**JSON Serialization**

**JSON**(JavaScript Object Notation) is a lightweight data-interchange format. It is readable for humans and is easy for machines to parse and generate. JSON is the favorite format for data being used in JavaScript applications, for REST based application and is the internal format used by quite a few NoSQL databases. So, it is only appropriate that we show you how to serialize Java objects using this format as well. The advantage of serializing Java objects to JSON is that there is more than one library providing classes to do so, which means at least one of them is stable with Java 9+ versions.

The most preferred library for JSON serialization is the Jackson library, because it can convert Java objects to JSON objects and back again without much code being needed to be written. Unfortunately, no version compatible with Java 9+ had been released yet, so for this section, a less advanced library compatible with Java 9+ will be used.

JSON format is a collection of key-pair values. The values can be arrays, or collections of key-pairs themselves. Converting Java objects to JSON objects using the JSON library (yes, it’s named exactly like that) is easy. We create a JSONObject and populate it with the field names and values of the Singer object, and then we convert the JSONObject to String and eventually write it to a file. In the following code sample, we skipped the writing to file part and we transform the Singer back into a copy of the initial Singer object.

package com.apress.bgn.ch11.json;

import com.apress.bgn.ch11.xml.Singer;

import org.json.JSONObject;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import java.time.LocalDate;

import java.time.Month;

import java.time.format.DateTimeFormatter;

Tuần tự hóa XML đã thống trị lĩnh vực phát triển trong nhiều năm, được sử dụng trong hầu hết các dịch vụ web và giao tiếp từ xa. Nhưng các tệp XML có xu hướng đông đúc, dư thừa và khó đọc khi chúng trở nên lớn hơn. Vì vậy, một định dạng mới đã kế thừa chương trình: JSON.

**Tuần tự hóa JSON**

**JSON** (JavaScript Object Notation) là một định dạng trao đổi dữ liệu nhẹ. Nó có thể đọc được đối với con người và dễ dàng để máy móc phân tích cú pháp cũng như tạo ra. JSON là định dạng rất thích hợp cho dữ liệu được sử dụng trong các ứng dụng JavaScript, cho ứng dụng dựa trên REST và là định dạng nội bộ được khá nhiều cơ sở dữ liệu NoSQL sử dụng. Vì vậy, chúng tôi chỉ cho bạn cách tuần tự hóa các đối tượng Java bằng cách sử dụng định dạng này. Ưu điểm của việc tuần tự hóa các đối tượng Java sang JSON là có nhiều hơn một thư viện cung cấp các lớp để làm điều đó, có nghĩa là ít nhất một trong số chúng ổn định với các phiên bản Java 9+.

Thư viện được ưa thích nhất để tuần tự hóa JSON là thư viện Jackson, vì nó có thể chuyển đổi các đối tượng Java thành các đối tượng JSON và quay lại một lần nữa mà không cần viết nhiều code. Rất tiếc, chưa có phiên bản nào tương thích với Java 9+ được phát hành, vì vậy đối với phần này, một thư viện ít nâng cao hơn tương thích với Java 9+ sẽ được sử dụng.

Định dạng JSON là một tập hợp các giá trị của cặp khóa. Các giá trị có thể là mảng hoặc tập hợp các cặp khóa. Chuyển đổi các đối tượng Java thành các đối tượng JSON bằng cách sử dụng thư viện JSON (vâng, nó được đặt tên chính xác là như vậy) thật dễ dàng. Chúng tôi tạo một JSONObject và điền nó với các tên trường và giá trị của đối tượng Singer, sau đó chúng tôi chuyển đổi JSONObject thành String và cuối cùng ghi nó vào một tệp. Trong mẫu code sau, chúng tôi đã bỏ qua phần ghi vào tệp và chuyển đổi Singer trở lại thành bản sao của đối tượng Singer ban đầu.

package com.apress.bgn.ch11.json;

import com.apress.bgn.ch11.xml.Singer;

import org.json.JSONObject;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import java.time.LocalDate;

import java.time.Month;

import java.time.format.DateTimeFormatter;

public class JsonSerializationDemo {

private static final Logger log =

LoggerFactory.getLogger(JsonSerializationDemo.class);

public static void main(String... args) {

LocalDate johnBd = LocalDate.of(1977, Month.OCTOBER, 16);

Singer john = new Singer("John Mayer", 5.0, johnBd);

JSONObject jsonObject = new JSONObject();

jsonObject.put("name", john.getName());

jsonObject.put("rating", john.getRating());

jsonObject.put("birthdate", john.getBirthDate().toString());

String jsonData = jsonObject.toString(2);

log.info("--> Serialized {}", jsonData);

JSONObject readJson = new JSONObject(jsonData);

Singer copyOfJohn = new Singer((String) readJson.get("name"),

Double.parseDouble(((Integer)readJson.get("rating")).

toString()),

LocalDate.parse((String)readJson.get("birthdate"),

DateTimeFormatter.ISO\_LOCAL\_DATE));

log.info("Are objects equal? {}", copyOfJohn.equals(john));

log.info("--> Deserialized {}", copyOfJohn);

}

}

The number given as a parameter to the jsonObject.toString(2); method is an indentation value used to format the resulted text. When the previous program is executed, the output you can expect to see in the console should look very similar to this.

[main] INFO com.apress.bgn.ch11.json.JsonSerializationDemo - -->

Serialized {

"birthdate": "1977-10-16",

"name": "John Mayer",

"rating": 5

}

public class JsonSerializationDemo {

private static final Logger log =

LoggerFactory.getLogger(JsonSerializationDemo.class);

public static void main(String... args) {

LocalDate johnBd = LocalDate.of(1977, Month.OCTOBER, 16);

Singer john = new Singer("John Mayer", 5.0, johnBd);

JSONObject jsonObject = new JSONObject();

jsonObject.put("name", john.getName());

jsonObject.put("rating", john.getRating());

jsonObject.put("birthdate", john.getBirthDate().toString());

String jsonData = jsonObject.toString(2);

log.info("--> Serialized {}", jsonData);

JSONObject readJson = new JSONObject(jsonData);

Singer copyOfJohn = new Singer((String) readJson.get("name"),

Double.parseDouble(((Integer)readJson.get("rating")).

toString()),

LocalDate.parse((String)readJson.get("birthdate"),

DateTimeFormatter.ISO\_LOCAL\_DATE));

log.info("Are objects equal? {}", copyOfJohn.equals(john));

log.info("--> Deserialized {}", copyOfJohn);

}

}

Số được cung cấp dưới dạng tham số cho jsonObject.toString(2); phương thức là một giá trị thụt lề được sử dụng để định dạng văn bản kết quả. Khi chương trình trước đó được thực thi, đầu ra bạn có thể thấy trong bảng điều khiển sẽ trông rất giống với kết quả này.

[main] INFO com.apress.bgn.ch11.json.JsonSerializationDemo - -->

Serialized {

"birthdate": "1977-10-16",

"name": "John Mayer",

"rating": 5

}

[main] INFO com.apress.bgn.ch11.json.JsonSerializationDemo - Are objects equal? true

[main] INFO com.apress.bgn.ch11.json.JsonSerializationDemo - -->

Deserialized Singer{name='John Mayer', rating=5.0, birthDate=1977-10-16}

Theoretically this library provides a method to serialize an object directly by calling:

LocalDate johnBd = LocalDate.of(1977, Month.OCTOBER, 16);

Singer john = new Singer("John Mayer", 5.0, johnBd);

JSONObject jo = new JSONObject(john);

But the version that the project is currently using seems to have a bug and cannot actually do that. So, the only hope for developers that plan to write Java 9+ applications to use practical JSON serialization/deserialization is to either build one themselves, or to hope that a stable version of Jackson built with Java9+ is available soon.

**The Media API**

Aside from text data, Java can manipulate binary files such as images. The Java Media

API contains a set of image encoder/decoder (codec) classes for several popular image

storage formats: BMP, GIF (decoder only), FlashPix (decoder only), JPEG, PNG, PNM, TIFF, and WBMP.

In Java 9, the Java media API was transformed as well and functionality to encapsulate many images with different resolutions into a multiresolution image was added.

The core of the Java Media API is the java.awt.Image class that is the superclass to represent graphical images. The most important image classes and their relationships are depicted in Figure 11-7.

[main] INFO com.apress.bgn.ch11.json.JsonSerializationDemo - Are objects equal? true

[main] INFO com.apress.bgn.ch11.json.JsonSerializationDemo - -->

Deserialized Singer{name='John Mayer', rating=5.0, birthDate=1977-10-16}

Về mặt lý thuyết, thư viện này cung cấp một phương thức để tuần tự hóa một đối tượng trực tiếp bằng cách gọi:

LocalDate johnBd = LocalDate.of(1977, Month.OCTOBER, 16);

Singer john = new Singer("John Mayer", 5.0, johnBd);

JSONObject jo = new JSONObject(john);

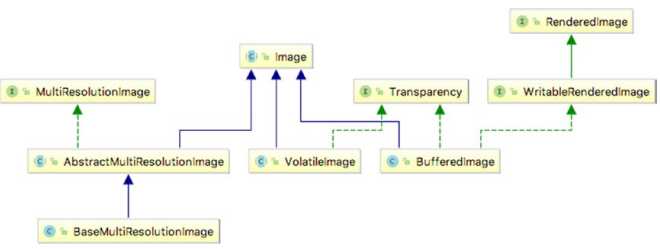
Nhưng phiên bản mà dự án hiện đang sử dụng dường như có một lỗi và thực sự không thể làm được điều đó. Vì vậy, hy vọng duy nhất cho các nhà phát triển có kế hoạch viết các ứng dụng Java 9+ để sử dụng JSON serialization/deserialization thực tế là tự xây dựng một ứng dụng hoặc hy vọng rằng phiên bản ổn định của Jackson được xây dựng bằng Java9 + sẽ sớm ra mắt.

**Media API**

Ngoài dữ liệu văn bản, Java có thể thao tác các tệp nhị phân như hình ảnh. Phương tiện Java API chứa một tập hợp các lớp mã hóa hình ảnh / giải mã hình ảnh (codec) cho một số hình ảnh phổ biến định dạng lưu trữ: BMP, GIF (chỉ bộ giải mã), FlashPix (chỉ bộ giải mã), JPEG, PNG, PNM, TIFF và WBMP.

Trong Java 9, Java media API cũng được chuyển đổi và chức năng đóng gói nhiều hình ảnh có độ phân giải khác nhau thành một hình ảnh đa phân giải đã được thêm vào.

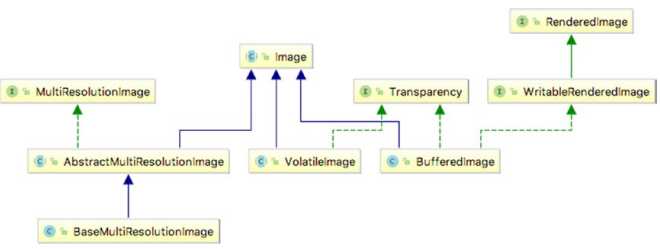
Cốt lõi của Java Media API là lớp java.awt.Image class đó là lớp siêu cấp để biểu diễn hình ảnh đồ họa. Các lớp hình ảnh quan trọng nhất và mối quan hệ của chúng được mô tả trong Hình 11-7.



***Figure 11-7****. Image classes hierarchy*

Although the java.awt.Image class is the most important in this hierarchy, the most used is java.awt.BufferedImage, which is an implementation with an accessible buffer of image data. It provides a lot of methods to create an image, to set its size and its contents, to extract its contents and analyze them, and so much more. In this section, we make use of this class to read and write images.

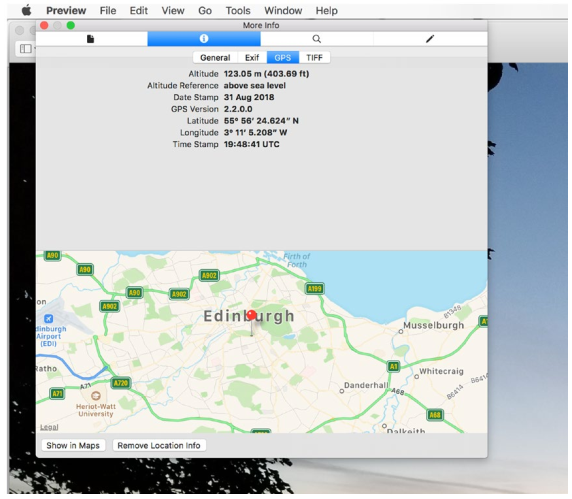
An image file is a complex file, aside from the picture itself, contains a lot of additional information, the most important nowadays is the location where that image was created. If you ever wondered how a social network proposes a check-in location for an image you are posting, this is where the information is found. This might not seem that important, but posting a picture of your cat, taken in your house, exposes your location to the whole world getting their hands on it. I’m not sure what you think about it, but to me this is terrifying. I used to post pictures of my cat sitting comfortable on the computer I am writing this book on now on my personal blog. I basically exposed my location and that of an expensive laptop to the whole world. Sure, most people do not care about my cat, nor the laptop, but somebody that might be looking to make an easy buck might. So, after a friendly and knowledgeable reader send me a private email telling be about something called EXIF data and how he knows where I live because of the last picture I’ve posted on my blog, I looked into it. A photo’s EXIF data contains a ton of information about your camera, and where the picture was taken (GPS coordinates). Most smartphones embed EXIF data into pictures taken with their camera. Figure 11-8 shows the EXIF information depicted in the macOS **Preview** application.



***Hình 11-7.*** *Hệ thống phân cấp các lớp hình ảnh*

Mặc dù lớp java.awt.Image là quan trọng nhất trong hệ thống phân cấp này, nhưng được sử dụng nhiều nhất là java.awt.BufferedImage, một triển khai với bộ đệm dữ liệu hình ảnh có thể truy cập được. Nó cung cấp rất nhiều phương pháp để tạo hình ảnh, đặt kích thước và nội dung của nó, trích xuất nội dung của nó và phân tích chúng, v.v. Trong phần này, chúng tôi sử dụng lớp này để đọc và ghi hình ảnh.

Tệp hình ảnh là một tệp phức tạp, ngoài bản thân bức ảnh, còn chứa rất nhiều thông tin bổ sung, quan trọng nhất hiện nay là vị trí nơi hình ảnh đó được tạo ra. Nếu bạn từng thắc mắc về cách mạng xã hội đề xuất địa điểm đăng ký cho hình ảnh bạn đang đăng tải, thì đây chính là nơi thông tin được tìm thấy. Điều này có vẻ không quan trọng lắm, nhưng việc đăng một bức ảnh về con mèo của bạn, được chụp trong nhà của bạn, sẽ hiển thị vị trí của bạn cho cả thế giới bắt tay vào. Tôi không chắc bạn nghĩ gì về điều đó, nhưng với tôi điều này thật kinh hoàng. Tôi đã từng đăng hình ảnh con mèo của tôi đang ngồi thoải mái trên máy tính. Hiện tôi đang viết cuốn sách này trên blog cá nhân của mình. Về cơ bản, tôi đã tiết lộ vị trí của mình và của một chiếc máy tính xách tay đắt tiền cho toàn thế giới. Chắc chắn, hầu hết mọi người không quan tâm đến con mèo của tôi, cũng không phải máy tính xách tay, nhưng ai đó có thể đang tìm cách kiếm tiền dễ dàng. Vì vậy, sau khi một độc giả thân thiện và hiểu biết gửi cho tôi email riêng tư, cho tôi biết về một thứ gọi là dữ liệu EXIF ​​và cách anh ấy biết tôi sống ở đâu nhờ bức ảnh cuối cùng tôi đăng trên blog của mình, tôi đã xem xét nó. Dữ liệu EXIF ​​của ảnh chứa rất nhiều thông tin về máy ảnh của bạn và nơi chụp ảnh (tọa độ GPS). Hầu hết các điện thoại thông minh nhúng dữ liệu EXIF ​​vào ảnh được chụp bằng máy ảnh của họ. Hình 11-8 cho thấy thông tin EXIF ​​được mô tả trong ứng dụng MacOS **Preview (Xem trước)**.



***Figure 11-8.*** *EXIF information on a JPG image*

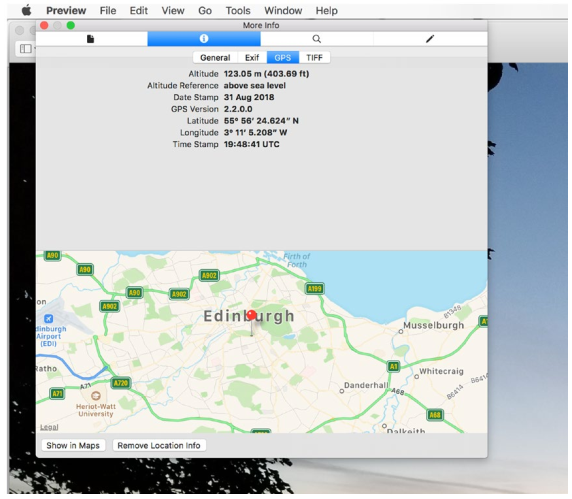
The EXIF info contains the exact location (latitude and longitude included) where the picture was taken. **EXIF** stands for Exchangeable Image File Format. There are utilities to remove it, but when you post a lot of pictures on your blog (like I do), it takes too much time to clean them one by one. This is where Java comes in and I will share with you a snippet of code that I use to clean my pictures of EXIF data.

package com.apress.bgn.ch11;

import org.apache.commons.imaging.formats.jpeg.exif.ExifRewriter;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;



***Hình 11-8.*** *Thông tin EXIF trên hình ảnh JPG*

Thông tin EXIF chứa vị trí chính xác (bao gồm cả vĩ độ và kinh độ) nơi bức ảnh được chụp. **EXIF** là viết tắt của Exchangeable Image File Format (Định dạng tệp hình ảnh có thể trao đổi). Có những tiện ích để loại bỏ nó, nhưng khi bạn đăng nhiều ảnh lên blog của mình (như tôi), bạn sẽ mất quá nhiều thời gian để làm sạch từng bức một. Đây là lúc Java xuất hiện và tôi sẽ chia sẻ với bạn một đoạn code mà tôi sử dụng để làm sạch ảnh dữ liệu EXIF của mình.

package com.apress.bgn.ch11;

import org.apache.commons.imaging.formats.jpeg.exif.ExifRewriter;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;