04 序列化介绍以及序列化协议选择

04 序列化介绍以及序列化协议选择

序列化和反序列化相关概念

什么是序列化?什么是反序列化?

如果我们需要持久化Java对象比如将Java对象保存在文件中，或者在网络传输Java对象，这些场景都需要用到序列化。

简单来说：

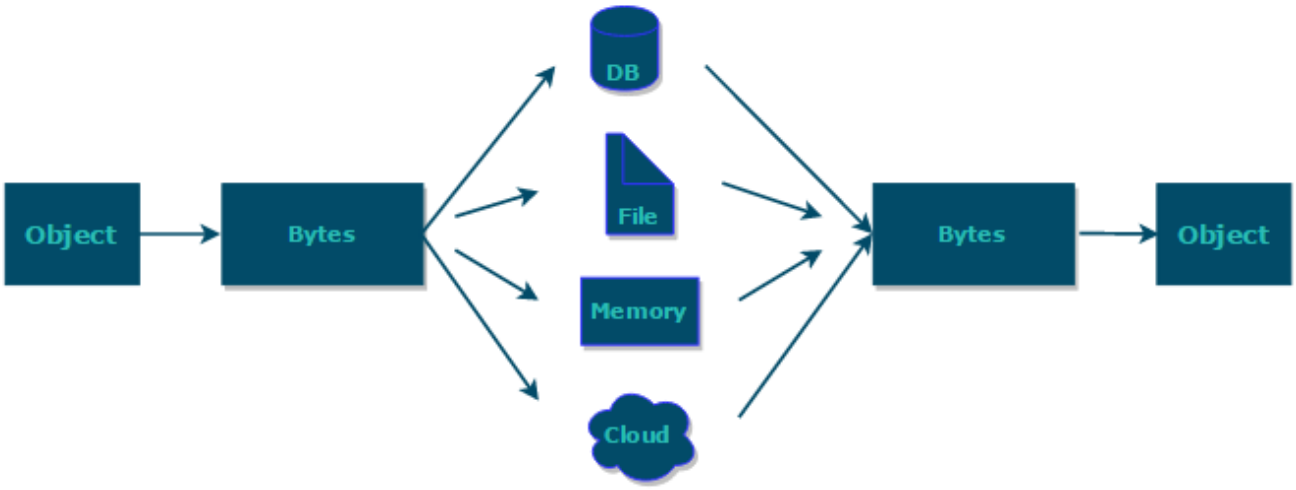
- **序列化**：将数据结构或对象转换成二进制字节流的过程
- **反序列化**：将在序列化过程中所生成的二进制字节流的过程转换成数据结构或者对象的过程

对于Java这种面向对象编程语言来说，我们序列化的都是对象（Object）也就是实例化后的类(Class)，但是在C++这种半面向对象的语言中，struct(结构体)定义的是数据结构类型，而class对应的是对象类型。

维基百科是如是介绍序列化的：

> **序列化**（serialization）在计算机科学的数据处理中，是指将数据结构或对象状态转换成可取用格式（例如存成文件，存于缓冲，或经由网络中发送），以留待后续在相同或另一台计算机环境中，能恢复原先状态的过程。依照序列化格式重新获取字节的结果时，可以利用它来产生与原始对象相同语义的副本。对于许多对象，像是使用大量引用的复杂对象，这种序列化重建的过程并不容易。面向对象中的对象序列化，并不概括之前原始对象所关系的函数。这种过程也称为对象编组（marshalling）。从一系列字节提取数据结构的反向操作，是反序列化（也称为解编组、deserialization、unmarshalling）。

综上：**序列化的主要目的是通过网络传输对象或者说是将对象存储到文件系统、数据库、内存中**。



https://www.corejavaguru.com/java/serialization/interview-questions-1
<https://www.corejavaguru.com/java/serialization/interview-questions-1>

实际开发中有哪些用到序列化和反序列化的场景

1. 对象在进行网络传输（比如远程方法调用RPC的时候）之前需要先被序列化，接收到序列化的对象之后需要再进行反序列化；
2. 将对象存储到文件中的时候需要进行序列化，将对象从文件中读取出来需要进行反序列化。
3. 将对象存储到缓存数据库（如 Redis）时需要用到序列化，将对象从缓存数据库中读取出来需要反序列化。

序列化协议对应于TCP/IP 4层模型的哪一层?

我们知道网络通信的双方必须要采用和遵守相同的协议。TCP/IP 四层模型是下面这样的，序列化协议属于哪一层呢?

1. 应用层
2. 传输层
3. 网络层
4. 网络接口层

如上图所示，OSI七层协议模型中，表示层做的事情主要就是对应用层的用户数据进行处理转换为二进制流。反过来的话，就是将二进制流转换成应用层的用户数据。这不就对应的是序列化和反序列化么？

因为，OSI七层协议模型中的应用层、表示层和会话层对应的都是TCP/IP 四层模型中的应用层，所以序列化协议属于TCP/IP协议应用层的一部分。

常见序列化协议对比

JDK自带的序列化方式一般不会用 ，因为序列化效率低并且部分版本有安全漏洞。比较常用的序列化协议有 hessian、kyro、protostuff。

下面提到的都是基于二进制的序列化协议，像 JSON 和 XML这种属于文本类序列化方式。虽然 JSON 和 XML可读性比较好，但是性能较差，一般不会选择。

JDK自带的序列化方式

JDK 自带的序列化，只需实现 `java.io.Serializable` 接口即可。

```java
@AllArgsConstructor
@NoArgsConstructor
@Getter
@Builder
@ToString
public class RpcRequest implements Serializable {
    private static final long serialVersionUID = 1905122041950251207L;
    private String requestId;
    private String interfaceName;
    private String methodName;
    private Object[] parameters;
    private Class<?>[] paramTypes;
    private RpcMessageTypeEnum rpcMessageTypeEnum;
}
```

> 序列化号 serialVersionUID 属于版本控制的作用。序列化的时候serialVersionUID也会被写入二级制序列，当反序列化时会检查serialVersionUID是否和当前类的serialVersionUID一致。如果serialVersionUID不一致则会抛出 `InvalidClassException` 异常。强烈推荐每个序列化类都手动指定其 `serialVersionUID` ，如果不手动指定，那么编译器会动态生成默认的序列化号

我们很少或者说几乎不会直接使用这个序列化方式，主要原因有两个：

1. **不支持跨语言调用** : 如果调用的是其他语言开发的服务的时候就不支持了。
2. **性能差** ： 相比于其他序列化框架性能更低，主要原因是序列化之后的字节数组体积较大，导致传输成本加大。

Kryo

Kryo是一个高性能的序列化/反序列化工具，由于其变长存储特性并使用了字节码生成机制，拥有较高的运行速度和较小的字节码体积。

另外，Kryo 已经是一种非常成熟的序列化实现了，已经在Twitter、Groupon、Yahoo以及多个著名开源项目（如Hive、Storm）中广泛的使用。

guide-rpc-framework <https://github.com/Snailclimb/guide-rpc-framework> 就是使用的 kyro 进行序列化，序列化和反序列化相关的代码如下： 、

Java | Copy

```java
/**
 * Kryo serialization class, Kryo serialization efficiency is very high
 *
 * @author shuang.kou
 * @createTime 2020年05月13日 19:29:00
 */
@Slf4j
public class KryoSerializer implements Serializer {

    /**
     * Because Kryo is not thread safe. So, use ThreadLocal to store Kry
     */
    private final ThreadLocal<Kryo> kryoThreadLocal = ThreadLocal.withIn
        Kryo kryo = new Kryo();
        kryo.register(RpcResponse.class);
        kryo.register(RpcRequest.class);
        return kryo;
    });

    @Override
    public byte[] serialize(Object obj) {
        try (ByteArrayOutputStream byteArrayOutputStream = new ByteArray
             Output output = new Output(byteArrayOutputStream)) {
            Kryo kryo = kryoThreadLocal.get();
            // Object->byte:将对象序列化为byte数组
            kryo.writeObject(output, obj);
            kryoThreadLocal.remove();
            return output.toBytes();
        } catch (Exception e) {
            throw new SerializeException("Serialization failed");
        }
    }

    @Override
    public <T> T deserialize(byte[] bytes, Class<T> clazz) {
        try (ByteArrayInputStream byteArrayInputStream = new ByteArrayIn
             Input input = new Input(byteArrayInputStream)) {
            Kryo kryo = kryoThreadLocal.get();
            // byte->Object:从byte数组中反序列化出对对象
            Object o = kryo.readObject(input, clazz);
            kryoThreadLocal.remove();
            return clazz.cast(o);
        } catch (Exception e) {
            throw new SerializeException("Deserialization failed");
        }
    }

}
```

Github 地址：https://github.com/EsotericSoftware/kryo <https://github.com/EsotericSoftware/kryo> 。

Protobuf

Protobuf出自于Google，性能还比较优秀，也支持多种语言，同时还是跨平台的。就是在使用中过于繁琐，因为你需要自己定义 IDL 文件和生成对应的序列化代码。这样虽然不然灵活，但是，另一方面导致protobuf没有序列化漏洞的风险。

> Protobuf包含序列化格式的定义、各种语言的库以及一个IDL编译器。正常情况下你需要定义proto文件，然后使用IDL编译器编译成你需要的语言

一个简单的 proto 文件如下：

```protobuf
// protobuf的版本
syntax = "proto3";
// SearchRequest会被编译成不同的编程语言的相应对象，比如Java中的class、Go中的str
message Person {
  //string类型字段
  string name = 1;
  // int 类型字段
  int32 age = 2;
}
```

Github地址：https://github.com/protocolbuffers/protobuf <https://github.com/protocolbuffers/protobuf> 。

ProtoStuff

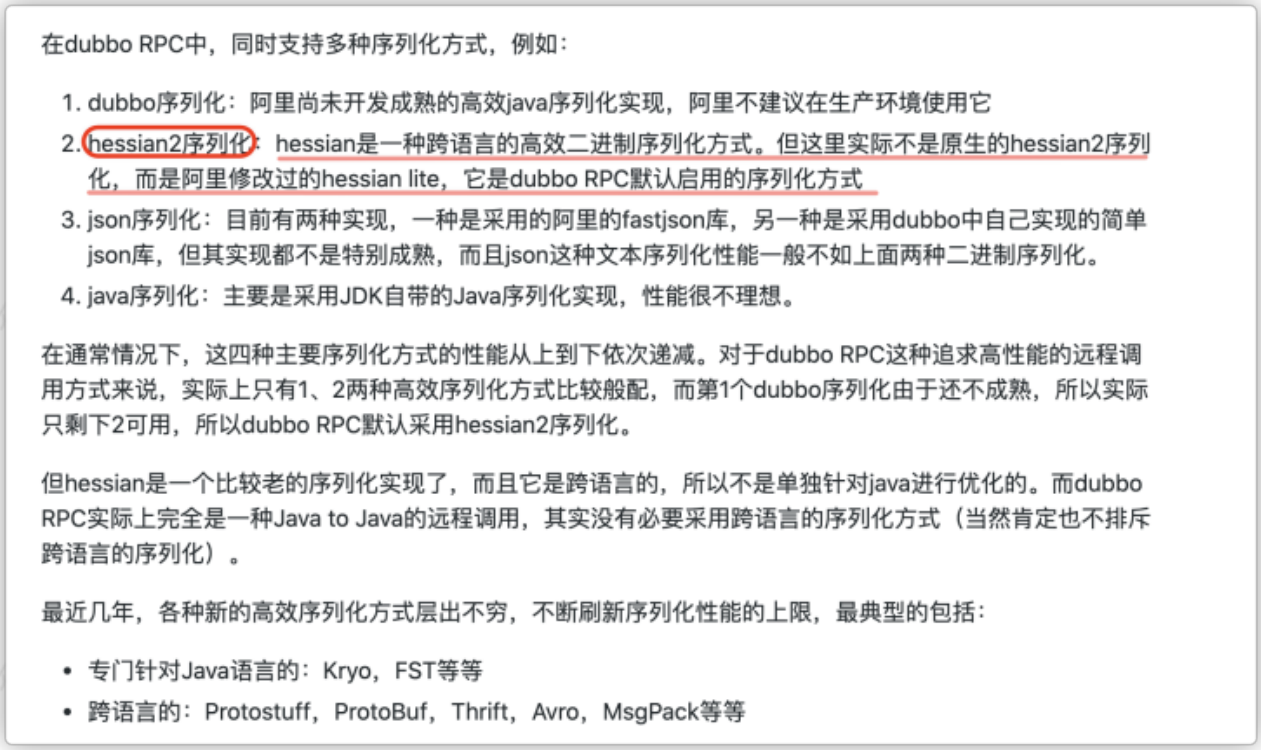
由于Protobuf的易用性，它的哥哥 Protostuff 诞生了。

protostuff 基于Google protobuf，但是提供了更多的功能和更简易的用法。虽然更加易用，但是不代表 ProtoStuff 性能更差。

Gihub地址：https://github.com/protostuff/protostuff <https://github.com/protostuff/protostuff> 。

hession

hessian 是一个轻量级的,自定义描述的二进制RPC协议。hessian是一个比较老的序列化实现了，并且同样也是跨语言的。



在dubbo RPC中，同时支持多种序列化方式，例如：

1. dubbo序列化：阿里尚未开发成熟的高效java序列化实现，阿里不建议在生产环境使用它
2. hessian2序列化：hessian是一种跨语言的高效二进制序列化方式。但这里实际不是原生的hessian2序列化，而是阿里修改过的hessian lite，它是dubbo RPC默认启用的序列化方式
3. json序列化：目前有两种实现，一种是采用的阿里的fastjson库，另一种是采用dubbo中自己实现的简单json库，但其实现都不是特别成熟，而且json这种文本序列化性能一般不如上面两种二进制序列化。
4. java序列化：主要是采用JDK自带的Java序列化实现，性能很不理想。

在通常情况下，这四种主要序列化方式的性能从上到下依次递减。对于dubbo RPC这种追求高性能的远程调用方式来说，实际上只有1、2两种高效序列化方式比较般配，而第1个dubbo序列化由于还不成熟，所以实际只剩下2可用，所以dubbo RPC默认采用hessian2序列化。

但hessian是一个比较老的序列化实现了，而且它是跨语言的，所以不是单独针对java进行优化的。而dubbo RPC实际上完全是一种Java to Java的远程调用，其实没有必要采用跨语言的序列化方式（当然肯定也不排斥跨语言的序列化）。

最近几年，各种新的高效序列化方式层出不穷，不断刷新序列化性能的上限，最典型的包括：

- 专门针对Java语言的：Kryo，FST等等
- 跨语言的：Protostuff，ProtoBuf，Thrift，Avro，MsgPack等等

dubbo RPC默认启用的序列化方式是 hession2 ,但是，Dubbo对hessian2进行了修改，不过大体结构还是差不多。

总结

Kryo 是专门针对Java语言序列化方式并且性能非常好，如果你的应用是专门针对Java语言的话可以考虑使用，并且 Dubbo 官网的一篇文章中提到说推荐使用 Kryo 作为生产环境的序列化方式。(文章地址：https://dubbo.apache.org/zh/docs/v2.7/user/references/protocol/rest/ <https://dubbo.apache.org/zh/docs/v2.7/user/references/protocol/rest/>)

有鉴于此，我们为dubbo引入Kryo和FST这两种高效Java序列化实现，来逐步取代hessian2。

其中，Kryo是一种非常成熟的序列化实现，已经在Twitter、Groupon、Yahoo以及多个著名开源项目（如Hive、Storm）中广泛的使用。而FST是一种较新的序列化实现，目前还缺乏足够多的成熟使用案例，但我认为它还是非常有前途的。

在面向生产环境的应用中，我建议目前更优先选择Kryo。

像Protobuf、 ProtoStuff、hession这类都是跨语言的序列化方式，如果有跨语言需求的话可以考虑使用。

除了我上面介绍到的序列化方式的话，还有像 Thrift，Avro 这些。

其他推荐阅读

1. 美团技术团队–序列化和反序列化：https://tech.meituan.com/2015/02/26/serialization-vs-deserialization.html <https://tech.meituan.com/2015/02/26/serialization-vs-deserialization.html>
2. 在Dubbo中使用高效的Java序列化（Kryo和FST）: https://dubbo.apache.org/zh-cn/docs/user/serialization.html <https://dubbo.apache.org/zh-cn/docs/user/serialization.html>