QuickStart For MessagePack Java 0.6.X

中文标题【MessagePack Java 0.6.X 快速开始指南】

0.6.x 版本的 `MessagePack` 已经过期被淘汰了。如果你现在开始使用 `MessagePack` 话，请不要使用这个版本。

我们在这里保留 `0.6.x` 版本的内容主要用于参考用途。

最新的MessagePack 版本请参考：https://github.com/msgpack/msgpack-java中的项目源代码。

MessagePack 中文文档请参考：http://docs.ossez.com/messagepack-docs/index.html

MessagePack 测试和示例源代码：https://github.com/cwiki-us-demo/serialize-deserialize-demo-java

这个指南提供了使用msgpack-java 的快速指南。在开始的时候，我们将会介绍如何安装msgpack-java，然后将会运行如何使用msgpack 来对对象序列化/反序列化（serialize/deserizalize）对象。

本页中的内容：

安装

你可以使用下面 2 种方法来安装 msgpack-java —— 从 maven 下载或者直接构建 jar 包。

从 Maven2 仓库中进行安装

MessagePack 针对 Java 的使用已经发布到 Maven 的中央仓库中（Maven Central Repository）。你可以使用下面的参数来配置你项目的pom.xml 文件。

```
<dependency>
    <groupId>org.msgpack</groupId>
    <artifactId>msgpack</artifactId>
    <version>${msgpack.version}</version>
</dependency>
```

你需要将${msgpack.version} 替换为当前的MessagePack 版本，有关可以使用的具体版本你可以访问http://repo1.maven.org/maven2/org/msgpack/msgpack/中的版本。

请注意，在 0.6.x 版本中最新的版本只更新到 0.6.12。

从 git 仓库中进行安装

你可以从代码仓库中获得最新的代码。

```
$ git clone git@github.com:msgpack/msgpack-java.git
$ cd msgpack-java
$ mvn package
```

使用上面的代码进行编译后，你将会在msgpack-java/target 目录中得到msgpack.jar 包。

同时你也需要https://code.google.com/p/json-simple/和https://github.com/jboss-javassist/javassist来让msgpack.jar 可以在项目中使用。否则你将会收到NoClassDefFoundError 错误。

如何使用

下面是如何使用的示例代码。

使用一个消息打包（message-packable）类

使用注解`@Message` 来让你可以序列化你自己类中对象的public 字段。

本代码可以在https://github.com/cwiki-us-demo/messagepack-6-demo-java/blob/master/src/test/java/com/insight/demo/msgpack/MessagePack6Object.java

```
package com.insight.demo.msgpack;

import org.junit.Test;
import org.msgpack.MessagePack;
import org.msgpack.annotation.Message;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.util.UUID;

import static org.junit.Assert.assertEquals;

/**
 * MessagePack6Objects
 *
 * @author yhu
 */
public class MessagePack6Object {
    final Logger logger = LoggerFactory.getLogger(MessagePack6Object.class);

    /**
     * MessageData Message Object
     */
    @Message // Annotation
    public static class MessageData {
        // public fields are serialized.
        public String uuid;
        public String name;
        public double version;
    }


    /**
     * Test MessagePack6Objects
     */
    @Test
    public void testMessagePack6Objects() {
        logger.debug("MessagePack6Objects for Objects");

        String uuid = UUID.randomUUID().toString();

        // INIT OBJ
        MessageData src = new MessageData();
        src.uuid = uuid;
        src.name = "MessagePack6";
        src.version = 0.6;

        try {
            MessagePack msgPack = new MessagePack();

            // Serialization
            logger.debug("------ Serialization ------");
            byte[] bytes = msgPack.write(src);
            logger.debug("Bytes Array Length: [{}]", bytes.length);

            // Deserialization
            logger.debug("------ Deserialization ------");
            MessageData dst = msgPack.read(bytes, MessageData.class);
            logger.debug("Check Object for UUID: [{}]", dst.uuid);

            assertEquals(uuid, dst.uuid);

        } catch (Exception ex) {
            logger.error("MessagePack Serialization And Deserialization error", ex);
        }
    }
}
```

如果你希望按照顺序序列化多个对象的话，你可以使用Packer和Unpacker多个对象。

这是因为MessagePack.write(Object)和read(byte[]) 实际上每次都调用创建了Packer 和Unpacker 对象。

为了使用Packer和Unpacker对象，请调用createPacker(OutputStream) 和createUnpacker(InputStream)。

本代码可以https://github.com/cwiki-us-demo/messagepack-6-demo-java/blob/master/src/test/java/com/insight/demo/msgpack/MessagePack6Objects.java中查看。

```
package com.insight.demo.msgpack;

import org.junit.Test;
import org.msgpack.MessagePack;
import org.msgpack.annotation.Message;
import org.msgpack.packer.Packer;
import org.msgpack.unpacker.Unpacker;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.util.UUID;

import static org.junit.Assert.assertEquals;

/**
 * MessagePack6Objects
 *
 * @author yhu
 */
public class MessagePack6Objects {
    final Logger logger = LoggerFactory.getLogger(MessagePack6Objects.class);

    /**
     * MessageData Message Objects
     */
    @Message // Annotation
    public static class MessageData {
        // public fields are serialized.
        public String uuid;
        public String name;
        public double version;
    }


    /**
     * Test MessagePack6Objects
     */
    @Test
    public void testMessagePack6Objects() {
        logger.debug("MessagePack6Objects for Objects");

        String uuid = UUID.randomUUID().toString();

        // INIT OBJ
        MessageData src1 = new MessageData();
        src1.uuid = uuid;
        src1.name = "MessagePack6-src1";
        src1.version = 0.6;

        MessageData src2 = new MessageData();
        src2.uuid = uuid;
        src2.name = "MessagePack6-src2";
        src2.version = 10.6;

        MessageData src3 = new MessageData();
        src3.uuid = uuid;
        src3.name = "MessagePack6-src3";
        src3.version = 1.6;
```

```
        try {
            MessagePack msgPack = new MessagePack();

            // Serialization
            logger.debug("------ Serialization ------");
            ByteArrayOutputStream out = new ByteArrayOutputStream();
            Packer packer = msgPack.createPacker(out);
            packer.write(src1);
            packer.write(src2);
            packer.write(src3);

            byte[] bytes = out.toByteArray();
            logger.debug("Bytes Array Length: [{}]", bytes.length);

            // Deserialization
            logger.debug("------ Deserialization ------");
            ByteArrayInputStream in = new ByteArrayInputStream(bytes);
            Unpacker unpacker = msgPack.createUnpacker(in);

            MessageData dst1 = unpacker.read(MessageData.class);
            MessageData dst2 = unpacker.read(MessageData.class);
            MessageData dst3 = unpacker.read(MessageData.class);

            logger.debug("Check Object for UUID: [{}]", dst1.uuid);

            assertEquals(uuid, dst1.uuid);

        } catch (Exception ex) {
            logger.error("MessagePack Serialization And Deserialization error", ex);
        }
    }
}
```

多种类型变量的序列化和反序列化（serialization/deserialization）

类Packer/Unpacker允许序列化和反序列化多种类型的变量，如后续程序所示。这个类启用序列化和反序列化多种类型的变量和序列化主要类型变量以及包装类，String 对象，byte[]对象，ByteBuffer对象等的方法相似。

如上面提示的，你可以序列化和反序列化你自己的对象，前提是你自己的对象需要使用@Message 注解。

https://github.com/cwiki-us-demo/messagepack-6-demo-java/blob/master/src/test/java/com/insight/demo/msgpack/MessagePack6Types.java

```
package com.insight.demo.msgpack;

import org.junit.Test;
import org.msgpack.MessagePack;
import org.msgpack.packer.Packer;
import org.msgpack.unpacker.Unpacker;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.math.BigInteger;
import java.nio.ByteBuffer;

/**
 * MessagePack6Types
 *
 * @author yhu
 */
public class MessagePack6Types {
    final Logger logger = LoggerFactory.getLogger(MessagePack6Types.class);
```

```
    /**
     * Test MessagePack6Types
     */
    @Test
    public void testMessagePack6Types() {
        logger.debug("testMessagePack6Types for Types");

        MessagePack msgpack = new MessagePack();
        try {

            //
            // Serialization
            //
            ByteArrayOutputStream out = new ByteArrayOutputStream();
            Packer packer = msgpack.createPacker(out);

            // Serialize values of primitive types
            packer.write(true); // boolean value
            packer.write(10); // int value
            packer.write(10.5); // double value

            // Serialize objects of primitive wrapper types
            packer.write(Boolean.TRUE);
            packer.write(new Integer(10));
            packer.write(new Double(10.5));

            // Serialize various types of arrays
            packer.write(new int[]{1, 2, 3, 4});
            packer.write(new Double[]{10.5, 20.5});
            packer.write(new String[]{"msg", "pack", "for", "java"});
            packer.write(new byte[]{0x30, 0x31, 0x32}); // byte array

            // Serialize various types of other reference values
            packer.write("MessagePack"); // String object
            packer.write(ByteBuffer.wrap(new byte[]{0x30, 0x31, 0x32})); // ByteBuffer object
            packer.write(BigInteger.ONE); // BigInteger object

            //
            // Deserialization
            //
            byte[] bytes = out.toByteArray();
            ByteArrayInputStream in = new ByteArrayInputStream(bytes);
            Unpacker unpacker = msgpack.createUnpacker(in);

            // to primitive values
            boolean b = unpacker.readBoolean(); // boolean value
            int i = unpacker.readInt(); // int value
            double d = unpacker.readDouble(); // double value

            // to primitive wrapper value
            Boolean wb = unpacker.read(Boolean.class);
            Integer wi = unpacker.read(Integer.class);
            Double wd = unpacker.read(Double.class);

            // to arrays
            int[] ia = unpacker.read(int[].class);
            Double[] da = unpacker.read(Double[].class);
            String[] sa = unpacker.read(String[].class);
            byte[] ba = unpacker.read(byte[].class);

            // to String object, ByteBuffer object, BigInteger object, List object and Map object
            String ws = unpacker.read(String.class);
            ByteBuffer buf = unpacker.read(ByteBuffer.class);
            BigInteger bi = unpacker.read(BigInteger.class);

        } catch (Exception ex) {
            logger.error("MessagePack Serialization And Deserialization error", ex);
        }
    }
}
```

方法Packer#write()允许序列化多种类型的数据。

类Unpacker针对反序列化二进制数据为主要变量，提供了一个反序列化方法。例如，你希望将二进制数据反序列化为boolean(或者int) 数据类型，你可以使用 Unpacker 中的readBoolean(或者readInt) 方法。

Unpacker同时也为参考变量提供了一个读取的方法。这个方法允许为一个参考变量从二进制数据中进行反序列化。参考变量的定义为你将类型指定为一个参数。例如，你希望反序列化二进制数据到String(或者byte[]) 对象，你必须在调用read(String.class)(或者read(byte[].class)) 方法的时候定义描述。

List, Map 对象的序列化和反序列化（serialization/deserialization）

为了序列化原生的容器对象例如 List 和Map对象，你必须使用Template。

Template　serializer 和 deserializer 的配对。例如，为了序列化一个 List 对象，在 List 对象中 Integer 对象为元素，你可以使用下面的方法来创建一个模板对象（Template object）。

```
Template listTmpl = Templates.tList(Templates.TInteger);
```

tListTInteger是静态方法，字段为Templates

List 和Map对象的用例如下显示：

本代码可以在https://github.com/cwiki-us-demo/messagepack-6-demo-java/blob/master/src/test/java/com/insight/demo/msgpack/MessagePack6Template.java中查看。

```
package com.insight.demo.msgpack;

import org.junit.Test;
import org.msgpack.MessagePack;
import org.msgpack.packer.Packer;
import org.msgpack.template.Template;
import org.msgpack.unpacker.Unpacker;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.math.BigInteger;
import java.nio.ByteBuffer;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import static org.msgpack.template.Templates.*;

/**
 * MessagePack6Template
 *
 * @author yhu
 */
public class MessagePack6Template {
    final Logger logger = LoggerFactory.getLogger(MessagePack6Template.class);


    /**
     * Test MessagePack6Template
     */
    @Test
    public void testMessagePack6Template() {
        logger.debug("MessagePack6Template for Template");

        MessagePack msgpack = new MessagePack();
        try {
```

```
            // Create templates for serializing/deserializing List and Map objects
            Template<List<String>> listTmpl = tList(TString);
            Template<Map<String, String>> mapTmpl = tMap(TString, TString);

            //
            // Serialization
            //

            ByteArrayOutputStream out = new ByteArrayOutputStream();
            Packer packer = msgpack.createPacker(out);

            // Serialize List object
            List<String> list = new ArrayList<String>();
            list.add("msgpack");
            list.add("for");
            list.add("java");
            packer.write(list); // List object

            // Serialize Map object
            Map<String, String> map = new HashMap<String, String>();
            map.put("sadayuki", "furuhashi");
            map.put("muga", "nishizawa");
            packer.write(map); // Map object

            //
            // Deserialization
            //

            byte[] bytes = out.toByteArray();
            ByteArrayInputStream in = new ByteArrayInputStream(bytes);
            Unpacker unpacker = msgpack.createUnpacker(in);

            // to List object
            List<String> dstList = unpacker.read(listTmpl);

            // to Map object
            Map<String, String> dstMap = unpacker.read(mapTmpl);

        } catch (Exception ex) {
            logger.error("MessagePack Serialization And Deserialization error", ex);
        }
    }
}
```

不使用注解（annotations）来序列化

如果你不能添加 `@Message` 到你的定义对象中但是你还是希望进行序列化。你可以使用register 方法来在类中启用序列化对象。

如下的代码所示：

```
MessagePack msgpack = new MessagePack();
msgpack.register(MyMessage2.class);
```

例如，如果MyMessage2 类被包含到了外部的库中了。你没有办法比较容易的编辑源代码，添加 `@Message` 到源代码中。

register 方法能够允许为MyMessage2 自动创建一个serializer 和deserializer 对。

你可以在执行方面后序列化对象MyMessage2。

可选字段

你可添加一个新的字段来保持可用性。在新字段中使用`@Optional`注解。

```
@Message
public static class MyMessage {
    public String name;
    public double version;

    // new field
    @Optional
    public int flag = 0;
}
```

如果你尝试反序列化老版本数据的话，可选字段将会被忽略。

动态类型

我们知道 Java 是一个静态类型的语言。通过输入 ValueMessagePack能够实现动态的特性。

Value 有方法来检查自己的类型（isIntegerType(),isArrayType(), 等...），同时也转换为自己的类型（asStringValue(),convert(Template)

本代码可以在https://github.com/cwiki-us-demo/messagepack-6-demo-java/blob/master/src/test/java/com/insight/demo/msgpack/MessagePack6DynamicTyping.java中查看。

```
package com.insight.demo.msgpack;

import org.junit.Test;
import org.msgpack.MessagePack;
import org.msgpack.type.Value;
import org.msgpack.unpacker.Converter;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.util.ArrayList;
import java.util.List;

import static org.msgpack.template.Templates.TString;
import static org.msgpack.template.Templates.tList;

/**
 * MessagePack6Objects
 *
 * @author yhu
 */
public class MessagePack6DynamicTyping {
    final Logger logger = LoggerFactory.getLogger(MessagePack6DynamicTyping.class);


    /**
     * Test MessagePack6Objects
     */
    @Test
    public void MessagePack6DynamicTyping() {
        logger.debug("MessagePack6Objects for Objects");

        // Create serialize objects.
        List<String> src = new ArrayList<String>();
        src.add("msgpack");
        src.add("kumofs");
        src.add("viver");

        MessagePack msgpack = new MessagePack();

        try {

            // Serialize
            byte[] raw = msgpack.write(src);

            // Deserialize directly using a template
            List<String> dst1 = msgpack.read(raw, tList(TString));

            // Or, Deserialze to Value then convert type.
            Value dynamic = msgpack.read(raw);
            List<String> dst2 = new Converter(dynamic).read(tList(TString));

        } catch (Exception ex) {
            logger.error("MessagePack Serialization And Deserialization error", ex);
        }
    }
}
```