# Bboss和xstream序列化/反序列化性能对比

本报告分别测试bboss和xstream的序列化和反序列化功能，测试的接口如下：

## 1.接口方法

Bboss序列化和反序列化方法

//序列化

String xml =

ObjectSerializable.*convertBeanObjectToXML*("person",joe,Person.**class**);

//反序列化

Person p = ObjectSerializable.*convertXMLToBeanObject*("person", xml, Person.**class**);

xStream序列化和反序列化方法

XStream *xStream* = **new** XStream();

String xmlXstream = *xStream*.toXML(joe);

Person p = (Person)*xStream*.fromXML(xmlXstream);

Bboss依赖的包：

**bboss-aop.jar**

**cglib-2.2.jar**

**frameworkset-util.jar**

**log4j-1.2.14.jar**

**bboss-soa.jar**

Xstream依赖的包：

**xpp3-1.1.4c.jar**

**xstream-1.3.1.jar**

接口基本上都非常简单。依赖的包也都非常少。

## 2.数据结构

测试的数据结构包含int，String，Date，Date[],Object嵌套结构，String数组，List，Map，Set，二进制文件

## 3.数据规模

分为以下三种

小负荷规模-1000字节数据，

大负荷规模-47K的 xml数据

文件对象数据-文件大小47K，文件内容为xml串。

## 4.测试环境

所以的测试用例基于以下环境测试：

联想thinkpad sl400   
OS 32位windows xp professional sp3   
内存2G   
cpu:   
型号 Intel(R) Core(TM)2 Duo T5870   
主频 2GHz   
  
用例运行工具：myeclipse 8.0   
jdk 1.5.0\_06

Junit 4

## 5.测试用例1-小负荷测试

### 5.1测试数据

PhoneNumber phone = **new** PhoneNumber();

phone.setCode(123);

phone.setNumber("1234-456");

PhoneNumber fax = **new** PhoneNumber();

fax.setCode(123);

fax.setNumber("<aaaa>9999-999</bbbb>");

Set dataSet = **new** TreeSet();

dataSet.add("aa");

dataSet.add("bb");

List dataList = **new** ArrayList();

dataList.add("aa");

dataList.add("bb");

Map dataMap = **new** HashMap();

dataMap.put("aa","aavalue");

dataMap.put("bb","bbvalue");

String[] dataArray = **new** String[]{"aa","bb"};

Person joe = **new** Person();

joe.setFirstname("Joe");

// joe.setLastname("Walnes");

//用来验证bboss和Xstream是否会按照null值传递，也就是说lastname有默认值"ssss"

//这样我们手动把lastname设置为null，理论上来说反序列化后joe中的lastname应该是null而不是默认值"ssss"

joe.setLastname(**null**);

joe.setPhone(phone);

joe.setFax(fax);

joe.setDataArray(dataArray);

joe.setDataList(dataList);

joe.setDataMap(dataMap);

joe.setDataSet(dataSet);

### 5.2测试结果

#### 5.2.1序列化

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **执行次数** | **Bboss耗时** | **XStream耗时** | **Bboss包大小** | **XStream包大小** |
| 1次 | 0毫秒 | 0毫秒 | 1002字节 | 753字节 |
| 10次 | 0毫秒 | 16毫秒 |
| 100次 | 31毫秒 | 78毫秒 |
| 1000次 | 78毫秒 | 218毫秒 |
| 10000次 | 469毫秒 | 1625毫秒 |

#### 5.2.2反序列化

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **执行次数** | **Bboss耗时** | **XStream耗时** | **Bboss包大小** | **XStream包大小** |
| 1次 | 0毫秒 | 0豪秒 | 1002字节 | 753字节 |
| 10次 | 47毫秒 | 15毫秒 |
| 100次 | 282毫秒 | 63毫秒 |
| 1000次 | 1250毫秒 | 312毫秒 |
| 10000次 | 9328毫秒 | 2344毫秒 |

## 6.测试用例2-大负荷测试

### 6.1测试数据

测试的对象实例字段包含了47k的xml串，本用例针对这个对象实例分别采用bboss和xstream来进行序列化和反序列化测试，并给出测试结果。

//这个文件中内容有47565 字节，约47k的数据

String bigcontent = FileUtil.*getFileContent*(**new** File("D:\\workspace\\bbossgroups-3.2\\bboss-soa\\test\\org\\frameworkset\\soa\\testxstream.xml"), "GBK");

PhoneNumber phone = **new** PhoneNumber();

phone.setCode(123);

phone.setNumber("1234-456");

PhoneNumber fax = **new** PhoneNumber();

fax.setCode(123);

//设置大字段的值

fax.setNumber(bigcontent);

Set dataSet = **new** TreeSet();

dataSet.add("aa");

dataSet.add("bb");

List dataList = **new** ArrayList();

dataList.add("aa");

dataList.add("bb");

Map dataMap = **new** HashMap();

dataMap.put("aa","aavalue");

dataMap.put("bb","bbvalue");

String[] dataArray = **new** String[]{"aa","bb"};

Person joe = **new** Person();

joe.setFirstname("Joe");

// joe.setLastname("Walnes");

//用来验证bboss和Xstream是否会按照null值传递，也就是说lastname有默认值"ssss"

//这样我们手动把lastname设置为null，理论上来说反序列化后joe中的lastname应该是null而不是默认值"ssss"

joe.setLastname(**null**);

joe.setPhone(phone);

joe.setFax(fax);

joe.setDataArray(dataArray);

joe.setDataList(dataList);

joe.setDataMap(dataMap);

joe.setDataSet(dataSet);

### 6.2测试结果

#### 6.2.1序列化

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **执行次数** | **Bboss耗时** | **XStream耗时** | **生成的Bboss包大小** | **生成XStream包大小** |
| 1次 | 32毫秒 | 31毫秒 | 48K | 78K |
| 10次 | 31毫秒 | 94毫秒 |
| 100次 | 312毫秒 | 875毫秒 |
| 1000次 | 2922毫秒 | 8078毫秒 |
| 10000次 | 29141毫秒 | 81938毫秒 |

#### 6.2.2反序列化

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **执行次数** | **Bboss耗时** | **XStream耗时** | **Bboss包大小** | **XStream包大小** |
| 1次 | 15毫秒 | 46毫秒 | 48K | 78K |
| 10次 | 125毫秒 | 235毫秒 |
| 100次 | 1391毫秒 | 2234毫秒 |
| 1000次 | 12547毫秒 | 22781毫秒 |
| 10000次 | 126047毫秒 | 230688毫秒 |

## 7.测试用例3-File对象测试

### 7.1测试数据

测试的对象实例字段包含File对象，文件大小为47K，本用例针对这个对象实例分别采用bboss和xstream来进行序列化和反序列化测试，并给出测试结果。

//这个文件中内容有47565 字节，约47k的数据

File fileData = **new** File("D:\\workspace\\bbossgroups-3.2\\bboss-soa\\test\\org\\frameworkset\\soa\\testxstream.xml");

PhoneNumber phone = **new** PhoneNumber();

phone.setCode(123);

phone.setNumber("1234-456");

PhoneNumber fax = **new** PhoneNumber();

fax.setCode(123);

fax.setNumber("<aaaa>9999-999</bbbb>");

Set dataSet = **new** TreeSet();

dataSet.add("aa");

dataSet.add("bb");

List dataList = **new** ArrayList();

dataList.add("aa");

dataList.add("bb");

Map dataMap = **new** HashMap();

dataMap.put("aa","aavalue");

dataMap.put("bb","bbvalue");

String[] dataArray = **new** String[]{"aa","bb"};

FilePerson joe = **new** FilePerson();

joe.setFileData(fileData);

joe.setFirstname("Joe");

// joe.setLastname("Walnes");

//用来验证bboss和Xstream是否会按照null值传递，也就是说lastname有默认值"ssss"

//这样我们手动把lastname设置为null，理论上来说反序列化后joe中的lastname应该是null而不是默认值"ssss"

joe.setLastname(**null**);

joe.setPhone(phone);

joe.setFax(fax);

joe.setDataArray(dataArray);

joe.setDataList(dataList);

joe.setDataMap(dataMap);

joe.setDataSet(dataSet);

### 7.2测试结果

#### 7.2.1序列化

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **执行次数** | **Bboss耗时** | **XStream耗时** | **生成的Bboss包大小** | **生成XStream包大小** |
| 1次 | 31毫秒 | 无法对文件对象内容进行序列化 | 65K | --- |
| 10次 | 141毫秒 | 无法对文件对象内容进行序列化 |
| 100次 | 1609毫秒 | 无法对文件对象内容进行序列化 |
| 1000次 | 16531毫秒 | 无法对文件对象内容进行序列化 |
| 10000次 | 166641毫秒 | 无法对文件对象内容进行序列化 |

#### 7.2.2反序列化

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **执行次数** | **Bboss耗时** | **XStream耗时** | **Bboss包大小** | **XStream包大小** |
| 1次 | 94毫秒 | 无法对文件对象内容进行反序列化 | 65K | --- |
| 10次 | 297毫秒 | 无法对文件对象内容进行反序列化 |
| 100次 | 2313毫秒 | 无法对文件对象内容进行反序列化 |
| 1000次 | 22375毫秒 | 无法对文件对象内容进行反序列化 |
| 10000次 | 226968毫秒 | 无法对文件对象内容进行反序列化 |

## 8总结

### 序列化

小数量的情况下，bboss比xstream要快3到4倍，bboss产生的序列化数据包比xstream略大一点。

中等数据量或者大数据量情况下bboss比xstream要快3到4倍，bboss产生的序列化数据包为48K基本接近实际数据大小，比xstream产生的包要小一半（xstream为78K）

### 反序列化

小数据量情况下，xstream比bboss快3-4倍

中等数据（xml数据）量或者大数据量（xml数据）情况下，bboss比xstream要快2-3倍

### 数据类型的支持

Bboss和xstream都提供了以下数据类型的序列化/反序列化支持：

基础数据类型；

list，map，set，数组；

复杂对象类型；

异常Exception；

Date和Date[]类型

以及所有上面类型的组合嵌套类型

另外，从测试的情况来看bboss支持对文件对象类型的序列化/反序列化，而xstream不支持。

综上所述，从整体性能、产生的序列化包大小以及对数据类型支持类别三个角度来看，bboss都比XStream要高出很多，如果从技术选型的角度上来看bboss应该是首选。