

**本科生毕业设计[论文]**

**基于Java语言的序列化框架开发**

院 系 计算机科学与技术

专业班级 2013级卓越工程师班

姓 名 胡振宇

学 号 U201315035

指导教师 石宣化

2017 年 5 月 17 日

**学位论文原创性声明**

本人郑重声明：所呈交的论文是本人在导师的指导下独立进行研究所取得的研究成果。除了文中特别加以标注引用的内容外，本论文不包括任何其他个人或集体已经发表或撰写的成果作品。本人完全意识到本声明的法律后果由本人承担。

作者签名： 年 月 日

**学位论文版权使用授权书**

本学位论文作者完全了解学校有关保障、使用学位论文的规定，同意学校保留并向有关学位论文管理部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权省级优秀学士论文评选机构将本学位论文的全部或部分内容编入有关数据进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

本学位论文属于 1、保密囗，在 年解密后适用本授权书

2、不保密囗 。

（请在以上相应方框内打“√”）

作者签名： 年 月 日

导师签名： 年 月 日

# 摘 要

在大数据处理火热的今天，分布式系统在大数据的处理上起着至关重要的作用。分布式系统分布在网络计算机上，依靠消息通讯。因此很多时候需要通过把需要传递的对象序列化为特定的二进制码流，从而能通过网络传输发送到各个网络节点，使不同的节点能接收到任务数据。在Java领域，虽然JDK提供了一套方便灵活的序列化机制，但是因为其序列化后的码流太多，会侵占大量的网络带宽，以及其编解码的性能过低导致大多数的分布式系统都没有使用JDK标准序列化机制，而是采用优秀的开源的序列化框架或者自行研发高效率的序列化框架。

考虑到JDK标准序列化机制需要实现特定的接口，并且在序列化过程中写入了很多额外的信息以及使用了反射机制，导致在延展性和性能，码流大小上都表现的不尽如人意。针对以上几点，新序列化框架Fox作出了优化。Fox采用了序列化类和目标类分离的策略，每个序列化类都针对目标类高度定制，采用id来代替类名以减少冗余信息的写入，使用了缓存机制，在序列化属于同一类的目标对象时，重复利用序列化对象，并且取消了使用反射机制，直接使用规定的接口方法。为每个类都写相应的序列化类是一个很繁重的工作，为了使新的序列化的框架也拥有高灵活性，在Fox里集成了开源社区的Kryo序列化框架。实验结果表明Fox与JDK的标准序列化相比在性能显著的提升，并明显的缩小的码流大小,与手写的JDK序列化相比在性能上也有较大的提升。并且在集成Kryo序列化框架的过程中，并未对Kryo序列化框架的性能造成损耗。

**关键词**：Java；序列化；高度定制化；优化；Kryo

# Abstract

Today, big data processing is so hot, the distributed system plays a vital role in the processing of big data. Distributed systems are distributed over network computers and rely on message communication. At most moment, we need to serialize the objects that need to be transferred to a specific binary stream，so that they can be sent to each network node through network transmission and that different nodes can receive the task data。In the field of Java, while the JDK provides a convenient serialization mechanism, but the much serialized stream will occupy a large amount of network bandwidth and the bad performance. So, many distributed system use a good open source serialization framework instead of JDK’s own.

Taking into account the JDK’s default serialization mechanism comes with the need to implement a specific interface, during the serialization process writes a lot of extra information and using reflection mechanism, resulting in scalability, performance, stream size is not satisfactory. In view of the above, a new serialization framework named Fox made optimization. Fox takes the strategy that make target class and serialization class separation, each serialization class is highly customized for target class, uses caching mechanism to use serialization object repeatedly, uses id instead of class name to reduce the stream size, uses specific method to serialize and deserialize instead of using reflect mechanism. Writing a corresponding serialization class for each target class is a heavy task, and in order to make the new serialization framework also have high flexibility, the Kryo serialization framework is integrated in Fox. The experimental results show that the performance of Fox is significantly improved compared with the default serialization of JDK, and the size of the bitstream is obviously reduced. Compared with the handwritten JDK serialization, the performance is also greatly improved. And in the process of integrating the Kryo serialization framework, the performance of the Kryo serialization framework is not wasted

**Keywords:** Java, Serialization, Customized, Optimization, Kryo

目 录

[摘 要 1](#_Toc483060914)

[Abstract 2](#_Toc483060915)

[1 绪论 5](#_Toc483060916)

[1.1 课题研究的背景和意义 5](#_Toc483060917)

[1.2 国内外研究现状 6](#_Toc483060918)

[1.3 课题要求 8](#_Toc483060919)

[1.4 研究方案和主要内容 9](#_Toc483060920)

[1.5 课题来源 9](#_Toc483060921)

[2 Fox序列化框架工作原理设计 11](#_Toc483060922)

[2.1 功能拆分 11](#_Toc483060923)

[2.2 功能图解 11](#_Toc483060924)

[2.3 空对象的处理 12](#_Toc483060925)

[2.4 本章小结 13](#_Toc483060926)

[3 Fox序列化框架接口设计 13](#_Toc483060927)

[3.1 Fox序列化框架的接口设计 13](#_Toc483060928)

[3.2 Fox中重要的数据结构及其作用 17](#_Toc483060929)

[3.3 帧内预测模块的SSE优化分析 17](#_Toc483060930)

[3.4 反变换及重建模块的SSE优化分析 17](#_Toc483060931)

[3.5 In-loop Deblock（环路滤波）模块的SSE优化分析 17](#_Toc483060932)

[3.6 Image Padding（图像边界扩展）模块的SSE优化分析 18](#_Toc483060933)

[3.7 本章小结 18](#_Toc483060934)

[4 19](#_Toc483060935)

[4.1 帧内宏块级并行优化技术的可行性分析 19](#_Toc483060936)

[4.2 帧内宏块的多线程调度机制 19](#_Toc483060937)

[4.3 本章小结 19](#_Toc483060938)

[5 性能测试与分析 20](#_Toc483060939)

[5.1 xIVC解码器项目测试的环境准备 20](#_Toc483060940)

[5.2 xIVC解码器测试结果及数据分析 20](#_Toc483060941)

[5.3 xIVC解码器的实际开发应用 20](#_Toc483060942)

[5.4 本章小结 20](#_Toc483060943)

[6 总结及展望 21](#_Toc483060944)

[致谢 22](#_Toc483060945)

[参考文献 23](#_Toc483060946)

# 绪论

## 课题研究的背景和意义

在大数据处理火热的今天，分布式系统在大数据的处理上起着至关重要的作用。分布式系统分布在网络计算机上，依靠消息通讯。因此很多时候需要通过把需要传递的对象序列化为特定的二进制码流，从而能通过网络传输发送到各个网络节点，使不同的节点能接收到任务数据。分布式系统的定义参考：“A distributed system is one in which components located at networked computers communicate and coordinate their actions only by passing messages”。可以看出分布式系统定义的两个基本点：分布在网络计算机上和消息通信。很多的消息通讯都会用到序列化，因此序列化的效率是对性能影响的至关重要的因素。网络的拥塞和包文的大小是密切相关的。巨大的包文无疑会加剧网络的负担和拥堵，对于分布式系统的性能造成巨大的损失。

Java平台允许在内存中创建可复用的Java对象，但一般情况下，只有当JVM处于运行时，这些对象才可能存在，即，这些对象的生命周期不会比JVM的生命周期更长。但在现实应用中，就可能要求在JVM停止运行之后能够保存(持久化)指定的对象，并在将来重新读取被保存的对象。Java对象序列化就能够帮助我们实现该功能。

使用Java对象序列化，在保存对象时，会把其状态保存为一组字节，在未来，再将这些字节组装成对象。必须注意地是，对象序列化保存的是对象的"状态"，即它的成员变量。由此可知，对象序列化不会关注类中的静态变量。

除了在持久化对象时会用到对象序列化之外，当使用RMI(远程方法调用)，或在网络中传递对象时，都会用到对象序列化。Java序列化API为处理对象序列化提供了一个标准机制，该API简单易用

## 国内外研究现状

### 序列化简介

Java 串行化技术可以使你将一个对象的状态写入一个Byte 流里，并且可以从其它地方把该Byte 流里的数据读出来，重新构造一个相同的对象。这种机制允许你将对象通过网络进行传播，并可以随时把对象持久化到数据库、文件等系统里。Java的串行化机制是RMI、EJB等技术的技术基础。用途：利用对象的串行化实现保存应用程序的当前工作状态，下次再启动的时候将自动地恢复到上次执行的状态。

序列化就是一种用来处理对象流的机制，所谓对象流也就是将对象的内容进行流化。可以对流化后的对象进行读写操作，也可将流化后的对象传输于网络之间。序列化是为了解决在对对象流进行读写操作时所引发的问题。

JDK序列化的实现：将需要被序列化的类实现Serializable接口，然后使用一个输出流(如：FileOutputStream)来构造一个ObjectOutputStream(对象流)对象，接着，使用ObjectOutputStream对象的writeObject(Object obj)方法就可以将参数为obj的对象写出(即保存其状态)，要恢复的话则用输入流

### JDK标准序列化特点

1. 如果某个类能够被串行化，其子类也可以被串行化。如果该类有父类，则分两种情况来考虑，如果该父类已经实现了可串行化接口。则其父类的相应字段及属性的处理和该类相同；如果该类的父类没有实现可串行化接口，则该类的父类所有的字段属性将不会串行化。
2. 声明为static和transient类型的成员数据不能被串行化。因为static代表类的状态， transient代表对象的临时数据；
3. 相关的类和接口：在java.io包中提供的涉及对象的串行化的类与接口有ObjectOutput接口、ObjectOutputStream类、ObjectInput接口、ObjectInputStream类。
   1. ObjectOutput接口：它继承DataOutput接口并且支持对象的串行化，其内的”writeObject()”方法实现存储一个对象。ObjectInput接口：它继承DataInput接口并且支持对象的串行化，其内的”readObject()”方法实现读取一个对象。
   2. ObjectOutputStream类：它继承OutputStream类并且实现ObjectOutput接口。利用该类来实现将对象存储(调用ObjectOutput接口中的”writeObject()”方法)。ObjectInputStream类：它继承InputStream类并且实现ObjectInput接口。利用该类来实现读取一个对象(调用ObjectInput接口中的”readObject()”方法)。

对于父类的处理，如果父类没有实现串行化接口，则其必须有默认的构造函数（即没有参数的构造函数）。否则编译的时候就会报错。在反串行化的时候，默认构造函数会被调用。但是若把父类标记为可以串行化，则在反串行化的时候，其默认构造函数不会被调用。因为Java 对串行化的对象进行反串行化的时候，直接从流里获取其对象数据来生成一个对象实例，而不是通过其构造函数来完成。

### JDK 标准序列化机制的不足

Java自己提供序列化而且用起来非常简单，但是在远程服务调用中很少用它，主要存在以下缺点：

1. 无法跨语言。由于java序列化是java内部私有的协议，其他语言不支持，导致别的语言无法反序列化。
2. 序列后的码流太大。
3. 序列化性能太低。
4. 要使用JDK序列化机制，必须让相应的实现规定的接口，丧失了灵活性，不利于开源库的使用。

### 目前流行的序列化框架及其特点

出于对序列化性能，网络传输，跨语言等各方面的考虑，开源社区上出现了许多优秀的序列化框架，其中使用比较的广发的几种

1. Google的Protobuf，Protobuf是Google开源的项目，全称 Google Protocol Buffers.特点：
   1. 结构化数据存储格式（Xml, Json等）
   2. 高性能编解码技术
   3. 语言和平台无关，扩展性好
   4. 支持JAVA, C++, Python三种语言。
2. FaceBook的Thrift
   1. Thrift支持多种语言
   2. Thrift适用于组建大型数据交换及存储工具，对于大型系统中的内部数据传输，相对于Json和Xml在性能上和传输大小上都有明显的优势
   3. Thrift支持三种比较典型的编码方式。（通用二进制编码，压缩二进制编码，优化的可选字段压缩编解码）
3. Kryo
   1. 速度快，序列化后体积小
   2. 支持的类全面
   3. 使用方便

## 课题要求

JDK的标准序列化机制需要实现特定的接口，并且在序列化过程中写入了很多额外的信息以及使用了反射机制，导致在延展性和性能，码流大小上都表现的不尽如人意，新框架Fox针对以上作出了改进。

具体目标如下：

1. 目标类和序列化类分离，即序列化和反序列化的方法不再包含于目标类内部，而全部继承于一个经过设计的抽象类。从而达到解耦合的目的。
2. 额外信息很大一部分来自于包名与类名的重复写入，使用整形id来映射包名与类名，这样可以节省大量的码流空间
3. 不再使用反射来探寻目标类中是否适用了特定的方法，而是约定序列化和反序列化的接口方法，直接调用，简化了序列化的流程。
4. 使用Cache缓存序列化对象，避免序列化对象的大量创建，减少JVM的垃圾回收。
5. 每个目标类的序列化类都是经过高度定制的，由编码者掌握具体序列化和反序列化细节。
6. 为了不丧失Fox的灵活性，集成Kryo，实现对通用类的序列化类。

## 研究方案和主要内容

在本次课题的研究中，根据对Fox的目标，首要的整个框架结构的设计。确定大体框架结构，明确每个类的功能，及其继承关系。明确需要的数据结构，以及在框架内的传递方式。最后是明确每个类的细节，并最终实现1.3章节中的课题要求

本文共分为五个章节：

第一章，通过分析分布式在处理大数据的过程中，引出对象序列化的重要性。简要介绍了JDK标准序列化机制的工作原理，并指出JDK的标准序列化机制的不足，提出优化目标，和对新框架Fox的期望

第二章，根据课题需要，提出对Fox的设计方案

第三章，根据设计方案，选择合理的数据结构

第四章，使用Fox，与Kryo和JDK的默认序列化机制，对相同类进行详细的客观测试，并通过分析性能和码流大小的数据来分析优化方案的缺陷以及优势。展示Fox的最终测试结果

第五章，总结整个过程中完成的主要内容，以及碰到的困难。以及对Fox的不足进行分析和未来展望。

## 课题来源

中兴飞流信息科技有限公司是专业从事创新大数据核心技术引擎研发和服务的高科技软件企业，是具有核心自主知识产权的大数据技术及解决方案提供商。

下面简称为飞流。大四期间在飞流公司Yita内核组担任实习生期间，在Yita原有的序列化框架基础上，自主设计融合Kryo序列化系统进入到Yita的手写序列化框架体系。在完成整个项目后，对Java序列化有了更深的认识，也认识到Java标准序列化的很多不足之处，所以Yita作为一个分布式流处理系统的才自己设计了一套自己的序列化框架。在研读Yita序列化相关代码后，我发现Yita序列化框架的实现并不是十分复杂，它充分利用了Java标准库中已经提供的很多底层类。Yita在序列化的设计是十分值得我学习，其中有不少亮点，如将被序列化的对象和负责序列化的类独立开来，不必像Java标准序列化框架一样必须实现某个接口才能实现序列化，这可以为开发人员使用第三方的库的时候带来很多的方便。同时序列化框架加入了Cache的概念，当某个所属某个类的对象已经被序列化过一次之后，会将用于序列化的对象缓存，以便下次序列化同类对象时，不必新建一个执行序列化的对象。这也催生了我参考Yita的设计思路，将整个序列化框架剥离，形成一套独立的序列化框架的想法。

# Fox序列化框架工作原理设计

课题的研究重点是在确定优化目标后，主要的是如何针对缺点设计框架模型，和选择数据结构。本章的重点在于对新框架Fox的设计解析，以及对工作模式的解释

## 功能拆分

1. 为保证序列化功能和目标类分离，需要满足可插拔的功能。因此，不应该由目标类去实现序列化的整个功能，而应该由专门的类负责。这里需要一个抽象类，去定义序列化和反序列化的标准方法，在该抽象类里需要有一个字段来记录该类服务于哪个目标类以及一些其他的功能将在后面功能细化中指出。
2. 实现的众多序列化类需要有一个保存序列化类实例的地方。Fox把这个调度功能交给注册类。注册类负责记录目前序列化系统已经存在的序列化类，提供注册功能，用户可通过注册方法，将实现好的序列化类记录。并提供查询功能，可根据目标类名查询到对应的序列化类。
3. Cache，用于缓存近期使用过的序列化类实例。在序列化和反序列化的过程中，系统先从Cache中查询是否有对应的序列化类实例，如果有则直接使用，如果没有则Cache类负责向注册类查询，若注册类未查询到结果则报错，否则Cache将查询到的实例缓存到自己的存储结构中，并由Cache返回查询到的实例。
4. 考虑到Java的输入流和输出流众多，为统一调度，和简化序列化过程，Fox中提供了一个包装性质的输出流和输入流。在输出流中，定义了写方法，负责将指定对象写入到输出流中。在输入流中，定义了读方法，负责从输入流中解析对象。在这两个流中都使用到了Cache。

## 功能图解

通过章节2.1的功能介绍，功能大致分为四个模块。下图是四个模块的功能及联系的示意图如图 2‑1所示



图 2‑1 功能示意图

首先是为目标类实现相应的序列化类，然后通过注册类提供的注册方法将序列化类的实例写入到注册类的数据结构中(该数据结构在第三章会有讨论,本章不做讨论)。用户通过Fox提供的输出流中提供的写方法将指定对象写入到输出流中，输出流先查询内部的Cache是否存在对应的序列化类实例，若没有，则从注册类中寻找，最终获取序列化类实例，并使用该实例中的序列化方法对用户提供的对象进行序列化。在反序列化的过程，与序列化的过程类似，在找到对应的序列化类实例后，通过实例中的反序列化方法进行反序列化。

## 空对象的处理

在序列化的过程，需要考虑到当用户提供的对象为null时的特殊情况。在Fox中，把空对象视为一个类，在Fox中该类为一个单例类。Fox内置了对单例类的序列化方法，在写入的过程中首先判断对象是否为null，若为null则直接将其替换为提到的单例对象，该单例对象在序列化时，会往输出流在一个字节，值为0，在反序列化时，会被反序列化为null。

## 本章小结

在本章中，主要介绍了Fox框架的整体设计思路，以及大体的工作流程。不涉及具体的方法和类。在下一章将会介绍Fox框架提供的用户接口和内部工作原理。

# Fox序列化框架详细设计

章节2介绍了大体的框架设计，本章将介绍Fox框架暴露给用户的接口以及内部使用的接口

## Fox序列化框架的接口设计

### 序列化类

该类是一个抽象类，所有目标类的序列化类都继承自该方法。在该类中定义了序列化和反序列化方法，并且记录了该序列化类对应的目标类，需要在构造序列化类实例的时候指服务的目标类名称以及记录了序列化实例在**注册类**中的位置

**序列化类Serialization接口如表格 3‑1**

表格 3‑1 Serialization接口表

|  |  |
| --- | --- |
| 方法／字段名 | 说明 |
| 用户可用 | |
| private final Class<?> serializationClass | 目标类名，即该类的服务目标名 |
| public void serialize(T object, SerializedOutput output) | 需要覆写的序列化方法 |
| public T deserialize(SerializedInput input) | 需要覆写的反序列化方法 |
| 内部使用 | |
| int cacheIndex | 序列化类在SerializationRegistry数据结构中储存的位置，用于代替映射类名 |
| public abstract Serialization<T> clone() | 序列化类实例自我复制方法 |
| final Serialization<T> clonedCopy() | 被SerializationRegistry调用，用户序列化实例的自我复制，及设置cacheIndex |

### 注册类

注册类用来储存用户通过注册类方法注册的序列化类实例，并且提供为Fox系统内部提供查询功能。该类对外只暴露注册接口。

**注册类SerializationRegistry接口如**

表格 3‑2 SerializationRegistry接口表

|  |  |
| --- | --- |
| 方法／字段名 | 说明 |
| 用户可用 | |
| public SerializationRegistry() | 构造方法，在构造方法默认注册了Integer、String、Float、Boolean、Byte、Char、Short、Long、Double、Void、NullReference |
| public synchronized <T> void registerOrigin(Serialization<T>serialization) | 用于向数据结构中写入用户自定义的序列化类，使用了同步锁，线程安全，在方法内部对Serialization中的cachIndex字段进行了设置 |
| public synchronized <T> void registerKryo(Class<T>serializationClass) | 用于向数据结构中写入需要使用Kryo序列化的类名，使用了同步锁，线程安全，在方法内部对Serialization中的cachIndex字段进行了设置 |
| public Kryo newKryo() | 根据已经的注册信息，对Kryo实例注册，并生成注册好的实例，更具体的在后续会有介绍 |
| 内部使用 | |
| public <T> Serialization<T> newSerialization(Class<T> targetClass) | 根据目标类名从数据结构中查找出对应的序列化实例 |
| Serialization<?> newSerialization(int index) | 根据cacheIndex从数据结构中查找出对应的序列化实例 |

### 缓存类

缓存类用来保存历史用过的序列化类实例，避免每次都由**注册类**生成新对象。Fox根据条件查询先从Cache中寻找对应的序列化类实例，若是在**缓存类**中未找到则从**注册类**中寻找，并加入到缓存中。

**Cache类 SerializationCache接口如表格 3‑3**

表格 3‑3 **SerializationCache接口**

|  |  |
| --- | --- |
| 方法／字段 | 说明 |
| 用户可用 | |
| public SerializationCache(SerializationRegistry serializationRegistry) | 构造方法，输入参数为注册类，SerializationCache将从serializationRegistry中获取信息 |
| 内部使用 | |
| public final <T> Serialization<T> getSerialization(final Class<T> targetClass) | 根据类名先从Cache中查找，如果查找不到，再从SerializationRegistry中查找，然后将结果缓存 |
| public final Serialization<?> getSerialization(final int id) | 根据cacheIndex从Cache中查找，如果查找不到，再从SerializationRegistry中查找，然后将结果缓存 |
| private final <T> void cache(final Serialization<?> serialization, final Class<T> targetClass) | 将从SerializationRegistry中查询到的结果缓存 |

### 输出流

输出流的包装器，在Fox中通过SerializedOutputStream的writeObject和wirte来从Cache中获取序列化类实例，并序列化对象到底层的outputStream中

输出流 SerializedOutputStream接口如**表格 3‑4**

表格 3‑4 **SerializedOutputStream接口**

|  |  |
| --- | --- |
| 方法／字段 | 说明 |
| 用户可用 | |
| public SerializedOutputStream(OutputStream out, SerializationCache serializationCache) | 构造方法，将Java提供的一些输出流和Cache类包装成Fox框架可用的输出流 |
| public SerializedOutputStream(OutputStream out, SerializationCache serializationCache, Kryo kryo) | 构造方法，将Java提供的一些输出流和Cache类以及初始化完成的Kryo包装成Fox框架可用的输出流，该方法在需要使用到Kryo序列化时使用 |
| public <T extends Object> void writeObject(T object) | 将对象写入到输出流，输出流中会写入cacheIndex，对应输入流中的readObject()方法 |
| public <T> void write(Object object, Class<T> clazz) | 将对象写入到输出流，输出流中将不会写入cacheIndex，对应输入流中的readObject(Class<T> objectClass)方法 |

### 输入流

输入流的包装器，Fox中通过SerializedInputStream中的readObject或者read方法从Cache中获取序列化类实例，并从从底层inputStream中反序列化出对应的目标对象。

**输入流 SerializedInputStream接口如表格 3‑5**

表格 3‑5 **SerializedInputStream**

|  |  |
| --- | --- |
| 方法／字段 | 说明 |
| 用户可用 | |
| public SerializedInputStream(InputStream in, SerializationCache serializationCache) | 构造方法，将Java提供的一些输入流和Cache类包装成Fox框架可用的输入流 |
| public SerializedInputStream(InputStream in, SerializationCache serializationCache, Kryo kryo) | 构造方法，将Java提供的一些输入流和Cache类以及初始化完成的Kryo包装成Fox框架可用的输入流，该方法在需要使用到Kryo反序列化时使用 |
| public <T> T read(Class<T> objectClass) | 从输入流中读取对象，输入参数为对象所属类名称。对应输出流中的write(Object object, Class<T> clazz) |
| public Object readObject() | 从输入流中读取对象，对应输出流中的writeObject(T object) |

## Fox中重要的数据结构及其作用

### 序列化类中的重要数据结构

**表格 3‑6**记录了在序列化类中有重要作用的字段，及其说明。

表格 3‑6序列化类中的重要数据结构

|  |  |
| --- | --- |
| 字段 | 说明 |
| private final Class<?> serializationClass | 记录序列化类服务的目标类名称，这是搜索从注册类搜索序列化类实例的根据 |
| int cacheIndex | 整形变量，记录序列化类实例在注册类中的列表中的下标，可依靠这个下表寻找序列序列化类实例 |

### 注册类中的重要数据结构

**表格 3‑7**记录了注册类中重要的数据结构，关于Kryo的数据结构在本章不做介绍，放在章节4中介绍

表格 3‑7 注册类中的重要数据结构

|  |  |
| --- | --- |
| 字段 | 说明 |
| private final ConcurrentHashMap<Class<?>, Serialization<?>>registry = new ConcurrentHashMap<Class<?>, Serialization<?>>() | registry 是一个HashMap，Key值是目标类名，Value值是序列化类实例 |
| private final CopyOnWriteArrayList<Serialization<?>> index | index 用于存储序列化类实例，实例在index中的下标将赋值给对应的cacheIdex |

### 缓存类中的重要数据结构

缓存类中，有自定义的GrowableObjectArray<E>，该类实现了List接口，主要功能是缓存从注册类中获取的序列化类实例，并按照实例中的cacheIdex中的值，放入到相同的位置中。

**表格 3‑8**记录了缓存类中的重要数据结构

表格 3‑8 缓存类中的重要数据结构

|  |  |
| --- | --- |
| 字段 | 说明 |
| final HashMap<Class<?>, Serialization<?>> serializations | Key值是目标类名，Value值是序列化类实例 |
| final GrowableObjectArray<Serialization<?>>  serializationIndex | 该类实现了List接口，主要功能是缓存从注册类中获取的序列化类实例，并按照实例中的cacheIdex中的值，放入到相同的位置中 |
| SerializationRegistry serializationRegistry | 该变量是对注册类的引用，当serializations中未找到对应序列化类实例时，缓存类将从注册类中去寻找并缓存至缓存类 |

### 输出流中的重要数据结构

**表格 3‑9**记录了输出流中的重要数据结构，有关Kryo的数据结构在章节4中介绍

表格 3‑9 输出流中的重要数据结构

|  |  |
| --- | --- |
| 字段 | 说明 |
| final SerializationCache serializationCache | 输出流从此serializationCache中获取实例，并序列化 |

### 输入流中的重要数据结构

**表格 3‑10**记录了输入流中的重要数据结构，有关Kryo的数据结构在章节4中介绍

表格 3‑10输入流中的重要数据结构

|  |  |
| --- | --- |
| 字段 | 说明 |
| final SerializationCache serializationCache | 输入流从此serializationCache中获取实例，并序列化 |

## 线程安全性

Fox的**注册类**做了线程安全保护，可以多线程同时注册和读取。但是**缓存类**，**输出流**和**输入流**都是线程独享的，在一个缓存类实例、输出流实例、输入流实例都应该只被一个线程操作，否则可能不可预知的错误。

## 本章小结

本章先介绍了Fox的接口设计，以及这些接口如何在互相配合下完成序列化与反序列化。然后详细介绍了类中的数据及其作用，最后简要介绍了Fox对于多线程的支持。在下章我们将要介绍如何融合Kryo框架

# Fox序列化框架融合Kryo

章节2和章节3介绍了Fox工作原理，以及如何自定义序列化类。自定义序列化类需要用户对序列化比较了解，虽然高效，但是不够灵活。Kryo是只用于Java的快速高效的对象图序列化框架，它的API简单易用，被用于众多知名的项目，如Storm，Spark等。Kryo可以自动序列化绝大多数类，无需用户去写序列化和反序列化方法，因此满足灵活使用的条件。

## 帧内宏块级并行优化技术的可行性分析

## 帧内宏块的多线程调度机制

### 帧内宏块并行解码的信号量设计

### 帧内宏块并行解码的进一步优化

图 ‑1 加入Deblock和Image Padding的宏块级并行优化示意图

## 本章小结

。

# 性能测试与分析

。

## xIVC解码器项目测试的环境准备

### 测试服务器和测试序列准备

### xIVC解码器的可执行文件和处理脚本的准备

。

### xIVC解码器客观测试

。

## xIVC解码器测试结果及数据分析

### 测试结果

### 测试结果对比及分析

。

## xIVC解码器的实际开发应用

## 本章小结

。

# 总结及展望

。

# 致谢

。

# 参考文献

1. 秦岭, 王煜坚, 李东新, 等. 视频编码标准 H. 264 的主要技术特点及其应用前景[J]. 微计算机应用, 2004, 25(4): 449-455.
2. 欧阳合，韩军.视频编解码器设计：开发图像与视频压缩系统，MPEG（活动图像专家组）.国防科技大学出版社.60-70
3. Sullivan G J, Ohm J R, Han W J, et al. Overview of the high efficiency video coding (HEVC) standard[J]. Circuits and Systems for Video Technology, IEEE Transactions on, 2012, 22(12): 1649-1668.