# JSON概念

1、java对象转json格式就是两种方式，一种是框架，一种是用一些开源的封装好的json jar包。

首先说下如何利用框架吧，框架本人熟悉的就两种，一种是strtus2，strtus2无非就是将要转化为json的数据，放到我们设置的bean中，然后通过值栈可以自动转化为json格式的数据，springmvc就是利用@ResponseBody这个注解，将数据转化为json，方式同strtus2一样。

2、封装好的json 包，本人接触过的有两个，一个JSONObject，这个坑死了，对于复杂的json bean转化会报错，所以我放弃了，后来改用了阿里巴巴的fastjson，目前这个还没有发现有什么问题，需要的java包一个：fastjson.jar包

# fastjson API

## 基本概念

Fastjson API入口类是com.alibaba.fastjson.JSON，常用的序列化操作都可以在JSON类上的静态方法直接完成，JSONObject类继承自JSON类。

说明：

SerializeWriter：相当于StringBuffer

JSONArray：相当于List<Object>

JSONObject：相当于Map<String, Object>

JSON反序列化没有真正数组，本质类型都是List<Object>

## JSON类

## JSONObject类

### 构造函数

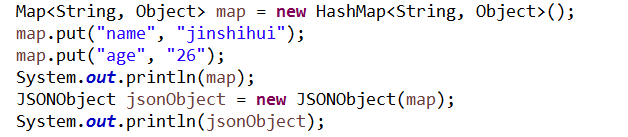
#### public JSONObject(boolean ordered)

#### public JSONObject(int initialCapacity)

#### JSONObject(Map<String, Object> map);

//将map集合转换为JSONObject对象

示例：



输出：

{age=26, name=jinshihui}

{"age":"26","name":"jinshihui"}

### 方法

#### public static final Object parse(String text);

把JSON文本字符串parse为JSONObject或者JSONArray

#### public static final JSONObject parseObject(String text)；

//把JSON文本字符串解析成JSONObject对象（说明：json字符串中的顺序发生变化）

#### public static final <T> T parseObject(String text, Class<T> clazz);

把JSON文本字符串parse为JavaBean

#### public static final JSONArray parseArray(String text);

//把JSON字符串parse成JSONArray

#### public static final <T> List<T> parseArray(String text, Class<T> clazz);

把JSON字符串parse成List集合

#### public static final String toJSONString(Object object);

将对象序列化为JSON文本字符串

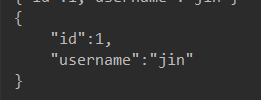
示例：



#### public static final String toJSONString(Object object, boolean prettyFormat);

//将对象序列化为带格式的JSON文本

示例：



#### public static final Object toJSON(Object javaObject);

//将Java对象转换为JSONObject或者JSONArray对象

#### public Object get(String name)

//获取指定名称的数据值

#### public JSONObject fluentPut(String key, Object value)

用于向 JSON 对象中添加键值对，并以链式调用的方式进行操作。

## SerializeWriter类

## 注解

### @JSONField

#### 基本概念

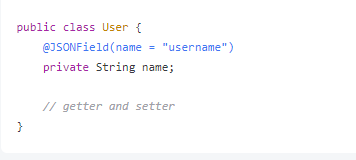
用于指定 JSON 序列化和反序列化时的属性映射和配置。总之，@JSONField 注解提供了很多灵活的选项，可以通过注解为 Java 对象与 JSON 之间的转换过程指定映射关系、格式化规则和顺序等。

#### 作用

##### 1、属性重命名

通过 name 属性，可以将 Java 对象中的属性名映射到 JSON 中的不同字段名。

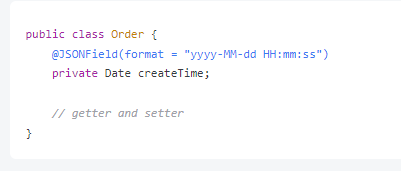
示例：



在上述示例中，@JSONField(name = "username") 注解将 Java 对象中的 name 属性映射为 JSON 中的 username 字段。

##### 2、序列化和反序列化格式控制

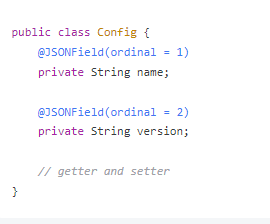
通过 format 属性，可以定义日期、时间、数值等类型在序列化和反序列化过程中的格式。



在上述示例中，@JSONField(format = "yyyy-MM-dd HH:mm:ss") 注解指定了 createTime 属性在序列化为 JSON 时的日期格式。

##### 3、指定属性在序列化或反序列化时的顺序

通过 ordinal 属性，可以指定属性在 JSON 对象中的顺序。较小的值将排在前面。



在上述示例中，@JSONField(ordinal = 1) 注解指定了 name 属性在 JSON 对象中的顺序位于第一个。

##### 4、其他配置项

还有一些其他常用的配置项，如 serialize 和 deserialize 可以控制属性的序列化和反序列化行为；serialzeFeatures 和 deserializeFeatures 可以启用或禁用某些特性等。

## 示例

### JavaBean转Json

@Test

public void BeanToJson(){

Person p1=new Person();

p1.setId(0);

p1.setName("nike");

String str = JSON.toJSONString(p1);

System.out.println(str);

}

### List转Json

@Test

public void listToJson(){

List <Person>list=new ArrayList<>();

Person p1=new Person();

p1.setId(0);

p1.setName("nike");

Person p2=new Person();

p2.setId(1);

p2.setName("linda");

list.add(p2);

list.add(p1);

String str = JSON.toJSONString(list);

System.out.println(str);

}

### Map转Json

@Test

public void mapToJson(){

Map<String, Object> map=new HashMap<String, Object>();

List <Person>list=new ArrayList<>();

Person p1=new Person();

p1.setId(0);

p1.setName("nike");

Person p2=new Person();

p2.setId(1);

p2.setName("linda");

list.add(p2);

list.add(p1);

map.put("list", list);

map.put("total", 1);

String str = JSON.toJSONString(map);

System.out.println(str);

}

# Gson

## 基本概念

Gson 是google解析Json的一个开源框架,同类的框架fastJson,JackJson等等。

## Gson API

### 方法

#### String toJson(Object src)

将java对象转换为json字符串

#### Object fromJson(String,Class)

将json字符串转换为javabean对象

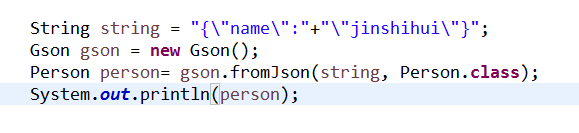
## JavaBean转换为json串



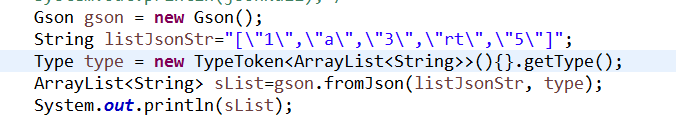
### List Map转换为json串



### Json字符串转JavaBean



### Json字符串转List



### JsonPrimitive

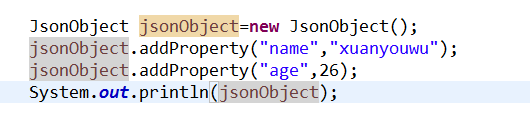
 JsonPrimitive非常有意思,我们知道如果json转换成字符串可能包含引号的转义，但是通过JsonPrimative我们可以获得为转义的字符串。



### 创建JsonObject

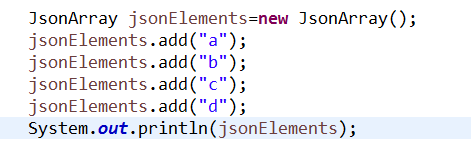
通过addPropert(key,value)可以向JsonObject对象中添加字段 跟HashMap类似。

示例：



输出：{"name":"xuanyouwu","age":26}

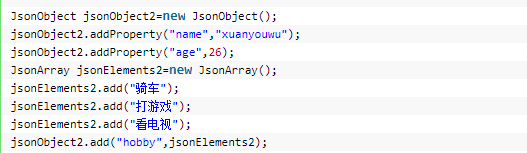
### 创建JsonArray



输出：["a","b","c","d"]

### JsonObject 嵌套数组或者说嵌套JsonArray

通过JsonObject的add(key,JsonElement)可以为jsonObject 添加一个数组的字段



输出：

{"name":"jinshihui","age":26,"hobby":["骑车","打游戏","看电视"]}

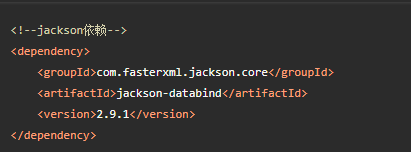
### Gson注解

# Jackson

## 基本概念

在日常开发过程中，用于处理json和xml格式化的类库中，用的比较广泛的就属Jackson了，而且Spring MVC 框架默认的json解析器也是Jackson。对于Jackson的1.x版本的包名为org.codehaus.jackson，当版本升级到2.x以上时，包名变为com.fasterxml.jackson。

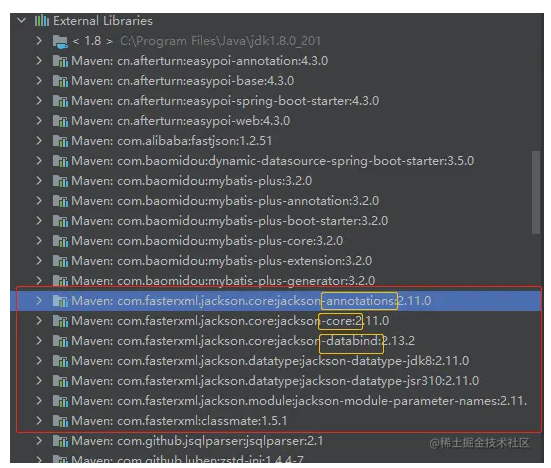
## jackson依赖



注意：如果你是springboot项目，并且有引入spring-boot-starter-web这个starter，你就不需要引入上方jackson的依赖包了，因为引入web这个starter自带就带有jackson依赖包，所以大家不要重复导包。

## 核心模块

引入Jackson包，我们可以看到，在依赖中分别有如下三类包，大家请看



### jackson-core

核心包，它提供基于"流模式"解析的相关 API，它包括JsonPaser和JsonGenerator。Jackson 内部实现正是通过高性能的流模式 API 的 JsonGenerator 和 JsonParser 来生成和解析 json。

### jackson-annotations

注解包，提供标准注解功能。

### jackson-databind

数据绑定包， 提供基于"对象绑定" 解析的相关 API （ ObjectMapper ） 和"树模型" 解析的相关 API （JsonNode）；基于"对象绑定" 解析的 API 和"树模型"解析的 API 依赖基于"流模式"解析的 API。

## 类

### ObjectMapper

#### 基本概念

ObjectMapper 类是 Jackson 库中的核心类之一，用于实现 Java 对象和 JSON 数据之间的相互转换。它提供了各种方法来处理序列化（将 Java 对象转换为 JSON 数据）和反序列化（将 JSON 数据转换为 Java 对象）操作。

#### 方法

##### public String writeValueAsString(Object value)

将Java对象转换为json字符串

##### public <T> T readValue(String content, Class<T> valueType)

将json字符串反序列化为类

##### public ObjectMapper setDateFormat(DateFormat dateFormat)

##### public ObjectMapper configure(DeserializationFeature f, boolean state)

##### public JsonNode readTree(String content)

#### 使用示例

##### 1、创建ObjectMapper 对象

你可以通过创建 ObjectMapper 对象来开始使用 Jackson 库。通常，可以直接创建一个全局的 ObjectMapper 实例，以便在整个应用程序中重复使用。

##### 2、序列化

使用 ObjectMapper 的 writeValueAsString() 方法可将 Java 对象序列化为 JSON 字符串



还可以使用其他方法，如 writeValue()，将 JSON 写入 OutputStream 或 File 中。

##### 3、反序列化

使用 ObjectMapper 的 readValue() 方法可将 JSON 数据反序列化为 Java 对象：

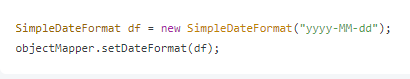


MyClass.class 是目标 Java 类型。

##### 4、配置ObjectMapper

ObjectMapper 提供了许多配置选项，以满足不同的需求。一些常见的配置包括：

###### 设置日期格式



###### 忽略为空的字段



###### 序列化空值



### JsonNodeFactory

#### 基本概念

### JsonNode

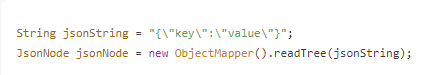
#### 基本概念

JsonNode 是 Jackson 库中的一个抽象类，用于表示 JSON 数据的节点。它是一个通用的数据结构，可以表示 JSON 中的各种类型，包括对象、数组、字符串、数字、布尔值等。

JsonNode 提供了一组通用的方法来处理和操作 JSON 数据，使得我们可以在不关心具体类型的情况下对 JSON 数据进行访问和操作。它可以用于解析和构建复杂的 JSON 数据结构，并提供了丰富的方法来处理不同的节点类型。

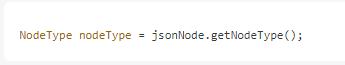
#### 用法

##### 1、解析 JSON 字符串为 JsonNode 对象



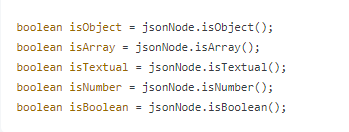
使用 readTree() 方法将 JSON 字符串解析为 JsonNode 对象。

##### 2、获取节点类型



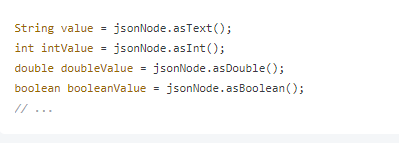
使用 getNodeType() 方法获取节点的类型，返回的是一个枚举值，表示节点的类型，例如 OBJECT、ARRAY、STRING、NUMBER、BOOLEAN 等。

##### 3、检查节点类型



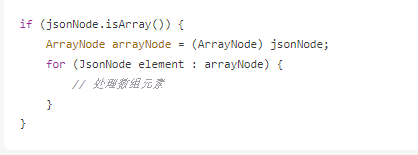
使用 isObject()、isArray()、isTextual()、isNumber()、isBoolean() 等方法检查节点的具体类型。

##### 4、获取节点的值



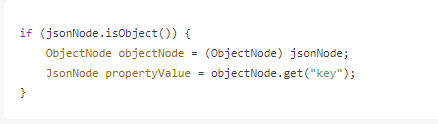
使用 asText()、asInt()、asDouble()、asBoolean() 等方法获取节点的具体值。

##### 5、迭代数组节点



如果节点是数组类型，可以将其强制转换为 ArrayNode 并使用迭代器遍历数组元素。

##### 6、获取对象节点的属性



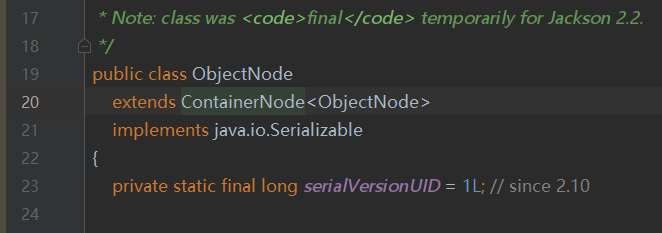
如果节点是对象类型，可以将其强制转换为 ObjectNode 并使用 get() 方法获取指定属性的值。

### ObjectNode

#### 基本概念

ObjectNode 是 Jackson 库中的一个类，它是 JsonNode 的子类，用于表示 JSON 对象。ObjectNode 类提供了一些方法，使得我们可以方便地操作和处理 JSON 对象。

#### 源码



#### 方法

##### public ObjectNode put(String fieldName, String v)

##### public JsonNode get(String fieldName)

##### public JsonNode remove(String fieldName)

##### public boolean has(String fieldName)

##### public Iterator<String> fieldNames()

获取 ObjectNode 中所有属性名的迭代器。

#### 用法

##### 1、创建空的ObjectNode 对象

ObjectNode objectNode = JsonNodeFactory.instance.objectNode();

##### 2、添加属性到 ObjectNode 对象

objectNode.put("key", "value");

##### 3、获取属性值

String value = objectNode.get("key").asText();

使用 get() 方法获取指定键的值，可以使用 asText() 方法将属性值转换为文本形式。

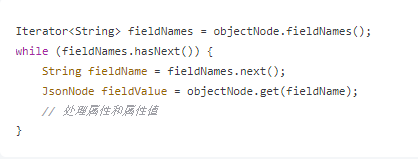
##### 4、检查属性是否存在

boolean hasKey = objectNode.has("key");

##### 5、删除属性

objectNode.remove("key");

##### 6、迭代属性



使用 fieldNames() 方法可以获取 ObjectNode 中所有属性名的迭代器，然后可以通过迭代器遍历所有属性和属性值。

### PropertyNamingStrategy

#### 基本概念

PropertyNamingStrategy 是 Jackson 库中的一个策略接口，用于指定 Java 对象属性与 JSON 字段之间的命名转换规则。通过实现和配置 PropertyNamingStrategy，你可以自定义属性命名策略，以控制序列化和反序列化过程中的字段命名。

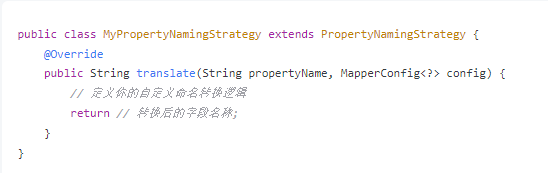
#### 默认命名策略

在默认情况下，Jackson 库使用 PropertyNamingStrategy.LOWER\_CAMEL\_CASE 作为命名策略。它将驼峰命名法应用于 Java 对象的属性，并将其转换为小写字母开头的下划线分隔的字段命名方式。例如，firstName 属性将被序列化为 first\_name 字段。

#### 自定义命名策略

如果默认的命名策略不符合你的需求，你可以自定义 PropertyNamingStrategy 的实现类，并将其配置到 ObjectMapper 中。你可以通过实现 PropertyNamingStrategy 接口的 translate() 方法来自定义属性命名转换规则。这个方法接收两个参数：propertyName（Java 对象属性名）和 config（包含一些属性信息的上下文对象），并返回转换后的字段名称。

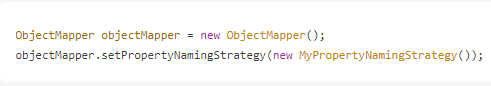
示例：



在上述示例中，你可以根据需要实现自己的命名转换逻辑，并返回转换后的字段名称。

#### 配置 ObjectMapper 使用自定义命名策略

一旦你实现了自定义的命名策略，你可以将其配置到 ObjectMapper 中：



然后，ObjectMapper 将使用你定义的命名策略来进行属性命名的转换。

#### 预定义的 PropertyNamingStrategy 实现

Jackson 库还提供了一些预定义的 PropertyNamingStrategy 实现，以方便你使用常见的命名转换规则。例如：

LOWER\_CASE：将属性名称转换为小写字母开头的下划线分隔命名方式（如 first\_name）。

UPPER\_CAMEL\_CASE：将属性名称转换为驼峰命名法（如 FirstName）。

KEBAB\_CASE：将属性名称转换为连字符分隔命名方式（如 first-name）。

你可以直接使用这些预定义的命名策略，而不需要实现自定义的策略。

### PropertyNamingStrategies

#### 基本概念

这个类是从2.12版本开始才有的。

### JsonSerializer

### JacksonProperties

#### 基本概念

用于配置 Jackson 库相关功能的属性类。

#### 功能

##### 1、修改 Jackson 库的默认配置

JacksonProperties 类的属性可以用于修改 Jackson 库的默认配置。通过设置不同的属性值，可以定制序列化、反序列化和其他 Jackson 功能的行为。一些常见的属性包括：

date-format：指定日期格式，默认为 yyyy-MM-dd HH:mm:ss。

time-zone：指定时区，默认为 Java 虚拟机的默认时区。

default-property-inclusion：指定默认的属性包含规则，例如 always、non\_null、non\_absent 等。

##### 2、开启Jackson 的特性

JacksonProperties 类的属性可以用于开启或关闭 Jackson 库的不同特性。通过设置不同的属性值，可以控制是否启用或禁用某些特性。一些常见的属性包括：

fail-on-unknown-properties：指定是否在遇到未知属性时抛出异常，默认为 true

fail-on-empty-beans：指定是否在序列化空对象时抛出异常，默认为 false。

##### 3、自定义Jackson模块

JacksonProperties 类的属性可以用于配置自定义的 Jackson 模块。通过设置不同的属性值，可以将自定义的 Jackson 模块注册到 Spring Boot 的 Jackson 配置中。一些常见的属性包括：

modules：指定要注册的自定义 Jackson 模块类。

##### 4、其他配置项

除了上述功能，JacksonProperties 类还提供了其他一些配置项，用于控制 Jackson 库的行为。例如，可以通过设置 locale 属性来指定序列化和反序列化时使用的区域设置。

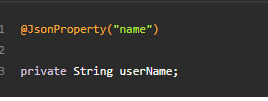
### TypeReference

## 注解

### @JsonProperty

指定属性在序列化和反序列化过程中的名称。

示例：



#### @JsonAlias注解和@JsonProperty注解的区别

1、使用方式

@JsonAlias 注解可以直接应用于字段上，用于指定一个或多个字段的别名。使用时需要将别名定义为字符串数组。

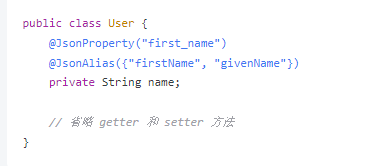
@JsonProperty 注解通常应用于字段上，用于指定字段在序列化和反序列化过程中对应的 JSON 属性的名称。使用时需要将属性名称作为字符串传递给注解。

2、功能

@JsonAlias 注解用于定义多个别名，以便能够处理具有不同名称的 JSON 数据字段。它是为了处理字段名称不完全匹配的情况而设计的。当 JSON 数据中的字段与 Java 对象的字段名称不匹配时，它可以识别别名并进行映射。

@JsonProperty 注解用于定义字段在序列化和反序列化过程中对应的 JSON 属性的名称。它用于明确指定 JSON 字段与 Java 对象字段之间的映射关系。

示例：



在上述示例中，User 类的 name 字段使用了 @JsonProperty 注解，指定在序列化和反序列化过程中对应的 JSON 属性名称为 first\_name。同时，它还使用了 @JsonAlias 注解，定义了两个别名：firstName 和 givenName。

这意味着当你进行反序列化时，无论 JSON 数据中使用 first\_name、firstName 还是 givenName，都会正确地映射到 Java 对象的 name 字段。而在序列化时，Java 对象的 name 字段的值将被写入 JSON 数据的 first\_name 字段中。

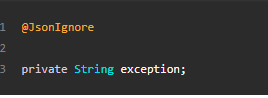
总结起来，@JsonAlias 注解用于处理多个字段别名，而 @JsonProperty 注解用于明确指定 JSON 字段和 Java 对象字段之间的映射关系。这两个注解在不同的场景中有其各自的重要性和作用。

### @JsonIgnore

#### 基本概念

此注解用于属性上，作用是进行JSON操作时忽略该属性。

示例：



### @JsonPropertyOrder

用法：此注解用于用于类；作用是指定属性在序列化时 json 中的顺序。

示例：



### @JsonIgnoreProperties

#### 基本概念

用于在序列化（将对象转换为 JSON）和反序列化（将 JSON 转换为对象）过程中忽略指定的属性。

在 Java 类与 JSON 之间的相互转换中，常常会遇到某些字段不需要进行序列化或反序列化的情况。例如某个字段包含敏感信息或者与业务逻辑无关，可以选择忽略这些字段，不进行序列化或反序列化操作。使用 @JsonIgnoreProperties 注解后，Jackson 库在进行序列化和反序列化时，会忽略指定的属性，不进行处理。从而在生成的 JSON 或者转换回的对象中忽略了被标记的属性。这个注解在实际开发中广泛应用，可以灵活地控制序列化和反序列化的行为，使得生成的 JSON 数据或者转换回的对象符合预期的需求。

#### 属性

##### ignoreUnknown

用于在反序列化过程中忽略未知的属性。

在进行 JSON 反序列化时，有时候源 JSON 数据中可能包含目标类中没有对应的属性。默认情况下当遇到未知属性时，Jackson 库会抛出异常。然而，有些情况下我们可能希望忽略这些未知属性，并继续进行反序列化操作。通过设置 ignoreUnknown = true，可以使 @JsonIgnoreProperties 注解在反序列化过程中忽略未知的属性。这样即使源 JSON 数据中包含了目标类中没有的属性，Jackson 也不会抛出异常，而是简单地忽略这些未知属性。

#### 示例

### @JsonInclude

控制在序列化过程中包含哪些属性

### @JsonSerialize

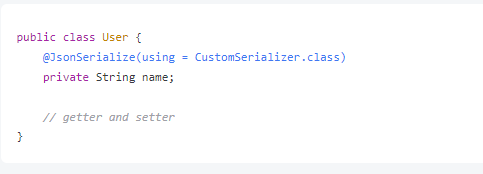
#### 基本概念

用于指定属性在序列化过程中的自定义处理。

#### 作用

##### 1、自定义序列化器

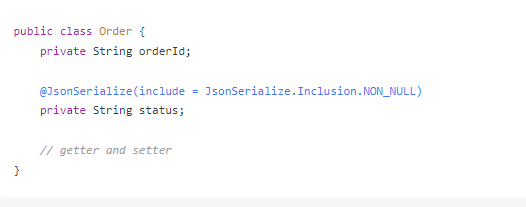
通过 using 属性，可以指定一个自定义的序列化器类，实现对属性的特殊序列化逻辑。



在上述示例中，@JsonSerialize(using = CustomSerializer.class) 注解用自定义的序列化器 CustomSerializer 对 name 属性进行序列化处理。

##### 2、排除属性

通过 include 属性，可以排除某些属性不进行序列化。



在上述示例中，@JsonSerialize(include = JsonSerialize.Inclusion.NON\_NULL) 注解指定了 status 属性，在序列化过程中仅当其值不为 null 时才进行序列化输出。

##### 3、其他配置项

还有一些其他常用的配置项，如 as 可以指定属性的类型；nullsUsing 可以指定序列化 null 值时使用的默认值等。

总之，通过 @JsonSerialize 注解，我们可以为属性在序列化过程中指定自定义的处理方式，例如使用自定义的序列化器、排除特定属性等。这样可以灵活地控制属性的输出形式，满足不同的序列化需求。

### @JsonRawValue

### @JsonCreator

### @JsonAlias

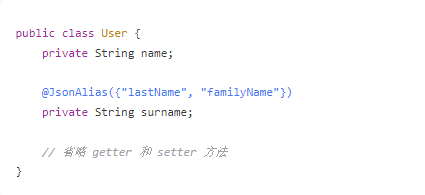
#### 基本概念

用于指定在反序列化或序列化过程中，JSON 字段的别名。它允许你在 Java 对象和 JSON 数据之间建立映射关系，使得能够处理不同名称的字段。具体来说，@JsonAlias 的作用有以下几点：

1、别名识别：当 JSON 数据中的字段名与你的 Java 对象的字段名不完全匹配时，通过使用 @JsonAlias 注解可以使得 Jackson 能够识别这个别名，并将相应的值赋给对应的 Java 对象字段。

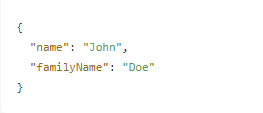
2、多个别名：如果一个字段可能有多个不同的名称，你可以使用多个 @JsonAlias 注解来定义多个别名。这样，无论 JSON 数据中使用哪个别名，都能正确地映射到 Java 对象的相应字段。

#### 示例



在上述示例中，User 类中的 surname 字段使用了 @JsonAlias 注解，并定义了两个别名：lastName 和 familyName。

假设你有以下 JSON 数据：



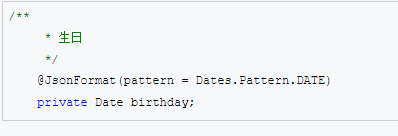
当你使用 Jackson 进行反序列化时，familyName 字段的值将会正确地映射到 User 对象的 surname 字段中。同样地，当你进行序列化操作时，Java 对象的 surname 字段的值将会被写入 JSON 数据的 familyName 字段中。总之，@JsonAlias 注解允许你定义多个别名，以实现 Java 对象和 JSON 数据之间的映射。这在处理字段名称不统一或需要兼容多个版本的数据时特别有用。

### @JsonFormat

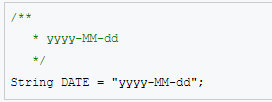
#### 基本概念

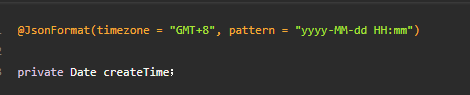
@JsonFormat注解是一个时间格式化注解，比如我们存储在mysql中的数据是date类型的，当我们读取出来封装在实体类中的时候，就会变成英文时间格式，而不是yyyy-MM-dd HH:mm:ss这样的中文时间，因此我们需要用到JsonFormat注解来格式化我们的时间。一般用于一个字段命名上面。

#### 使用方法



这里Dates.Pattern.DATE是枚举类型取值，对应值为：





要注意的是，它只会在类似@ResponseBody返回json数据的时候才会返回格式化的yyyy-MM-dd HH:mm:ss时间，你直接使用System.out.println()输出的话，仍然是类似“Fri Dec 01 21:05:20 CST 2017”这样的时间样式。

## Jackson转换LocalDateTime类型报错问题

### 基本说明

jackson 只支持Date类型，默认情况下不支持java.time.LocalDateTime，SpringBoot默认使用Jackson解析Json字符串。

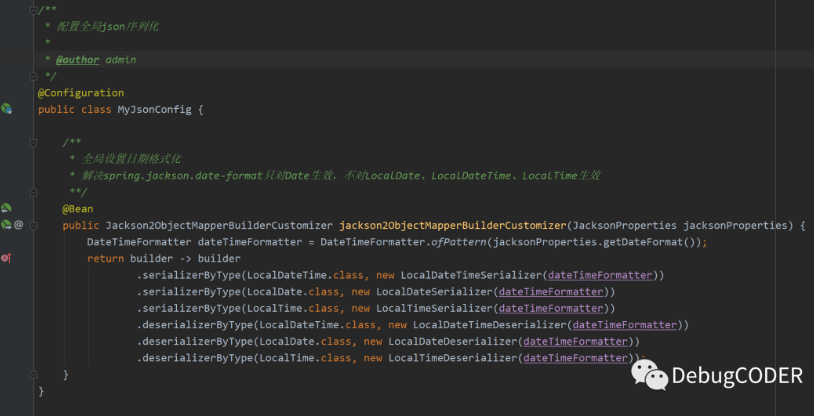
### 解决方法

#### 1、pom里新增依赖

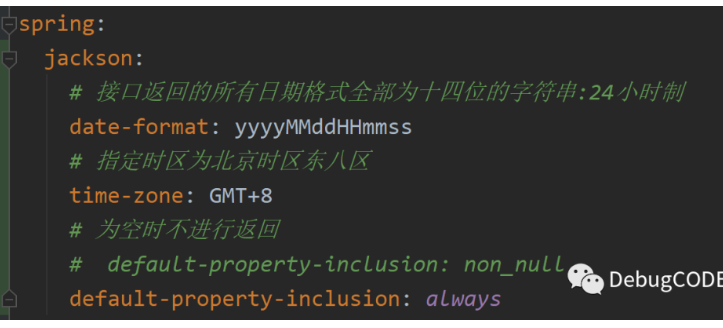
增加了这个jar依赖jackson-core-asl，可以直接用jar310提供的序列化方法，不需要自定义序列化了，比较方便。其他2个依赖包是spring自带的。



#### 2、自定义序列化（全局配置）



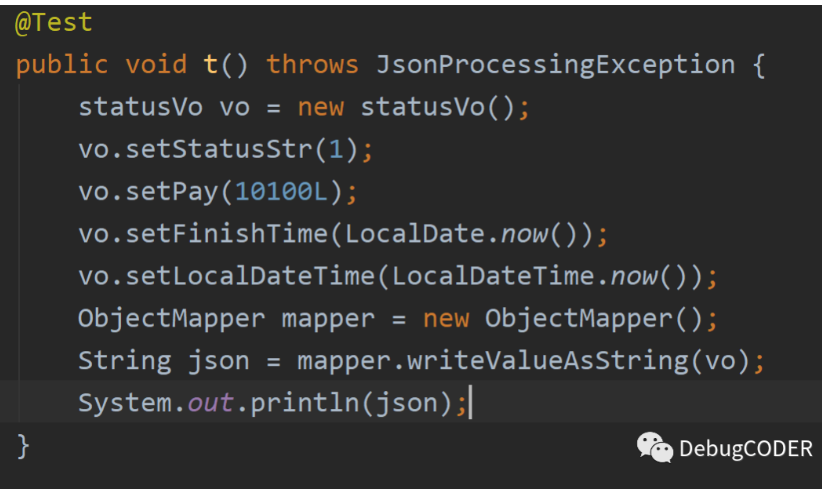
#### 3、配置文件中新增



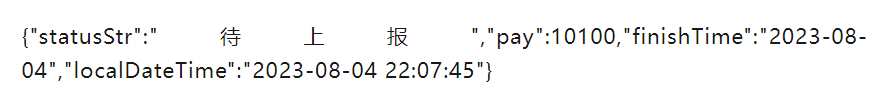
#### 4、实体类



#### 5、单元测试类



#### 6、测试结果



#### 总结

SpringBoot的Jackson默认支持的时间类型为Date，配置application.properties文件里的全局配置只对Date生效。@JsonFormat(pattern = "yyyy-MM-dd HH:mm:ss", timezone = "GMT+8") 只能解决Controller方法入参和返回对象的LocalDateTime，对ObjectMapper操作对象无效，需要配置LocalDateTime的序列化器和反序列化器。