JSON serialization frameworks

Java Collections
3 уровень, 4 лекция

ОТКРЫТА

— Привет, дружище!

— Здорово, Диего.

— Я тут смотрю, тебя познакомили с азами сериализации в JSON?

— Почему с азами? Я уже много знаю!

— Святая простота. Да ты и половины не знаешь. Процентов 10 от силы.

— Ух ты. А что там еще осталось?

— Десериализация иерархии объектов (полиморфизм при десериализации), десериализация коллекций, еще много всего. Jackson – большой и мощный фреймворк, а ты с ним, откровенно говоря, едва познакомился.

— Ладно. Тогда расскажи мне о чём-нибудь из этого, а я – послушаю.

Приятно становиться умнее с каждой лекцией!

— Как не помочь другу-роботу? Кто если не я?

Готов? Тогда слушай.

Как ты уже убедился, аннотации используются не только при сериализации, но и при десериализации. На практике для сериализации надо гораздо меньше информации, чем для десериализации. Пример:

Java class	JSON

`class Cat`
`{`
`  public String name = "murka";`
`  public Cat[] cats = new Cat[0];`
`}`
	`{`
	`  "name": "murka",`
	`  "cats": []`
	`}`

`class Cat`
`{`
`  public String name = "murka";`
`  public List<Cat> cats = new ArrayList<>();`
`}`
	`{`
	`  "name": "murka",`
	`  "cats": []`
	`}`

`class Cat`
`{`
`  public String name = "murka";`
`  public List<Cat> cats = new LinkedList<>();`
`}`
	`{`
	`  "name": "murka",`
	`  "cats": []`
	`}`

НАЧАТЬ ОБУЧЕНИЕ

А вот при десериализации неясно, какой объект создать — ArrayList или LinkedList?

— Согласен, если у класса есть поле, и тип поля – это интерфейс (как в случае с public List<Cat> cats), то совсем не ясно, какой именно объект ему присваивать.

— Можно добавить этому полю дополнительные аннотации или оставить jackson-у настройки по умолчанию. Смотри пример:

Конвертация объекта из JSON

```
1  public class Solution {
2      public static void main(String[] args) throws IOException {
3          String jsonString = "{\"name\":\"Murka\",\"cats\":[{\"name\":\"Timka\"},{\"name\":\"Killer\"}
4          ObjectMapper mapper = new ObjectMapper();
5          Cat cat = mapper.readValue(jsonString, Cat.class);
6          System.out.println(cat);
7          System.out.println(cat.cats.getClass());
8      }
9  }
```

Класс, объект которого десериализуется из JSON-формата

```
1  class Cat {
2      public String name;
3      public List<Cat> cats;
4  }
```

Т.е. мы не вмешиваемся и jackson сам определяет классы, которые будут использоваться при десериализации.

— А мне нравится. Удобно. Если конкретная реализация не имеет значения, можно не утруждать себя дополнительными настройками.

Ты еще говорил, что можно воспользоваться аннотациями. Это как?

— Да ничего сложного. Пример:

Конвертация объекта из JSON

```
1  public class Solution {
2      public static void main(String[] args) throws IOException {
3          String jsonString = "{\"name\":\"Murka\",\"cats\":[{\"name\":\"Timka\"},{\"name\":\"Killer\"}
4          ObjectMapper mapper = new ObjectMapper();
5          Cat cat = mapper.readValue(jsonString, Cat.class);
6          System.out.println(cat);
7          System.out.println(cat.cats.getClass());
8      }
9  }
```

Класс, объект которого десериализуется из JSON-формата

```
1  class Cat {
2      public String name;
3      @JsonDeserialize(as = LinkedList.class)
4      public List<Cat> cats;
5  }
```

В строке 3 мы просто добавили аннотацию @JsonDeserialize(as = LinkedList.class), где указали, какую реализацию интерфейса

— Ага. Ясно. Действительно – довольно просто.

— Но и это еще не все. Теперь представь, что тип данных в List тоже интерфейс! Что ты будешь делать?

— У нас есть аннотация и на этот случай?

— Да, причем та же самая. В ней можно указать еще и тип параметр. Выглядеть это будет вот так:

Тип коллекции	Как задать тип данных
List	@JsonDeserialize(contentAs=ValueTypeImpl.class)
Map	@JsonDeserialize(keyAs=KeyTypeImpl.class)

— Круто. Действительно, много нужных аннотаций для разных случаев, о которых заранее и не догадаешься.

— И это еще не все. Сейчас будет самое вкусное. В реальных проектах, классы данных очень часто унаследованы от одного базового класса или интерфейса, который используется практически везде. И вот представь, тебе надо десериализовать структуру данных, которая содержит такие классы. Пример:

Конвертация объекта в JSON

```
public static void main(String[] args) throws IOException
{
 Cat cat = new Cat();
 cat.name = "Murka";
 cat.age = 5;

 Dog dog = new Dog();
 dog.name = "Killer";
 dog.age = 8;
 dog.owner = "Bill Jeferson";

 ArrayList<Pet> pets = new ArrayList<Pet>();
 pets.add(cat);
 pets.add(dog);

 StringWriter writer = new StringWriter();
 ObjectMapper mapper = new ObjectMapper();
 mapper.writeValue(writer, pets);
 System.out.println(writer.toString());
}
```

Класс, объект которого конвертирует в JSON

```
@JsonAutoDetect
class Pet
{
 public String name;
}

@JsonAutoDetect
class Cat extends Pet
{
 public int age;
```

```
@JsonAutoDetect
class Dog extends Pet
{
 public int age;
 public String owner;
}
```

Результат сериализации и вывода на экран:

```
[
 { "name" : "Murka", "age" : 5},
 { "name": "Killer", "age" : 8 , "owner" : "Bill Jeferson"}
]
```

Обрати внимание на результат сериализации.

Мы не сможем провести десериализацию этих данных обратно в Java-объекты, т.к. они фактически неразличимы.

— Немного различимы — у Dog есть поле owner.

— Да, но это поле может быть равно null или вообще пропускаться при сериализации.

— А разве мы не можем задать тип данных с помощью известных нам аннотаций?

— Нет. В одной коллекции после десериализации должны хранится различные объекты типа Cat, Dog и еще пары десятков классов, которые можно унаследовать от Pet.

— И что же можно тут сделать?

— Тут применяют две вещи.

Во-первых, выделяют некоторое поле, которое используется для того, чтобы отличать один тип от другого. Если его нет – его заводят.

Во-вторых, есть специальные аннотации, которые позволяют управлять процессом «полиморфной десериализации». Вот что можно сделать:

Конвертация объекта в JSON

```
public static void main(String[] args) throws IOException
{
 Cat cat = new Cat();
 cat.name = "Murka";
 cat.age = 5;

 Dog dog = new Dog();
 dog.name = "Killer";
 dog.age = 8;
 dog.owner = "Bill Jeferson";

 House house = new House();
 house.pets.add(dog);
 house.pets.add(cat);

 StringWriter writer = new StringWriter();
 ObjectMapper mapper = new ObjectMapper();
 mapper.writeValue(writer, house);
```

Класс, объект которого конвертирует в JSON

```
@JsonTypeInfo(use = JsonTypeInfo.Id.NAME, property="type")
@JsonSubTypes({
 @JsonSubTypes.Type(value=Cat.class, name="cat"),
 @JsonSubTypes.Type(value=Dog.class, name="dog")
})
class Pet
{
 public String name;
}

class Cat extends Pet
{
 public int age;
}

class Dog extends Pet
{
 public int age;
 public String owner;
}

class House
{
 public List<Pet> pets = new ArrayList<>();
}
```

Результат сериализации и вывода на экран:

```
{
 "pets" : [
 {"type" : "dog","name" : "Killer", "age" : 8, "owner" : "Bill Jeferson"},
 {"type" : "cat","name" : "Murka", "age" : 5}
]
}
```

С помощью аннотаций мы указываем, что JSON-представление будет содержать специальное поле type, которое будет хранить значение cat, для класса Cat и значение dog, для класса Dog. Этой информации достаточно, чтобы выполнить корректную десериализацию объекта: при десериализации по значению поля type будет определяться тип объекта, который надо создать.

Иногда в качестве значения поля type используют имя класса (например, «com.example.entity.Cat.class»), но это не очень хорошо. Зачем стороннему приложению, которому мы пересылаем JSON, знать, как называются наши классы? К тому же, классы иногда переименовывают. Использование некоего уникального имени для обозначения конкретного класса – предпочтительнее.

— Круто! А я и не знал, что десериализация такая сложная вещь. И что столько всего можно настраивать.

— Ага. Это действительно новые для тебя вещи, но именно благодаря таким практическим знаниям, ты скоро станешь крутым программистом.

— Амиго – крутой программист. Круто!

— Ладно. Иди, отдыхай.

НАЧАТЬ ОБУЧЕНИЕ

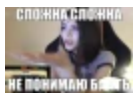
+100

Комментарии (44) популярные новые старые

JavaCoder

Введите текст комментария

Elidriel Уровень 35, Воронеж — 23 марта, 16:20



Ответить +5

Serhio Gonsales Уровень 35, Москва — 26 мая, 17:16

Ответить 0

Elvin Yagudin QA Automation Engineer — 7 февраля, 17:21

Я конечно максимально воздерживаюсь от комментариев, ценю труд JavaRush, но тут честно пригорело. В последнем примере мы сериализуем список объектов Cat и Dog, делаем это хитро через класс House. "Так мило, что мы не просто создали массив, а создали домик для животных" - скажет учащийся, но забота о мнимых животных тут на последнем месте, дело в том что аннотация @JsonTypeInfo не работает через сериализацию классов интерфейса List, раз уж взялись за описание достаточно сложных примеров полиморфной сериализации, этот важный нюанс можно указать.

Ответить +10

LuneFox инженер по сопровождению в **BIFIT** EXPERT — 28 декабря 2021, 15:41

Собрался с силами, сел читать лекции, и тут Диего:

— Ладно. Иди, отдыхай.

Ответить +7

Anton Solovev Уровень 26, Turkey — 29 ноября 2021, 09:30

Нашёл полезный туториал по Jackson аннотациям: Jackson Serialization and Deserialization

Ответить +3

Greatsky future developer в **future developer** — 10 января, 19:29

что то не то с ссылкой

Ответить 0

Anonymous #2631266 Уровень 41, Одесса — 19 мая 2021, 16:01

In Java, annotations are a kind of metadata that provide information about a program. They can mark classes, methods, fields, variables, and other elements of a program.

Annotations can be used for different purposes:

to provide information for the compiler;
to provide information for development tools to generate code, XML files and so forth;
to provide information for frameworks and libraries at runtime.

Ответить 0

Ira Tsygarova Уровень 36, Санкт-Петербург, Россия — 15 февраля 2021, 22:42

явно говорим, **что** нам нужно, а иногда и **как** нам это нужно сделать.

Это очень похоже на то, как мы имплиментили интерфейсы AutoCloseable, Serializable и т.д. Тем самым говоря java-машине, что "Вот смотри, что класс наш должен уметь"

Ответить +13

Valua Sinicyn Уровень 41, Харьков, Украина 7 февраля 2021, 14:44

Ничего не понятно, работаем дальше.

Ответить +1

Андрей Уровень 41, Самара 21 января 2021, 13:01

Ни уровня по аннотациям, ни уровня по рефлексии. А последние темы только на них и построены. Минус респект за такое

Ответить +5

Pig Man Главная свинья в **Свинарнике** 7 февраля 2021, 01:46

Потому что сначала в многопоточность идти нужно

Ответить 0

Valua Sinicyn Уровень 41, Харьков, Украина 7 февраля 2021, 14:40

По аннотациям - да. По рефлексии было.

Ответить 0

Сергеев Алексей Уровень 28, Москва, Россия 8 февраля 2021, 17:07

Аннотации в Философии джава у Экеля уж слишком сложны) Думаю пока нужно просто ими пользоваться и не думать, как они работают)

Ответить +1

Anonymous #2489173 Уровень 35 28 марта 2021, 16:29

Потому что сначала в многопоточность идти нужно

Ну тогда не надо было выбор давать пользователям.

Ответить +7

Pig Man Главная свинья в **Свинарнике** 28 марта 2021, 20:52

Я ничего не давал, вопрос к авторам. Изначально выбора не было, потом разрешили выбирать после 2 квеста

Ответить 0

Vadim Zakirov Java Developer 6 июля 2021, 23:21

кто-то говорит иди в Коллекции, кто-то в многопоточности, где правда то

Ответить 0

Blame QA Automation Engineer в **Mission:Luna** 3 сентября 2021, 08:00

в коллекциях без многопоточности как то совсем тяжо

Ответить 0

PaiMei in J# Grand Master в **Eagles' Claw** 21 октября 2021, 12:17

ЭЭЭммм, в многопоточности, рефлексии, как таковой, нет ♂

Ответить +2

Богдан Зінченко Frontend Developer в **iSolutions** 27 декабря 2020, 19:57

Почему, если сеариализовывать не house, a house.pets, то поле type не появляется в элементах массива?

Ответить 0

Андрей Шубин Уровень 28, Москва, Россия 9 сентября 2020, 21:44

Чет фигня какая-то написана. Не работают эти подтипы похоже. %)

Ответить 0

Artem K. Уровень 30, Москва 10 ноября 2020, 19:06

Все работает, проверил

Ответить +1