芋道源码 —— 知识星球

我是一段不羁的公告！
记得给艿艿这 3 个项目加油，添加一个 STAR 噢。
https://github.com/YunaiV/SpringBoot-Labs
https://github.com/YunaiV/onemall
https://github.com/YunaiV/ruoyi-vue-pro
2019-02-21
Dubbo

精尽 Dubbo 源码分析 —— 序列化（三）之 Kryo 实现

本文基于 Dubbo 2.6.1 版本，望知悉。

1. 概述

本文分享基于 Kryo 的序列化拓展实现。

> Java对象序列化框架 Kryo
>
> Kryo 是一个快速高效的Java对象图形序列化框架，主要特点是性能、高效和易用。该项目用来序列化对象到文件、数据库或者网络。
>
> 示例代码：

```
Kryo kryo = new Kryo();
// ...
Output output = new Output(new FileOutputStream("file.bin"));
SomeClass someObject = ...
kryo.writeObject(output, someObject);
output.close();
// ...
Input input = new Input(new FileInputStream("file.bin"));
SomeClass someObject = kryo.readObject(input, SomeClass.class);
input.close();
```

本文涉及，类图如下：

2. KryoSerialization

com.alibaba.dubbo.common.serialize.support.kryo.KryoSerialization ， 实现 Serialization 接口，Kryo 序列化实现类。代码如下：

```
public class KryoSerialization implements Serialization {

    @Override
    public byte getContentTypeId() {
        return 8;
    }

    @Override
    public String getContentType() {
        return "x-application/kryo";
    }

    @Override
    public ObjectOutput serialize(URL url, OutputStream out) throws IOException {
        return new KryoObjectOutput(out);
    }

    @Override
    public ObjectInput deserialize(URL url, InputStream is) throws IOException {
        return new KryoObjectInput(is);
    }

}
```

3. KryoObjectInput

com.alibaba.dubbo.common.serialize.support.kryo.KryoObjectInput ， 实现 ObjectInput, Cleanable 接口，Kryo 对象输入实现类。

构造方法

```
/**
 * Kryo 对象
 */
private Kryo kryo;
/**
 * Kryo 输入
 */
private Input input;

public KryoObjectInput(InputStream inputStream) {
    input = new Input(inputStream);
    this.kryo = KryoUtils.get();
}
```

kryo 属性，通过 KryoUtils#get() 方法，获取。

ObjectInput 实现方法

① 来自 DataInput 的实现方法，调用 input 对应的方法。例如：

```
@Override
public boolean readBool() throws IOException {
    try {
        return input.readBoolean();
    } catch (KryoException e) {
        throw new IOException(e);
    }
}
```

② 来自 ObjectInput 的实现方法，调用 kryo 对应的方法。例如：

```
@Override
public Object readObject() throws IOException, ClassNotFoundException {
    // TODO optimization
    try {
        return kryo.readClassAndObject(input);
    } catch (KryoException e) {
        throw new IOException(e);
    }
}
```

通过读取类，在根据类解析具体对象，字节内容“大体”是 [Class, 对象二进制数据] 。在 《深入理解RPC之序列化篇 - Kryo》 的 「三种读写方式」 ，对这块解析的相当不错。

Cleanable 实现方法

```
@Override
public void cleanup() {
    // 释放 Kryo 对象
    KryoUtils.release(kryo);
    // 清空
    kryo = null;
}
```

4. KryoObjectOutput

com.alibaba.dubbo.common.serialize.support.kryo.KryoObjectOutput ，实现 ObjectOutput, Cleanable 接口，Kryo 对象输出实现类。

构造方法

```
/**
 * Kryo 对象
 */
private Kryo kryo;
/**
 * Kryo 输出
 */
private Output output;
```

```
public KryoObjectOutput(OutputStream outputStream) {
    output = new Output(outputStream);
    this.kryo = KryoUtils.get();
}
```

`kryo` 属性，通过 `KryoUtils#get()` 方法，获取。

ObjectOutput 实现方法

① 来自 DataOutput 的实现方法，调用 `input` 对应的方法。例如：

```
@Override
public void writeBool(boolean v) throws IOException {
    output.writeBoolean(v);
}
```

② 来自 ObjectOutput 的实现方法，调用 `kryo` 对应的方法。例如：

```
@Override
@Override
public void writeObject(Object v) throws IOException {
    // TODO carries class info every time.
    kryo.writeClassAndObject(output, v);
}
```

通过写入类 + 具体对象，字节内容“大体”是 [Class, 对象二进制数据] 。在 《深入理解RPC之序列化篇 - Kryo》 的 「三种读写方式」 ，对这块解析的相当不错。

Cleanable 实现方法

```
@Override
public void cleanup() {
    // 释放 Kryo 对象
    KryoUtils.release(kryo);
    // 清空
    kryo = null;
}
```

5. CompatibleKryo

`com.alibaba.dubbo.common.serialize.support.kryo.CompatibleKryo` ， 实现 Kryo 类，兼容空构造方法的 Kryo 实现类。代码如下：

```
 1: @Override
 2: public Serializer getDefaultSerializer(Class type) {
 3:     if (type == null) {
 4:         throw new IllegalArgumentException("type cannot be null.");
 5:     }
```

```
 6:     // 空构造方法时，使用 JavaSerializer ，Java 原生序列化实现
 7:     if (!type.isArray() && !type.isEnum() && !ReflectionUtils.checkZeroArgConstructor(type)) {
 8:         if (logger.isWarnEnabled()) {
 9:             logger.warn(type + " has no zero-arg constructor and this will affect the serialization performance")
10:         }
11:         return new JavaSerializer();
12:     }
13:     // 使用 Kryo 默认序列化实现
14:     return super.getDefaultSerializer(type);
15: }
```

第 6 至 12 行：Kryo 不支持不包含空构造方法的类的序列化，因此，此时使用 Kryo 封装 Java 原生序列化实现类 `com.esotericsoftware.kryo.serializers.JavaSerializer` 。

- `ReflectionUtils#checkZeroArgConstructor(type)` 方法，代码如下：

```
/**
 * 判断类是否有空构造方法
 *
 * @param clazz 类
 * @return 是否
 */
public static boolean checkZeroArgConstructor(Class clazz) {
    try {
        clazz.getDeclaredConstructor();
        return true;
    } catch (NoSuchMethodException e) {
        return false;
    }
}
```

- x

第 14 行：使用 Kryo 默认序列化实现。

6. KryoFactory

6.1 AbstractKryoFactory

com.alibaba.dubbo.common.serialize.support.kryo.utils.AbstractKryoFactory ，实现 `com.esotericsoftware.kryo.pool.KryoFactory` 接口，Kryo 工厂抽象类。

构造方法

```
/**
 * 需要注册的类的集合
 */
private final Set<Class> registrations = new LinkedHashSet<Class>();

/**
 * 是否开启注册行为
 */
```

```
private boolean registrationRequired;
/**
 * Kryo 是否已经创建
 */
private volatile boolean kryoCreated;
```

registrations 静态属性，需要注册的类的集合。通过 #registerClass(Class) 方法，可以添加，代码如下：

```
/**
 * only supposed to be called at startup time
 *
 * later may consider adding support for custom serializer, custom id, etc
 */
public void registerClass(Class clazz) {
    if (kryoCreated) {
        throw new IllegalStateException("Can't register class after creating kryo instance");
    }
    registrations.add(clazz);
}
```

registrationRequired 属性，是否开启注册行为，默认关闭。

> Kryo 支持对注册行为，如 kryo.register(SomeClazz.class); ，这会赋予该 Class 一个从 0 开始的编号，但 Kryo 使用注册行为最大的问题在于，其不保证同一个 Class 每一次注册的号码相同，这与注册的顺序有关，也就意味着在不同的机器、同一个机器重启前后都有可能拥有不同的编号，这会导致序列化产生问题，所以在分布式项目中，一般关闭注册行为。

kryoCreated 属性，Kryo 是否已经创建。

抽象方法

```
/**
 * 返还 Kryo 对象
 *
 * @param kryo Kyro
 */
public abstract void returnKryo(Kryo kryo);

/**
 * 获得 Kryo 对象
 *
 * @return Kryo 对象
 */
public abstract Kryo getKryo();
```

create

```
1: @Override
2: public Kryo create() {
3:     // 标记已创建
4:     if (!kryoCreated) {
```

```
 5:         kryoCreated = true;
 6:     }
 7:
 8:     // 创建 CompatibleKryo 对象
 9:     Kryo kryo = new CompatibleKryo();
10:
11:     // TODO
12: //    kryo.setReferences(false);
13:     kryo.setRegistrationRequired(registrationRequired);
14:
15:     // 注册常用类
16:     kryo.register(Collections.singletonList("").getClass(), new ArraysAsListSerializer());
17:     kryo.register(GregorianCalendar.class, new GregorianCalendarSerializer());
18:     kryo.register(InvocationHandler.class, new JdkProxySerializer());
19:     kryo.register(BigDecimal.class, new DefaultSerializers.BigDecimalSerializer());
20:     kryo.register(BigInteger.class, new DefaultSerializers.BigIntegerSerializer());
21:     kryo.register(Pattern.class, new RegexSerializer());
22:     kryo.register(BitSet.class, new BitSetSerializer());
23:     kryo.register(URI.class, new URISerializer());
24:     kryo.register(UUID.class, new UUIDSerializer());
25:     UnmodifiableCollectionsSerializer.registerSerializers(kryo);
26:     SynchronizedCollectionsSerializer.registerSerializers(kryo);
27:
28:     // 注册常用数据结构
29:     // now just added some very common classes
30:     // TODO optimization
31:     kryo.register(HashMap.class);
32:     kryo.register(ArrayList.class);
33:     kryo.register(LinkedList.class);
34:     kryo.register(HashSet.class);
35:     kryo.register(TreeSet.class);
36:     kryo.register(Hashtable.class);
37:     kryo.register(Date.class);
38:     kryo.register(Calendar.class);
39:     kryo.register(ConcurrentHashMap.class);
40:     kryo.register(SimpleDateFormat.class);
41:     kryo.register(GregorianCalendar.class);
42:     kryo.register(Vector.class);
43:     kryo.register(BitSet.class);
44:     kryo.register(StringBuffer.class);
45:     kryo.register(StringBuilder.class);
46:     kryo.register(Object.class);
47:     kryo.register(Object[].class);
48:     kryo.register(String[].class);
49:     kryo.register(byte[].class);
50:     kryo.register(char[].class);
51:     kryo.register(int[].class);
52:     kryo.register(float[].class);
53:     kryo.register(double[].class);
54:
55:     // `registrations` 的注册
56:     for (Class clazz : registrations) {
57:         kryo.register(clazz);
58:     }
59:
60:     // SerializableClassRegistry 的注册
61:     for (Class clazz : SerializableClassRegistry.getRegisteredClasses()) {
62:         kryo.register(clazz);
63:     }
64:
```

```
65:     return kryo;
66: }
```

第 3 至 6 行：标记已创建 `kryoCreated = true`。
第 9 行：创建 CompatibleKryo 对象。
第 13 行：调用 `Kryo#setRegistrationRequired(registrationRequired)` 方法，设置是否要开启注册的功能。
开始一顿注册。
- 第 15 至 26 行：注册常用类到 Kryo 对象。
- 第 28 至 53 行：注册常用数据结构到 Kryo 对象。
- 第 55 至 58 行：注册 `registrations` 到 Kryo 对象。
- 第 60 至 63 行：注册 SerializableClassRegistry 到 Kryo 对象。

6.2 ThreadLocalKryoFactory

com.alibaba.dubbo.common.serialize.support.kryo.utils.ThreadLocalKryoFactory ，实现 AbstractKryoFactory 抽象类，基于 ThreadLocal 的 Kryo 工厂实现类。代码如下：

```
public class ThreadLocalKryoFactory extends AbstractKryoFactory {

    private final ThreadLocal<Kryo> holder = new ThreadLocal<Kryo>() {

        @Override
        protected Kryo initialValue() {
            return create(); // 创建 Kryo
        }

    };

    @Override
    public void returnKryo(Kryo kryo) {
        // do nothing
    }

    @Override
    public Kryo getKryo() {
        return holder.get();
    }

}
```

Kryo 的序列化和反序列的过程，是非线程安全的。所以通过 ThreadLocal 来保证，每个线程拥有一个 Kryo 对象。

6.3 KryoUtils

com.alibaba.dubbo.common.serialize.support.kryo.utils.KryoUtils ，Kryo 工具类，目前仅仅对 KryoFactory 进行操作。代码如下：

```
public class KryoUtils {
```

```
    private static AbstractKryoFactory kryoFactory = new ThreadLocalKryoFactory();

    public static Kryo get() {
        return kryoFactory.getKryo();
    }

    public static void release(Kryo kryo) {
        kryoFactory.returnKryo(kryo);
    }

    public static void register(Class<?> clazz) {
        kryoFactory.registerClass(clazz);
    }

    public static void setRegistrationRequired(boolean registrationRequired) {
        kryoFactory.setRegistrationRequired(registrationRequired);
    }

}
```

666. 彩蛋

推荐阅读：

[《深入理解RPC之序列化篇 - Kryo》]
[《Kryo官方文档-中文翻译》]

文章目录

回到首页