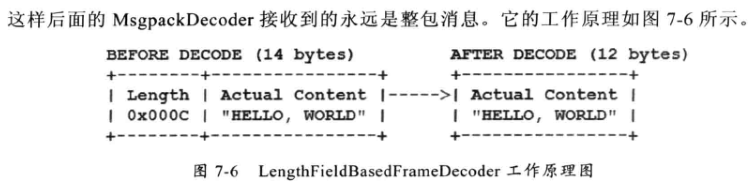
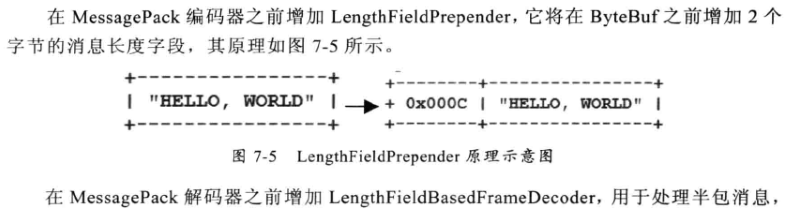
# MessagePack编解码



特点：

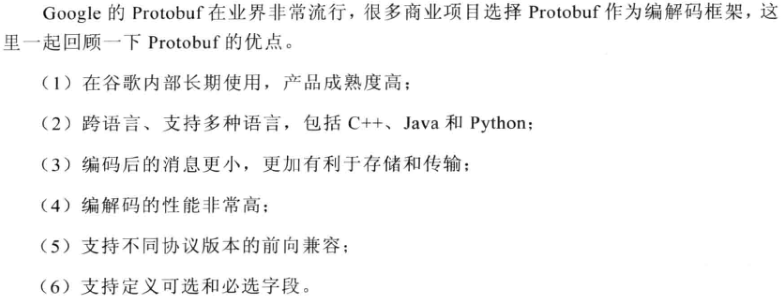
* 编解码高效，性能高
* 序列化之后码流小
* 支持跨语言

**LengthFieldPrepender和lengthFieldBasedFrameDecoder:解决TCP粘包半包问题**



对象需加@Message

# Google Protobuf编解码



//打印时会将对象中属性为0的字段抛弃。

## Protobuf使用注意事项：

ProtobufDecoder仅负责解码，不支持读半包。因此，在ProtobufDecoder前，一定要有能够处理读半包的解码器，有三种可选：

1. 使用Netty提供的ProtobufVarint32FrameDecoder,他可以处理半包消息；
2. 继承Netty提供的通用半包解码器LengthFieldBasedFrameDecoder;
3. 继承ByteToMessageDecoder类，自己处理半包消息。

# JBoss Marshalling编解码

# 

注意：需要引入jboss-marshalling 和 jboss-marshalling-serial两个包. POJO实现Serializable;

netty的marshalling编解码器支持半包和粘包的处理。