EsotericSoftware / **yamlbeans**

Java object graphs, to and from YAML automatically

79 commits **1** branch **5** releases **8** contributors

NathanSweet Renamed to setIgnoreUnknownProperties. Latest commit d6d8064 on Oct 5

lib	Moved over from SourceForge.	6 years ago
src/com/esotericsoftware/yamlbeans	Renamed to setIgnoreUnknownProperties.	2 months ago
test	Renamed to setIgnoreUnknownProperties.	2 months ago
.classpath	Moved over from SourceForge.	6 years ago
.gitignore	Added gitignore	2 years ago
.project	Moved over from SourceForge.	6 years ago
CREDITS	Moved over from SourceForge.	6 years ago
LICENSE	Moved over from SourceForge.	6 years ago
README.md	Update README.md	2 years ago
pom.xml	Merge branch 'master' into generics	4 months ago
project.yaml	1.09	10 months ago

README.md

YamlBeans

Please use the YamlBeans discussion group for support.

Overview

YamlBeans makes it easy to serialize and deserialize Java object graphs to and from YAML, a human-friendly data format. Replace XML and properties files with YAML for more expressive power (lists, maps, anchors, etc) and easier hand-editing.

Maven Central: http://repo1.maven.org/maven2/com/esotericsoftware/yamlbeans/yamlbeans/

Basic deserialization

The YamlReader class is used to deserialize YAML to Java objects. The following YAML defines a Map with four entries. The "phone numbers" entry is a List of two items, each of which is a Map.

```
name: Nathan Sweet
age: 28
```

```
address: 4011 16th Ave S
phone numbers:
 - name: Home
   number: 206-555-5138
 - name: Work
   number: 425-555-2306
```

The "read" method reads the next YAML document and deserializes it into HashMaps, ArrayLists, and Strings. Since we know the root object defined in the YAML of our example is a Map, we can cast the object and make use of it.

```
YamlReader reader = new YamlReader(new FileReader("contact.yml"));
Object object = reader.read();
System.out.println(object);
Map map = (Map)object;
System.out.println(map.get("address"));
```

Multiple objects

A stream of YAML can contain more than one YAML document. Each call to YamlReader#read() deserializes the next document into an object. YAML documents are delimited by "---" (this is optional for the first document).

```
name: Nathan Sweet
age: 28
---
name: Some One
age: 25
```

This prints the String "28" then "25":

```
YamlReader reader = new YamlReader(new FileReader("contact.yml"));
while (true) {
    Map contact = reader.read();
    if (contact == null) break;
    System.out.println(contact.get("age"));
}
```

Deserializing other classes

There are two ways to deserialize something other than HashMaps, ArrayLists, and Strings. Imagine this YAML document and Java class:

```
name: Nathan Sweet
age: 28
```

```
public class Contact {
    public String name;
    public int age;
}
```

The "read" method can be passed a class, so the YamlReader knows what it is deserializing:

```
YamlReader reader = new YamlReader(new FileReader("contact.yml"));
```

```
Contact contact = reader.read(Contact.class);
System.out.println(contact.age);
```

The YamlReader creates an instance of the Contact class and sets the "name" and "age" fields. The YamlReader determines the "age" value in the YAML needs to be converted into a int. Deserialization would have failed if the age was not a valid int. The YamlReader can set public fields and bean setter methods.

Instead of telling the YamlReader what type to deserialize, the type can alternatively be specified in the YAML using a tag:

```
!com.example.Contact
name: Nathan Sweet
age: 28
```

Serializing objects

The YamlWriter class is used to serialize Java objects to YAML. The "write" method automatically handles this by recognizing public fields and bean getter methods.

```
Contact contact = new Contact();
contact.name = "Nathan Sweet";
contact.age = 28;
YamlWriter writer = new YamlWriter(new FileWriter("output.yml"));
writer.write(contact);
writer.close();
```

This outputs:

```
!com.example.Contact
name: Nathan Sweet
age: 28
```

The tags are automatically output as needed so that the YamlReader class will be able to reconstruct the object graph. For example, serializing this ArrayList does not output any tag for the list because YamlReader uses an ArrayList by default.

```
List list = new ArrayList();
list.add("moo");
list.add("cow");
```

```
- moo
- cow
```

If the list was a LinkedList, then YamlWriter knows that a tag is needed and outputs:

```
List list = new LinkedList();
list.add("moo");
list.add("cow");
```

```
!java.util.LinkedList
- moo
- cow
```

Note that it is not advisable to subclass Collection or Map. YamlBeans will only serialize the collection or map and its

elements, not any additional fields.

Complex graphs

YamlBeans can serialize any object graph.

```
public class Contact {
    public String name;
    public int age;
    public List phoneNumbers;
}

public class Phone {
    public String name;
    public String number;
}
```

```
friends:
  - !com.example.Contact
    name: Bob
    age: 29
    phoneNumbers:
        - !com.example.Phone
          name: Home
          number: 206-555-1234
        - !com.example.Phone
          name: Work
          number: 206-555-5678
  - !com.example.Contact
    name: Mike
    age: 31
    phoneNumbers:
        - !com.example.Phone
          number: 206-555-4321
enemies:
  - !com.example.Contact
    name: Bill
    phoneNumbers:
        - !com.example.Phone
          name: Cell
          number: 206-555-1234
```

This is a map of lists of contacts, each with a list of phone numbers. Again, the public fields could also have been bean properties.

Tag shortcuts

Tags can be lengthy sometimes and can clutter up the YAML. Alternate tags can be defined for a class and will be used instead of the full class name.

```
YamlWriter writer = new YamlWriter(new FileWriter("output.yml"));
writer.getConfig().setClassTag("contact", Contact.class);
writer.write(contact);
writer.close();
```

The output no longer contains the full classname for the Contact class.

```
!contact
name: Nathan Sweet
age: 28
```

Lists and maps

When reading or writing a List or Map, YamlBeans cannot know what type of objects are supposed to be in the List or Map, so it will write out a tag.

```
!com.example.Contact
name: Bill
    phoneNumbers:
        - !com.example.Phone
          number: 206-555-1234
        - !com.example.Phone
          number: 206-555-5678
        - !com.example.Phone
          number: 206-555-7654
```

This can make the YAML less readable. To improve this, you may define what element type should be expected for a List or Map field on your object.

```
YamlWriter writer = new YamlWriter(new FileWriter("output.yml"));
writer.getConfig().setPropertyElementType(Contact.class, "phoneNumbers", Phone.class);
writer.write(contact);
writer.close();
```

Now YamlBeans knows what to expect for elements of the "phoneNumbers" field, so extra tags will not be output.

```
!com.example.Contact
name: Bill
    phoneNumbers:
        - number: 206-555-1234
        - number: 206-555-5678
        - number: 206-555-7654
```

Setting the element type for a Map field tells YamlBeans what to expect for values in the Map. Keys in a Map are always Strings.

Anchors

When an object graph contains multiple references to the same object, an anchor may be used so that the object only needs to be defined once in the YAML.

```
oldest friend:
    &1 !contact
    name: Bob
    age: 29
best friend: *1
```

In this map, the "oldest friend" and "best friend" keys reference the same object. The YamlReader automatically handles the anchors in the YAML when rebuilding the object graph. By default, the YamlWriter automatically outputs anchors when writing objects.

```
Contact contact = new Contact();
contact.name = "Bob";
contact.age = 29;
Map map = new HashMap();
map.put("oldest friend", contact);
map.put("best friend", contact);
```

Architecture

The YAML tokenizer, parser, and emitter are based on those from the JvYAML project. They have been heavily refactored, bugs fixed, etc. The rest of the JvYAML project was not used because of its complexity. YamlBeans strives for the simplest possible thing that works, with the goal being to make it easy to use the YAML data format with Java.

YamlBeans supports YAML version 1.0 and 1.1.

More info

See the javadocs for various other features available on the YamlConfig class.