Jackson – Bidirectional Relationships

Last modified: February 4, 2017

by Eugen Paraschiv (http://www.baeldung.com/author/eugen/)

Jackson (http://www.baeldung.com/category/jackson/)

1. Overview

In this tutorial, we'll go over the best ways to deal with **bidirectional relationships in Jackson**.

We'll discuss the Jackson JSON infinite recursion problem, then – we'll see how to serialize entities with bidirectional relationships and finally – we will deserialize them.

2. Infinite Recursion

First – let's take a look at the Jackson infinite recursion problem. In the following example we have two entities – "*User*" and "*Item*" – with **a simple one-to-many relationship**:

The "*User*" entity:

```
public class User {
    public int id;
    public String name;
    public List<Item> userItems;
}
```

The "*Item*" entity:

```
public class Item {
    public int id;
    public String itemName;
    public User owner;
}
```

When we try to serialize an instance of "*Item*", Jackson will throw a *JsonMappingException* exception:

```
@Test(expected = JsonMappingException.class)
public void givenBidirectionRelation_whenSerializing_thenException()
  throws JsonProcessingException {

    User user = new User(1, "John");
    Item item = new Item(2, "book", user);
    user.addItem(item);

    new ObjectMapper().writeValueAsString(item);
}
```

The **full exception** is:

```
com.fasterxml.jackson.databind.JsonMappingException:
Infinite recursion (StackOverflowError)
(through reference chain:
org.baeldung.jackson.bidirection.Item["owner"]
->org.baeldung.jackson.bidirection.User["userItems"]
->java.util.ArrayList[0]
->org.baeldung.jackson.bidirection.Item["owner"]
->…..
```

Let's see, over the course of the next few sections – how to solve this problem.

3. Use *@JsonManagedReference*, *@JsonBackReference*

First, let's annotate the relationship with *@JsonManagedReference*, *@JsonBackReference* to allow Jackson to better handle the relation:

Here's the "*User*" entity:

```
public class User {
    public int id;
    public String name;

    @JsonBackReference
    public List<Item> userItems;
}
```

And the "*Item*":

```
public class Item {
    public int id;
    public String itemName;

    @JsonManagedReference
    public User owner;
}
```

Let's now test out the new entities:

```
@Test
public void
  givenBidirectionRelation_whenUsingJacksonReferenceAnnotation_thenCorrect()
  throws JsonProcessingException {

    User user = new User(1, "John");
    Item item = new Item(2, "book", user);
    user.addItem(item);

    String result = new ObjectMapper().writeValueAsString(item);

    assertThat(result, containsString("book"));
    assertThat(result, containsString("John"));
    assertThat(result, not(containsString("userItems")));
}
```

Here is the output of serialization:

```
{
 "id":2,
 "itemName":"book",
 "owner":
    {
        "id":1,
        "name":"John"
    }
}
```

Note that:

- *@JsonManagedReference* is the forward part of reference – the one that gets serialized normally.
- *@JsonBackReference* is the back part of reference – it will be omitted from serialization.

4. Use *@JsonIdentityInfo*

Now – let's see how to help with the serialization of entities with bidirectional relationship using *@JsonIdentityInfo*.

We add the class level annotation to our "*User*" entity:

```
@JsonIdentityInfo(
  generator = ObjectIdGenerators.PropertyGenerator.class,
  property = "id")
public class User { ... }
```

And to the "*Item*" entity:

```java
@JsonIdentityInfo(
  generator = ObjectIdGenerators.PropertyGenerator.class,
  property = "id")
public class Item { ... }
```

Time for the test:

```java
@Test
public void givenBidirectionRelation_whenUsingJsonIdentityInfo_thenCorrect()
  throws JsonProcessingException {

    User user = new User(1, "John");
    Item item = new Item(2, "book", user);
    user.addItem(item);

    String result = new ObjectMapper().writeValueAsString(item);

    assertThat(result, containsString("book"));
    assertThat(result, containsString("John"));
    assertThat(result, containsString("userItems"));
}
```

Here is the output of serialization:

```json
{
 "id":2,
 "itemName":"book",
 "owner":
    {
        "id":1,
        "name":"John",
        "userItems":[2]
    }
}
```

5. Use *@JsonIgnore*

Alternatively, we can also use the *@JsonIgnore* annotation to simply **ignore one of the sides of the relationship**, thus breaking the chain.

In the following example – we will prevent the infinite recursion by ignoring the "*User*" property "*userItems*" from serialization:

Here is "*User*" entity:

```java
public class User {
    public int id;
    public String name;

    @JsonIgnore
    public List<Item> userItems;
}
```

And here is our test:

```java
@Test
public void givenBidirectionRelation_whenUsingJsonIgnore_thenCorrect()
  throws JsonProcessingException {

    User user = new User(1, "John");
    Item item = new Item(2, "book", user);
    user.addItem(item);

    String result = new ObjectMapper().writeValueAsString(item);

    assertThat(result, containsString("book"));
    assertThat(result, containsString("John"));
    assertThat(result, not(containsString("userItems")));
}
```

And here is the output of serialization:

```json
{
 "id":2,
 "itemName":"book",
 "owner":
    {
        "id":1,
        "name":"John"
    }
}
```

6. Use *@JsonView*

We can also use the newer *@JsonView* annotation to exclude one side of the relationship.

In the following example – we use **two JSON Views – *Public* and *Internal*** where *Internal* extends *Public*:

```java
public class Views {
    public static class Public {}

    public static class Internal extends Public {}
}
```

We'll include all *User* and *Item* fields in the *Public* View – **except the *User* field *userItems*** which will be included in the *Internal* View:

Here is our entity "*User*":

```java
public class User {
    @JsonView(Views.Public.class)
    public int id;

    @JsonView(Views.Public.class)
    public String name;

    @JsonView(Views.Internal.class)
    public List<Item> userItems;
}
```

And here is our entity "*Item*":

```java
public class Item {
    @JsonView(Views.Public.class)
    public int id;

    @JsonView(Views.Public.class)
    public String itemName;

    @JsonView(Views.Public.class)
    public User owner;
}
```

When we serialize using the *Public* view, it works correctly – **because we excluded *userItems*** from being serialized:

```java
@Test
public void givenBidirectionRelation_whenUsingPublicJsonView_thenCorrect()
  throws JsonProcessingException {

    User user = new User(1, "John");
    Item item = new Item(2, "book", user);
    user.addItem(item);

    String result = new ObjectMapper().writerWithView(Views.Public.class)
      .writeValueAsString(item);

    assertThat(result, containsString("book"));
    assertThat(result, containsString("John"));
    assertThat(result, not(containsString("userItems")));
}
```

But If we serialize using an *Internal* view, *JsonMappingException* is thrown because all the fields are included:

```java
@Test(expected = JsonMappingException.class)
public void givenBidirectionRelation_whenUsingInternalJsonView_thenException()
  throws JsonProcessingException {

    User user = new User(1, "John");
    Item item = new Item(2, "book", user);
    user.addItem(item);

    new ObjectMapper()
      .writerWithView(Views.Internal.class)
      .writeValueAsString(item);
}
```

7. Use a Custom Serializer

Next – let's see how to serialize entities with bidirectional relationship using a custom serializer.

In the following example – we will use a custom serializer to serialize the "*User*" property "*userItems*":

Here's the "*User*" entity:

```java
public class User {
    public int id;
    public String name;

    @JsonSerialize(using = CustomListSerializer.class)
    public List<Item> userItems;
}
```

And here is the "*CustomListSerializer*":

```java
public class CustomListSerializer extends StdSerializer<List<Item>>{

   public CustomListSerializer() {
        this(null);
    }

    public CustomListSerializer(Class<List> t) {
        super(t);
    }

    @Override
    public void serialize(
      List<Item> items,
      JsonGenerator generator,
      SerializerProvider provider)
      throws IOException, JsonProcessingException {

        List<Integer> ids = new ArrayList<>();
        for (Item item : items) {
            ids.add(item.id);
        }
        generator.writeObject(ids);
    }
}
```

Let's now test out the serializer and see the right kind of output being produced:

```java
@Test
public void givenBidirectionRelation_whenUsingCustomSerializer_thenCorrect()
  throws JsonProcessingException {
    User user = new User(1, "John");
    Item item = new Item(2, "book", user);
    user.addItem(item);

    String result = new ObjectMapper().writeValueAsString(item);

    assertThat(result, containsString("book"));
    assertThat(result, containsString("John"));
    assertThat(result, containsString("userItems"));
}
```

And **the final output** of the serialization with the custom serializer:

```
{
 "id":2,
 "itemName":"book",
 "owner":
    {
        "id":1,
        "name":"John",
        "userItems":[2]
    }
}
```

8. Deserialize with *@JsonIdentityInfo*

Now – let's see how to deserialize entities with bidirectional relationship using *@JsonIdentityInfo*.

Here is the "*User*" entity:

```java
@JsonIdentityInfo(
  generator = ObjectIdGenerators.PropertyGenerator.class,
  property = "id")
public class User { ... }
```

And the "*Item*" entity:

```java
@JsonIdentityInfo(
  generator = ObjectIdGenerators.PropertyGenerator.class,
  property = "id")
public class Item { ... }
```

Let's now write a quick test – starting with some manual JSON data we want to parse and finishing with the correctly constructed entity:

```java
@Test
public void givenBidirectionRelation_whenDeserializingWithIdentity_thenCorrect()
  throws JsonProcessingException, IOException {
    String json =
      "{\"id\":2,\"itemName\":\"book\",\"owner\":{\"id\":1,\"name\":\"John\",\"userItems\":[2]}}";

    ItemWithIdentity item
      = new ObjectMapper().readerFor(ItemWithIdentity.class).readValue(json);

    assertEquals(2, item.id);
    assertEquals("book", item.itemName);
    assertEquals("John", item.owner.name);
}
```

9. Use Custom Deserializer

Finally, let's deserialize the entities with bidirectional relationship using a custom deserializer.

In the following example – we will use custom deserializer to parse the "*User*" property "*userItems*":

Here's "*User*" entity:

```
public class User {
    public int id;
    public String name;

    @JsonDeserialize(using = CustomListDeserializer.class)
    public List<Item> userItems;
}
```

And here is our "*CustomListDeserializer*":

```
public class CustomListDeserializer extends StdDeserializer<List<Item>>{

    public CustomListDeserializer() {
        this(null);
    }

    public CustomListDeserializer(Class<?> vc) {
        super(vc);
    }

    @Override
    public List<Item> deserialize(
      JsonParser jsonparser,
      DeserializationContext context)
      throws IOException, JsonProcessingException {

        return new ArrayList<>();
    }
}
```

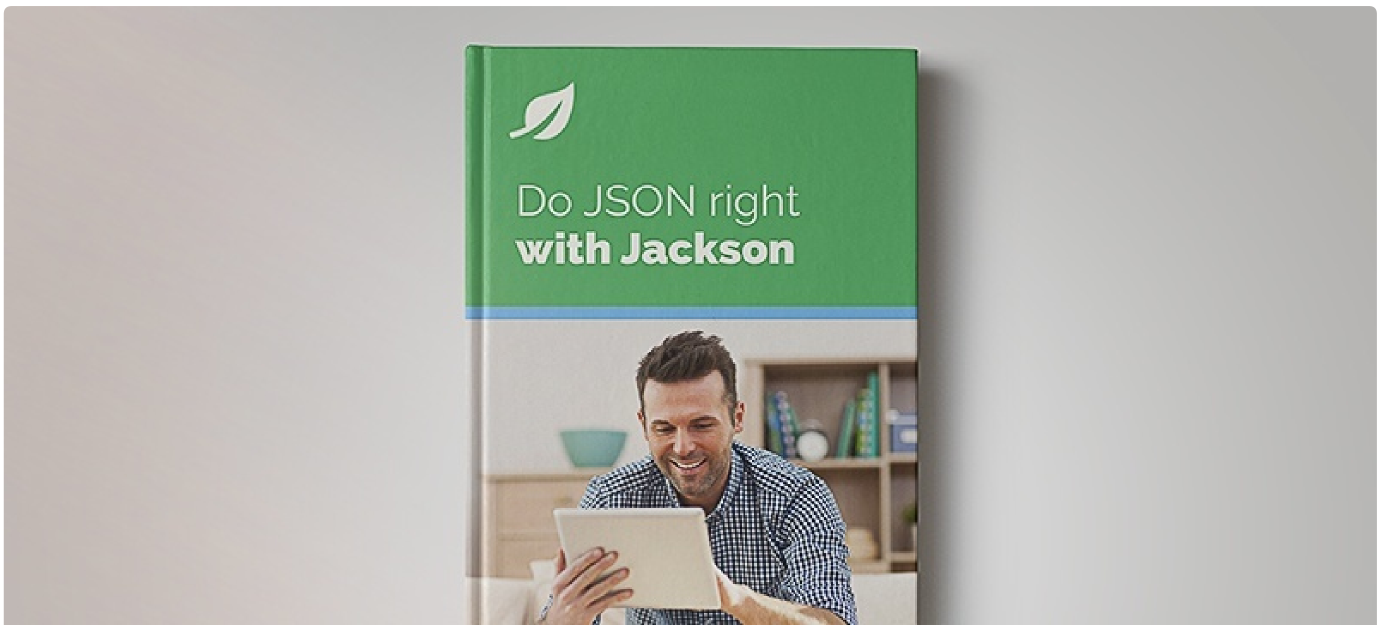
And the simple test:

```
@Test
public void givenBidirectionRelation_whenUsingCustomDeserializer_thenCorrect()
  throws JsonProcessingException, IOException {
    String json =
      "{\"id\":2,\"itemName\":\"book\",\"owner\":{\"id\":1,\"name\":\"John\",\"userItems\":[2]}}";

    Item item = new ObjectMapper().readerFor(Item.class).readValue(json);

    assertEquals(2, item.id);
    assertEquals("book", item.itemName);
    assertEquals("John", item.owner.name);
}
```

10. Conclusion

In this tutorial, we illustrated how to serialize/deserialize entities with bidirectional relationships using Jackson.

The implementation of all these examples and code snippets **can be found in our GitHub project (https://github.com/eugenp/tutorials/tree/master/jackson#readme)** – this is a Maven-based project, so it should be easy to import and run as it is.



Comments for this thread are now closed.

×

51 Comments **Baeldung blog** 1 **Login**

Recommend 2 Share Sort by Best

Jason Glez • 4 months ago
You can also use @JsonIgnoreProperties({ "parameter1","parameter2" }) for example:

public class User {
public int id;
public String name;
@JsonIgnoreProperties({ "owner" })
public List<item> userItems;
}
public class Item {
public int id;
public String itemName;
@JsonIgnoreProperties({ "userItems" })
public User owner;
}
1 • Share ›

Grzegorz Piwowarek Mod → Jason Glez • 4 months ago
But in this case, you are simply ignoring fields and the tricky is part is when you actually can't simply ignore them
• Share ›

Jason Glez → Grzegorz Piwowarek • 4 months ago
In my case I had problems getting the entities from one to many, and many to one, because in many to one with the other implementations the object is removed
• Share ›

Jason Glez → Grzegorz Piwowarek • 4 months ago
But I just ignore the circular reference field, it's really not used at all
• Share ›

Grzegorz Piwowarek Mod → Jason Glez • 4 months ago
Well, if you do not need it, then it's never problematic. Probably the best option would be to separate domain classes from REST responses. A new DTO class would simply not have an unnecessary field
• Share ›

Davi Alves • 2 years ago
You can also use @JsonView. In my case I had a mapping like this:

```
public class ParentDTO {
    private Long id;
    private String name;
    @JsonView(View.ParentWithChild.class)
    private List<childdto> childs;
}
public class ChildDTO {

    private Long id;

    private String name;

    private String description;

    @JsonView(View.ParentWithoutChild.class)
    private ParentDTO parent;

}
```

And on my controller two different methods, one that had to return the ParentDTO with the list of Child's but without Parent, and one that return the Child's and had to return it's Parent as well, but the Parent shouldn't include the Child otherwise it would lead to a StackOverflow error.
Worked like a charm.

1 ^ | v • Share ›

Eugen Paraschiv Mod → Davi Alves • 2 years ago

First, sorry about the late response - I saw the comment, integrated @JsonView, but I must have missed actually responding. So yes - that solution is now included.
Finally - thanks for the properly formatter code. I have been using Disqus for around 3 years now and have over a thousand comments, and this is the first one with correctly formatted code :)
Cheers,
Eugen.

2 ^ | v • Share ›

Rodrigo Estevao Rodrigues • 2 months ago

Excellent post! Thank you a lot!

^ | v • Share ›

Sagar Kapadia • 2 months ago

Hi!
I have a problem with array deserialization when using JsonIdentityInfo. The serialization takes place correctly and the array contains a few Ids where there are cyclic references. However, I cannot figure out how to deserialize the ids into objects. I get an array with some objects and some strings. [I use UUIDs for ids

@JsonIdentityInfo(
generator=ObjectIdGenerators.UUIDGenerator.class,
property="_lineItemExternalId",
scope=LineItemExternal.class
)

The array is serialized as

// edited out

Here, "cee9d79b-77a9-4b3b-a376-ead1d6347d03", and "a15661e1-b4d4-4145-8db8-4e66ad0e4f81" are LineItemExternal ids, which have been serialized completely in the above json. [Removed for brevity]

The code which throws the error is

see more

^ | v • Share ›

Grzegorz Piwowarek Mod → Sagar Kapadia • 2 months ago

Sagar, it's super hard to debug such code by using eyes only. If you could prepare a PR with a failing test case, that would make it much easier to investigate

^ | v • Share ›

Sagar Kapadia → Grzegorz Piwowarek • 2 months ago

How do I upload the test project?

^ | v • Share ›

Grzegorz Piwowarek Mod → Sagar Kapadia • 2 months ago

https://help.github.com/art...

^ | v • Share ›

Sagar Kapadia → Grzegorz Piwowarek • 2 months ago

Grzegorz,
Thanks for the prompt reply.
The repository is at

https://github.com/ks1974in...

The sample input json is in file input.json in the project folder. The test case is in package in.cloudnine.inventory.test;
It is TestSerializationAndDeserialization
The failing test is testWithFile()

Sorry for including so much code. But the previous tests with limited code ALL SUCCEEDED. However, the above test does fail with a class cast exception

Thanks,
Sagar

^ | v • Share ›

Grzegorz Piwowarek Mod → Sagar Kapadia • 2 months ago

Sagar, you are storing there objects of different types. During the first loop run, it deserializes everything correctly and during a second it does not receive a proper object and gets a "c85f6ee8-4f47-4210-9d33-6654a0b2ac3d" instead. This is why you get an exception because it can't deserialize a String into LineItemExternal

^ | v • Share ›

Sagar Kapadia → Grzegorz Piwowarek • 2 months ago

Yes, i know that. But that is the result of serialization. Those are IDs of line items previously encountered by Jackson. If you

uncomment the log statement in other tests,(testArrays()) you will see IDs in the other JSON too. But in other tests there are no problems.
Thanks

Share ›

Sagar Kapadia → Sagar Kapadia • 2 months ago
Any thoughts on this? should I file a bug report?

Share ›

Victor Kalinin • 5 months ago
Hello,
Can't understand in case with @JsonIdentity info, what is '"userItems":[2]'? Where it comes from? Is there any specification Jackson/Rest/.. describing this?

Share ›

Grant Walker • 5 months ago
Thanks for the post. Whats the best option to go with when you have a huge bidirectional ERD. In my project Im trying to Serialize a User (for eg), but it looks kinda like this:
User
- has a Company
- Company has a User (infinite loop)
-Have Appointments
- Apts have a Type
- Types have a list of Appointments (infinite loop)
- Apts have the original User (infinite loop)
It goes on with at least a dozen other objects all interlinked. Id like to keep my code as clean and readable as possible, so I think the @JsonIdentityInfo is my best option.

UPDATE:
Turns out the @JsonIdentityInfo doesnt work as I expected, when serializing a list of users, only the first user is serialized, the rest just have their Id's set. Im probably using it wrong. Any advice would be appreciated.

I'd appreciate any advice,Thanks.
Grant

Share ›

Grzegorz Piwowarek Mod → Grant Walker • 5 months ago
Grant, nothing is wrong with having such a structure. In order to avoid problems with recursive serialization/deserialization, you should check @JsonManagedReference and @JsonBackReference annotations. I hope it helps

Share ›

Grant Walker → Grzegorz Piwowarek • 5 months ago
Thanks for the quick response, so I've added a @JsonManagedReference to the Company attribute in User, and @JsonBackReference to the User attribute in Company.
This works great for retrieving a list of Users (company is populated, without the original user), but when I retrieve a list of Companies now, the Back Ref prevents the Users list from serializing. Is there a way to 'manage it' from both sides?

On a side bar, is there a way to prevent serialization of all collections in an object (without using a custom serializer)?

Share ›

Grzegorz Piwowarek Mod → Grant Walker • 5 months ago
Well, it's a tricky situation and honestly I do not know :) Let me know if you find out the answer for this one.

I do not think that there exists a tool that could allow disabling serialization of all collections but surely you can use @JsonIgnore on all of them manually

Share ›

Grant Walker → Grzegorz Piwowarek • 5 months ago
Hi, I've moved onto using the @JsonManagedReference & @JsonBackReference annotations, and need to know, how do I handle multiple managed and back reference annotation in a single class. Ive set matching values on the managed/back ref pairs but my requests are still failing with a 415 error. my server console prints:
Failed to evaluate Jackson deserialization for type [[simple type, class za.co.itdynamics.planner.domain.CompanyUser]]: com.fasterxml.jackson.databind.JsonMappingException: Multiple back-reference properties with name 'defaultReference'

2016-11-21 14:11:16.387 WARN 9436 --- [nio-8080-exec-8] .c.j.MappingJackson2HttpMessageConverter : Failed to evaluate Jackson deserialization for type [[simple type, class za.co.itdynamics.planner.domain.CompanyUser]]: com.fasterxml.jackson.databind.JsonMappingException: Multiple back-reference properties with name 'defaultReference'

2016-11-21 14:11:16.428 WARN 9436 --- [nio-8080-exec-8] .w.s.m.s.DefaultHandlerExceptionResolver : Resolved exception caused by Handler execution: org.springframework.web.HttpMediaTypeNotSupportedException: Content type 'application/json;charset=UTF-8' not supported

How do i resolve this?
Thanks, Grant

Share ›

Grzegorz Piwowarek Mod → Grant Walker • 5 months ago
If you have multiple @JsonManagedReference and @JsonBackReference, you could try naming them explicitly. They both accept String parameters so you could try helping them match by giving your properties a name like:

@JsonBackReference(value="property1")
@JsonManagedReference(value="property1")

Of course, use more meaningful names :) Let me know if that idea helped :)

Share ›

Grant Walker → Grzegorz Piwowarek • 5 months ago
They are named, both the managed and back reference pair are the same.
I have two Managed References that are objects and two back references that are Sets

Share ›

Grzegorz Piwowarek Mod ➦ Grant Walker • 5 months ago
Can you show the actual code? because this is how you are supposed to resolve this kind of problems but without seeing an actual code it's just a wild guessing
^ | ˅ • Share ›

Grant Walker ➦ Grzegorz Piwowarek • 5 months ago
Sure, I've only included the referenced fields for brevity:

```
public class CompanyUser{
@ManyToOne
@JsonManagedReference(value = "companyUser-user")
private User user;

@ManyToOne
@JsonManagedReference(value = "companyUser-company")
private Company company;

@OneToMany(mappedBy = "companyUser")
@JsonBackReference(value = "appointment-companyUser")
private Set<appointment> appointments;

@OneToMany(mappedBy = "costCentreCustodian")
@JsonBackReference(value = "salesAgencyCostCentres-companyUser")
private Set<salesagencycostcentre> salesAgencyCostCentres;
}
```

see more

^ | ˅ • Share ›

Grzegorz Piwowarek Mod ➦ Grant Walker • 5 months ago
Another question, why are your collections parameterized with a types whose names start with a lowercase? Or is this a mistake?
^ | ˅ • Share ›

Grant Walker ➦ Grzegorz Piwowarek • 5 months ago
I didnt notice, Its a mistake. Its not like that when i edit the post, its must be some kind of markup in the forum.
^ | ˅ • Share ›

Grant Walker ➦ Grant Walker • 5 months ago
@Grzegorz Piwowarek Have you had any luck yet?
^ | ˅ • Share ›

Grzegorz Piwowarek Mod ➦ Grant Walker • 5 months ago
Well, I do not really know the answer right away. What I would suggest you to do, is to start a new thread at Stackoverflow with a complete example and post a link here. If I do not find an answer, someone else surely will :)
^ | ˅ • Share ›

Grzegorz Piwowarek Mod ➦ Grant Walker • 5 months ago
I am speaking today on a conference in Kiev and I am pretty busy so far
^ | ˅ • Share ›

Edwin Quai Hoi • 6 months ago
Hi first of all nice post, but I have come across an edge case that I was hoping you could help with. It is as follows, I have 3 entities A,B and C respectively. A has a many to many relationship with both B and C, while B and C have a many to many relationship with each other. This creates a cycle from A -> B -> C -> A. Each object has a JsonIdentity annotation. Now when I go to serialize A where it is linked to B and C and B and C are linked to each other I get the following error

```
Already had POJO for id (java.lang.String) [[ObjectId:
key=71EA9967-766D-4884-A3AA-421255F3AA42,
type=com.fasterxml.jackson.databind.deser.impl.PropertyBasedObjectIdGenerator,
scope=au.com.genesis.model.Role]] (through reference chain:
au.com.genesis.model.User["roles"]->java.util.HashSet[0]->au.com.genesis.model.Role["id"])
```
^ | ˅ • Share ›

Eugen Paraschiv Mod ➦ Edwin Quai Hoi • 6 months ago
Hey Edwin,
That's an interesting question, but it's unfortunately going to be difficult to answer without looking at the code. The way to go here would be a PR on Github with a failing test - let me know and I'd be happy to have a look.
One quick note is that using annotation usually works up until a certain complexity - once you're past that you can have a look at custom serializers and have more control over the process.
Hope that helps. Cheers,
Eugen.
^ | ˅ • Share ›

Edwin Quai Hoi ➦ Eugen Paraschiv • 6 months ago
Hi Eugen,

Thanks for responding, I have posted code at for you to have a look at. In the meantime I have been experimenting with JSOG to see if it can provide a solution.

Link to git repo -> https://github.com/edwinqua...
^ | ˅ • Share ›

Eugen Paraschiv Mod ➦ Edwin Quai Hoi • 6 months ago
Sure thing, happy to help. Is this new code or the same one I looked at earlier (looks like it's the same repo)? Sure, Jackson isn't the only way to go, but when it comes to complex objects graphs, it's one of the more flexible options.
^ | ˅ • Share ›

Edwin Quai Hoi ➦ Eugen Paraschiv • 6 months ago
Sorry dude, I double posted thinking my first post didn't go through :(.

^ | v • Share ›

Eugen Paraschiv Mod ➦ Edwin Quai Hoi • 6 months ago
No worries. It did, but it was pending because it contained a link :)
^ | v • Share ›

Edwin Quai Hoi ➦ Eugen Paraschiv • 6 months ago
Hi Eugen,

Thanks for the prompt response here is a link to source code that provides an example. I have created a JUnit test to demonstrate the issue. Further investigation tells me that the issue occurs when de-serializing.

https://github.com/edwinqua...
^ | v • Share ›

Eugen Paraschiv Mod ➦ Edwin Quai Hoi • 6 months ago
Hey Edwin,
First, thanks for the link - had a look and opened an issue on your example repo with a few notes.
Hope that helps. Cheers,
Eugen.
^ | v • Share ›

Jodi Depp • 8 months ago
what if all the entities are configured with the xml, What I can do? Is there any config for hbm.xml ?
^ | v • Share ›

Eugen Paraschiv Mod ➦ Jodi Depp • 8 months ago
Hey Jodi,
So, this is primarily about Jackson, not Hibernate, so I'm not sure how the hbm.xml comes into it. Can you elaborate? Cheers,
Eugen.
^ | v • Share ›

Jodi Depp ➦ Eugen Paraschiv • 8 months ago
Hey Eugen,
I meant that I configured the enteties with hbm.xml, not anotations.
Anyway I solved my problem with the help of @JsonIdentityInfo :) Thanks
Cheers)
^ | v • Share ›

Eugen Paraschiv Mod ➦ Jodi Depp • 8 months ago
Glad it's sorted out. Cheers,
Eugen.
^ | v • Share ›

Muthu Raj • a year ago
I have a similar strutcture but it fails with the infinite error when I try to get the data
Parent class
public List<unitdetails> getUnitDetails() {
return unitDetails;
}

Child Class
public Project getProject() {
return project;
}

I have also annotated both the classes as below
@JsonIdentityInfo(generator = ObjectIdGenerators.PropertyGenerator.class, property = "id")

Anyone help
^ | v • Share ›

Eugen Paraschiv Mod ➦ Muthu Raj • a year ago
Hey Mathu,
First - the persistence annotations have nothing to do with Jackson, so you can remove them from the code example (to make it clearer).
Second - since this is very code-focused, go ahead and open an issue over on Github with a full test example that's runnable and I'd be happy to have a look. Cheers,
Eugen.
^ | v • Share ›

Muthu Raj ➦ Eugen Paraschiv • a year ago
Hi I removed the annotations as suggested.
FYI, This is my Jquery method
function getProjectDetails() {
var userID = localStorage.getItem('userID');
var request = $.ajax({
url : '/ABC/project.json',
data : {
userID : userID
},
//dataType : 'json',
type : 'GET',
contentType : 'application/json',
async : true
});

Here is the exception
The server encountered an internal error that prevented it from fulfilling this request.

exception

> net.sf.json.JSONException: There is a cycle in the hierarchy!

^ | v • Share ›

Erik Gollot • 2 years ago

Why don't use an annotation like jpa with opposite attribute ?
What happen if we've two relationships ?

^ | v • Share ›

Eugen Paraschiv Mod ➦ Erik Gollot • 2 years ago

Hey Erik - if you have a bidirectional relationship - without special configuration, Jackson will simply keep processing the graph and end up with a StackOverflow, as a result of the loop. Cheers,
Eugen.

^ | v • Share ›

Erik Gollot ➦ Eugen Paraschiv • 2 years ago

The objective of the presented solution is to deal with bidirectional relationships.
But the solution make me feel that it works only if you've only one relationship

^ | v • Share ›

Davi Alves ➦ Erik Gollot • 2 years ago

Check the @JsonView solution, it has a bidirectional relationship.

^ | v • Share ›

Load more comments

CATEGORIES

SPRING (HTTP://WWW.BAELDUNG.COM/CATEGORY/SPRING/)
REST (HTTP://WWW.BAELDUNG.COM/CATEGORY/REST/)
JAVA (HTTP://WWW.BAELDUNG.COM/CATEGORY/JAVA/)
SECURITY (HTTP://WWW.BAELDUNG.COM/CATEGORY/SECURITY-2/)
PERSISTENCE (HTTP://WWW.BAELDUNG.COM/CATEGORY/PERSISTENCE/)
JACKSON (HTTP://WWW.BAELDUNG.COM/CATEGORY/JACKSON/)
HTTPCLIENT (HTTP://WWW.BAELDUNG.COM/CATEGORY/HTTP/)

ABOUT

ABOUT BAELDUNG (HTTP://WWW.BAELDUNG.COM/ABOUT/)
THE COURSES (HTTP://COURSES.BAELDUNG.COM)
META BAELDUNG (HTTP://META.BAELDUNG.COM/)
THE FULL ARCHIVE (HTTP://WWW.BAELDUNG.COM/FULL_ARCHIVE)
WRITE FOR BAELDUNG (HTTP://WWW.BAELDUNG.COM/CONTRIBUTION-GUIDELINES)
ADVERTISE ON BAELDUNG (HTTPS://WWW.SYNDICATEADS.NET/DIR/BAELDUNG?
UTM_SOURCE=SYNDICATE&UTM_MEDIUM=INPOST&UTM_CAMPAIGN=BAELDUNG)
PRIVACY POLICY (HTTP://WWW.BAELDUNG.COM/PRIVACY-POLICY)
TERMS OF SERVICE (HTTP://WWW.BAELDUNG.COM/TERMS-OF-SERVICE)
CONTACT (HTTP://WWW.BAELDUNG.COM/CONTACT)