

xstream反序列化流程分析

测试demo代码如下所示：

```
XStream xStream = new XStream(new StaxDriver());
xStream.fromXML(xml);
```

跟入 `com.thoughtworks.xstream` 的 `fromXML` 函数，接收前端传入的 `String` 形式的xml文本内容，再调用 `fromXML` 函数将 `String` 形式xml转化为 `StringReader` 如下所示：

```
874 public Object fromXML(String xml) { xml: "java.util.PriorityQueue serialization" custom: in: "unserializable-parents" in: "java.util.PriorityQueue" in: "default" in: "size-2" /size: in: /default: in
875 return this.fromXML((Reader) new StringReader(xml)); xml: "java.util.PriorityQueue serialization" custom: in: "unserializable-parents" in: "java.util.PriorityQueue" in: "default" in: "size-2" /size: in:
877 }
```

进入上述调用的 `fromXML` 函数，使用 `hierarchicalStreamDriver.createReader(reader)` 将之前的 `StringReader` 形式内容转换为 `HierarchicalStreamReader` 格式，调用 `MarshallingStrategy` 的 `unmarshal` 方法进行解组获取相应的节点信息。

```
879 public Object fromXML(Reader reader) { reader: StringReader@1412
880 return this.unmarshal(this.hierarchicalStreamDriver.createReader(reader), (Object)null); hierarchicalStreamDriver: StaxDriver@1393 reader: StringReader@1412
881 }
882 }
```

```
932 public Object unmarshal(HierarchicalStreamReader reader, Object root, DataHolder dataHolder) { reader: StaxReader@1411 root: null dataHolder: null
933 try {
934 if (!this.securityInitialized && !this.securityWarningGiven) { securityInitialized: false
935 this.securityWarningGiven = true; securityWarningGiven: true
936 System.err.println("Security framework of XStream not explicitly initialized, using predefined black list on your own risk.");
937 }
938
939 return this.marshallingStrategy.unmarshal(root, reader, dataHolder, this.converterLookup, this.mapper); marshallingStrategy: ReferenceByXPathMarshalling
940 }
```

进入 `com.thoughtworks.xstream.core.AbstractTreeMarshallingStrategy.class` 的 `unmarshal` 方法，创建出 `TreeUnmarshaller`，进入 `TreeUnmarshaller` 的 `start` 函数开始启动解析。

```
19 public Object unmarshal(Object root, HierarchicalStreamReader reader, DataHolder dataHolder, ConverterLookup converterLookup, Mapper mapper) { root: n
20 TreeUnmarshaller context = this.createUnmarshallingContext(root, reader, converterLookup, mapper); context: ReferenceByXPathUnmarshaller@1380 dataHolder: null
21 return context.start(dataHolder); context: ReferenceByXPathUnmarshaller@1380 dataHolder: null
22 }
23 }
```

跟进 `com.thoughtworks.xstream.core.TreeUnmarshaller` 的 `start` 方法，调用 `readClassType` 方法获取 `reader` 中节点名将 `mapper` 中对应的值转换为相应的 `class` 类。调用 `TreeUnmarshaller` 的 `convertAnother` 方法将 `class` 类转换为相应的 `java` 对象。

```
125 public Object start(DataHolder dataHolder) { dataHolder: null
126 this.dataHolder = dataHolder; dataHolder: MapBackedDataHolder@1514 dataHolder: null
127 Class type = hierarchicalStream.readClassType(this.reader, this.mapper); type: "class java.util.PriorityQueue" reader: PathTracingReader@1387 mapper: CachingMapper@1397
128 Object result = this.convertAnother((Object)null, type); type: "class java.util.PriorityQueue"
129 Iterator validations = this.validationList.iterator();
130
131 while(validations.hasNext()) {
132 Rummable rummable = (Rummable)validations.next();
133 rummable.run();
134 }
135
136 return result;
137 }
138 }
```

`convertAnother` 方法中，首先使用 `ConverterLookup` 中的 `lookupConverterForType` 方法寻找对应的 `convert`，根据找到的 `convert` 将 `type` 转换为相对应的 `java` 对象

```
42 public Object convertAnother(Object parent, Class type, Converter converter) { parent: null type: "class java.util.PriorityQueue" converter: SerializableConverter@1379
43 type = this.mapper.defaultImplementationOf(type); mapper: CachingMapper@1397
44 if (converter == null) {
45 converter = this.converterLookup.lookupConverterForType(type); converterLookup: XStream@1395
46 } else if (!converter.canConvert(type)) {
47 ConversionException e = new ConversionException("Explicit selected converter cannot handle type");
48 e.add("name:" + "item-type", type.getName());
49 e.add("name:" + "converter-type", converter.getClass().getName());
50 throw e;
51 }
52
53 return this.convert(parent, type, converter); parent: null type: "class java.util.PriorityQueue" converter: SerializableConverter@1379
54 }
```

跟入 `com.thoughtworks.xstream.core.TreeUnmarshaller` 的 `convert` 调用 `unmarshal` 方法。

```
36  protected Object convert(Object parent, Class type, Converter converter) { parent: null type: "class java.util.PriorityQueue" converter: SerializableConverter@1379
37      this.types.push(type); types: "[class java.util.PriorityQueue]" type: "class java.util.PriorityQueue"
38
39      Object var4;
40      try {
41          var4 = converter.unmarshal(this.reader, unmarshallingContext: this); converter: SerializableConverter@1379 reader: PathTrackingReader@1507
42      } catch (ConversionException var10) {
43          this.addInformationTo(var10, type, converter, parent);
44          throw var10;
45      } catch (RuntimeException var11) {
46          ConversionException conversionException = new ConversionException(var11);
47          this.addInformationTo(conversionException, type, converter, parent);
48          throw conversionException;
49      } finally {
50          this.types.popSilently();
51      }
52  }
```

进入 `com.thoughtworks.xstream.converters.reflection.class` 的 `unmarshaller` 方法, 首先使用 `this.instantiateNewInstance` 创建对应class的instance, 然后调用 `this.doUnmarshal` 遍历节点获取相应class的变量。

```
262
263  public Object unmarshal(HierarchicalStreamReader reader, UnmarshallingContext context) { reader: PathTrackingReader@1507 context: ReferenceByXPathUnmarshaller@1380
264      Object result = this.instantiateNewInstance(reader, context); result: size = 0
265      result = this.doUnmarshal(result, reader, context); result: size = 0 reader: PathTrackingReader@1507 context: ReferenceByXPathUnmarshaller@1380
266      return this.serializationMembers.callReadResolve(result);
267  }
268
269  public Object doUnmarshal(Object result, HierarchicalStreamReader reader, UnmarshallingContext context) {
```

跟入 `com.thoughtworks.xstream.converters.reflection.class` 的 `doUnmarshal` 遍历节点获取class, 绕后使用 `serializationMembers.supportsReadObject` 进行判断获取传入的class是否支持 `ReadObject`, 若支持则调用 `serializationMembers.callReadObject` 进行反序列化操作。

```
279      reader.hasMoreChildren() reader.moveUp() {
280          reader.moveDown();
281          String nodeName = reader.getNodeName(); nodeName: "java.util.PriorityQueue"
282          if (nodeName.equals("unmarshalling:parents")) {
283              super.doUnmarshal(result, reader, context);
284          } else {
285              String classAttribute = HierarchicalStreams.readClassAttribute(reader, this.mapper); classAttribute: null reader: PathTrackingReader@1507
286              if (classAttribute == null) {
287                  currentType[0] = this.mapper.defaultImplementationOf(this.mapper.realClassName()); nodeName: "java.util.PriorityQueue"
288              } else {
289                  currentType[0] = this.mapper.realClass(classAttribute); classAttribute: null
290              }
291
292              if (this.serializationMembers.supportsReadObject(currentType[0], includeBaseClasses: false)) {
293                  CustomObjectInputStream objectInputStream = CustomObjectInputStream.getInstance(context, callback: this.classLoaderReference); objectInputStream: CustomObjectInputStream@1384
294                  this.serializationMembers.callReadObject(currentType[0], result, objectInputStream); currentType: Class[1]@1637 result: size = 0 objectInputStream: CustomObjectInputStream@1384
295                  objectInputStream.popCallback();
296              } else {
297                  try {
298                      callback.defaultReadObject();
299                  } catch (IOException var10) {
300                      throw new StreamException("Cannot read defaults", var10);
301                  }
302              }
303          }
304      }
```

跟入 `callReadObject` 方法, 获取 `readObjectMethod` 调用 `invoke` 进行反序列化;

```
311  Object ex = null; ex: null
312
313  try {
314      Method readObjectMethod = this.getMethod(type, "name": "readObject", new Class[] {ObjectInputStream.class}, includeBaseClasses: false); readObjectMethod: "private void java.com.server.RemoteObject.readObject(java.io.ObjectInputStream) throws java.io.IOException java.lang.ClassNotFoundException: object"
315      readObjectMethod.invoke(object, stream); readObjectMethod: "private void java.com.server.RemoteObject.readObject(java.io.ObjectInputStream) throws java.io.IOException java.lang.ClassNotFoundException: object"
316      catch (IllegalAccessException var5) {
317          ex = new IllegalAccessException("Cannot access method", var5);
318      } catch (InvocationTargetException var7) {
319          ex = new ConversionException("Failed calling method", var7.getTargetException());
320      }
321
322      if (ex != null) {
323          (ErrorHandlerException) add("name": "method", "information": object.getClass().getName() + " readObject()");
324          throw ex;
325      }
326  }
```

总体调用链:

