高性能NIO框架Netty-整合kryo高性能数据传输 ☆

尹吉欢 · 2018-03-04 · 2条评论 · 4052人阅读

java (http://cxytiandi.com/article/search/java) netty (http://cxytiandi.com/article/search/netty)

前言

本篇文章是Netty专题的第三篇，前面2篇文章如下：

- 高性能NIO框架Netty入门篇 (http://cxytiandi.com/blog/detail/17345)
- 高性能NIO框架Netty-对象传输 (http://cxytiandi.com/blog/detail/17403)

Netty 是 开源的基于java的网络通信框架，在上篇文章高性能NIO框架Netty-对象传输 (http://cxytiandi.com/blog/detail/17403)中对象的传输用的是自定义的编解码器，基于JDK的序列化来实现的，其实Netty自带的Object编解码器就可以实现对象的传输，并且也是基于JDK的序列化，而Kryo是性能更好的java序列化框架，本篇文章我们将用Kryo来替换JDK的序列化实现高性能的数据传输。

Kryo可能大家用的还不是特别多，我第一次见Kryo是在当当扩展的dubbox (https://github.com/dangdangdotcom/dubbox)中，其中有一条主要功能是这么介绍的：

- **支持基于Kryo和FST的Java高效序列化实现**：基于当今比较知名的Kryo (https://github.com/EsotericSoftware/kryo)和FST (https://github.com/RuedigerMoeller/fast-serialization)高性能序列化库，为Dubbo默认的RPC协议添加新的序列化实现，并优化调整了其序列化体系，比较显著的提高了Dubbo RPC的性能，详见文档中的基准测试报告。

为了提高RPC的性能，增加了Kryo和FST两种高性能的序列化方式，基准测试报告地址：https://dangdangdotcom.github.io/dubbox/serialization.html (https://dangdangdotcom.github.io/dubbox/serialization.html)

Kryo介绍

Kryo是一种快速高效的Java对象序列化框架。 该项目的目标是速度、效率和易于使用的API。 当对象需要持久化时，无论是用于文件、数据库还是通过网络，该项目都很有用。

Kryo还可以执行自动深层浅层的复制/克隆。这是从对象直接复制到对象，而不是object-> bytes-> object。

除了前面介绍的dubbox使用了Kryo，还有很多的开源框架都用到了Kryo，请看下面的列表：

- KryoNet (http://code.google.com/p/kryonet/) (NIO networking)
- Twitter’ s Scalding (https://github.com/twitter/scalding) (Scala API for Cascading)

- Twitter’s Chill (https://github.com/twitter/chill) (Kryo serializers for Scala)
- Apache Fluo (https://fluo.apache.org/) (Kryo is default serialization for Fluo Recipes)
- Apache Hive (http://hive.apache.org/) (query plan serialization)
- Apache Spark (http://spark.apache.org/) (shuffled/cached data serialization)
- DataNucleus (https://github.com/datanucleus/type-converter-kryo) (JDO/JPA persistence framework)
- CloudPelican (http://www.cloudpelican.com/)
- Yahoo’s S4 (http://www.s4.io/) (distributed stream computing)
- Storm (https://github.com/nathanmarz/storm/wiki/Serialization) (distributed realtime computation system, in turn used by many others (https://github.com/nathanmarz/storm/wiki/Powered-By))
- Cascalog (https://github.com/nathanmarz/cascalog) (Clojure/Java data processing and querying details (https://groups.google.com/d/msg/cascalog-user/qgwO2vbkRa0/UeClnLL5OsgJ))
- memcached-session-manager (https://code.google.com/p/memcached-session-manager/) (Tomcat high-availability sessions)
- Mobility-RPC (http://code.google.com/p/mobility-rpc/) (RPC enabling distributed applications)
- akka-kryo-serialization (https://github.com/romix/akka-kryo-serialization) (Kryo serializers for Akka)
- Groupon (https://code.google.com/p/kryo/issues/detail?id=67)
- Jive (http://www.jivesoftware.com/jivespace/blogs/jivespace/2010/07/29/the-jive-sbs-cache-redesign-part-3)
- DestroyAllHumans (https://code.google.com/p/destroyallhumans/) (controls a robot (http://www.youtube.com/watch?v=ZeZ3R38d3Cg)!)
- kryo-serializers (https://github.com/magro/kryo-serializers) (additional serializers)

Kryo简单使用

添加Kryo的Maven依赖,我这边用的是比较老的版本，跟dubbox中的版本一致，当然大家也可以用最新的4.0版本

```
<!-- kryo -->
<dependency>
    <groupId>com.esotericsoftware.kryo</groupId>
    <artifactId>kryo</artifactId>
    <version>2.24.0</version>
</dependency>
<dependency>
    <groupId>de.javakaffee</groupId>
    <artifactId>kryo-serializers</artifactId>
    <version>0.26</version>
</dependency>
```

创建一个测试类来演示下序列化和反序列化的功能

```java
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;

import com.esotericsoftware.kryo.Kryo;
import com.esotericsoftware.kryo.io.Input;
import com.esotericsoftware.kryo.io.Output;

public class KryoTest {
    public static void main(String[] args) throws FileNotFoundException {
        // 序列化
        Kryo kryo = new Kryo();
        Output output = new Output(new FileOutputStream("file.bin"));
        Message someObject = new Message();
        someObject.setContent("测试序列化");
        kryo.writeObject(output, someObject);
        output.close();

        // 反序列化
        Input input = new Input(new FileInputStream("file.bin"));
        Message message = kryo.readObject(input, Message.class);
        System.out.println(message.getContent());
        input.close();
    }
}
```

更多使用方式和细节请查看文档：https://github.com/EsotericSoftware/kryo (https://github.com/EsotericSoftware/kryo)

Netty整合Kryo进行序列化

1. 创建一个工厂类KryoFactory，用于创建Kryo对象

```
import com.esotericsoftware.kryo.Kryo;
import com.esotericsoftware.kryo.serializers.DefaultSerializers;
import com.netty.im.core.message.Message;

import de.javakaffee.kryoserializers.*;

import java.lang.reflect.InvocationHandler;
import java.math.BigDecimal;
import java.math.BigInteger;
import java.net.URI;
import java.text.SimpleDateFormat;
import java.util.*;
import java.util.concurrent.ConcurrentHashMap;
import java.util.regex.Pattern;


public abstract class KryoFactory {

    private final static KryoFactory threadFactory = new ThreadLocalKryoFactory();

    protected KryoFactory() {

    }

    public static KryoFactory getDefaultFactory() {
        return threadFactory;
    }

    protected Kryo createKryo() {

        Kryo kryo = new Kryo();
        kryo.setRegistrationRequired(false);
        kryo.register(Message.class);
        kryo.register(Arrays.asList("").getClass(), new ArraysAsListSerializer());
        kryo.register(GregorianCalendar.class, new GregorianCalendarSerializer());
        kryo.register(InvocationHandler.class, new JdkProxySerializer());
        kryo.register(BigDecimal.class, new DefaultSerializers.BigDecimalSerializer());
        kryo.register(BigInteger.class, new DefaultSerializers.BigIntegerSerializer());
        kryo.register(Pattern.class, new RegexSerializer());
        kryo.register(BitSet.class, new BitSetSerializer());
        kryo.register(URI.class, new URISerializer());
        kryo.register(UUID.class, new UUIDSerializer());
        UnmodifiableCollectionsSerializer.registerSerializers(kryo);
        SynchronizedCollectionsSerializer.registerSerializers(kryo);
```

```
45.
46.         kryo.register(HashMap.class);
47.         kryo.register(ArrayList.class);
48.         kryo.register(LinkedList.class);
49.         kryo.register(HashSet.class);
50.         kryo.register(TreeSet.class);
51.         kryo.register(Hashtable.class);
52.         kryo.register(Date.class);
53.         kryo.register(Calendar.class);
54.         kryo.register(ConcurrentHashMap.class);
55.         kryo.register(SimpleDateFormat.class);
56.         kryo.register(GregorianCalendar.class);
57.         kryo.register(Vector.class);
58.         kryo.register(BitSet.class);
59.         kryo.register(StringBuffer.class);
60.         kryo.register(StringBuilder.class);
61.         kryo.register(Object.class);
62.         kryo.register(Object[].class);
63.         kryo.register(String[].class);
64.         kryo.register(byte[].class);
65.         kryo.register(char[].class);
66.         kryo.register(int[].class);
67.         kryo.register(float[].class);
68.         kryo.register(double[].class);
69.
70.         return kryo;
71.     }
72. }
```

kryo在序列化对象时，首先会序列化其类的全限定名，由于我们通常序列化的对象都是有限范围内的类的实例，这样重复序列化同样的类的全限定名是低效的。通过注册kryo可以将类的全限定名抽象为一个数字，即用一个数字代表全限定名，这样就要高效一些。kryo.register()方法就是将需要序列化的类提前进行注册。

2.创建一个ThreadLocalKryoFactory继承KryoFactory，用来为每个线程创建一个Kryo对象，原因是由于Kryo 不是线程安全的。每个线程都应该有自己的 Kryo，Input 和 Output 实例。此外， bytes[] Input 可能被修改，然后在反序列化期间回到初始状态，因此不应该在多线程中并发使用相同的 bytes[]。

Kryo 实例的创建/初始化是相当昂贵的，所以在多线程的情况下，您应该线程池化 Kryo 实例。简单的解决方案是使用 ThreadLocal 将 Kryo实例绑定到 Threads。

```
import com.esotericsoftware.kryo.Kryo;

public class ThreadLocalKryoFactory extends KryoFactory {

    private final ThreadLocal<Kryo> holder  = new ThreadLocal<Kryo>() {
        @Override
        protected Kryo initialValue() {
            return createKryo();
        }
    };

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

3.创建一个序列化的工具类KryoSerializer

```java
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import com.esotericsoftware.kryo.Kryo;
import com.esotericsoftware.kryo.io.Input;
import com.esotericsoftware.kryo.io.Output;
import io.netty.buffer.ByteBuf;
import io.netty.buffer.ByteBufInputStream;
/**
 * Kryo序列化
 * @author yinjihuan
 *
 */
public class KryoSerializer {

    private static final ThreadLocalKryoFactory factory = new ThreadLocalKryoFactory();

    public static void serialize(Object object, ByteBuf out) {
        Kryo kryo = factory.getKryo();
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        Output output = new Output(baos);
        kryo.writeClassAndObject(output, object);
        output.flush();
        output.close();

        byte[] b = baos.toByteArray();
        try {
            baos.flush();
            baos.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
        out.writeBytes(b);
    }

    public static Object deserialize(ByteBuf out) {
        if (out == null) {
            return null;
        }
        Input input = new Input(new ByteBufInputStream(out));
        Kryo kryo = factory.getKryo();
        return kryo.readClassAndObject(input);
    }

}
```

4.创建Netty编码器KryoEncoder对数据进行Kryo序列化

```
import com.netty.im.core.serialize.kryo.KryoSerializer;
import io.netty.buffer.ByteBuf;
import io.netty.channel.ChannelHandlerContext;
import io.netty.handler.codec.MessageToByteEncoder;

public class KryoEncoder extends MessageToByteEncoder<Message> {

    @Override
    protected void encode(ChannelHandlerContext ctx, Message message, ByteBuf out) throws Exception {
        KryoSerializer.serialize(message, out);
        ctx.flush();
    }

}
```

5.创建Netty解码器KryoDecoder对数据进行Kryo反序列化

```

import java.util.List;
import com.netty.im.core.serialize.kryo.KryoSerializer;
import io.netty.buffer.ByteBuf;
import io.netty.channel.ChannelHandlerContext;
import io.netty.handler.codec.ByteToMessageDecoder;

public class KryoDecoder extends ByteToMessageDecoder {

    @Override
    protected void decode(ChannelHandlerContext ctx, ByteBuf in, List<Object> out) throws Exception {
        Object obj = KryoSerializer.deserialize(in);
        out.add(obj);
    }

}
```

6.将Netty服务端和客户端的编解码器都改成Kryo的编解码器即可

```
ch.pipeline().addLast("decoder", new KryoDecoder());
ch.pipeline().addLast("encoder", new KryoEncoder());
```