How to rotate JPEG images based on the orientation metadata?

Asked 13 years, 4 months ago   Modified 3 years, 3 months ago   Viewed 68k times

74

I have some server code that is generating thumbnails when an image is uploaded. The issue is that when the image was taken and the camera/device was rotated, the thumbnails are rotated, even though the full size images themselves are displayed in the correct orientation in any image viewing software. This is only happening with jpgs.

Using Preview on OSX, I can see that jpgs have orientation metadata embedded within. When I use ImageTools (Grails Plugin) to generate a thumbnail, the EXIF metadata is not in the thumbnail, which is why the thumbnails appear rotated.

Via offline conversations, I have learned that while it is relatively easy to read EXIF metadata, there is no easy way to write it, which is why the data is lost when generating a jpg thumbnail.

So it seems I have two options:

1. Use ImageMagick to generate the thumbnails. The downside is it requires installed more software on our servers.
2. Read the EXIF Orientation data is code and rotate the thumbnail appropriately.

Does anyone know of any other options?

java **image-processing** **groovy**

Share Improve this question Follow

edited Jul 14, 2015 at 4:11
RealSkeptic
34.4k 7 54 81

asked May 6, 2011 at 1:18
hvgotcodes
120k 33 207 237

2 If you just want a batch command-line option, imagickmagick can do this. Look into the `-auto-orient` command line flag. If you're transforming jpegs and want to avoid problems with re-compression, you can use `jhead` to do this, as well. `jhead -autorot *.jpg` should do what you need. I'm afraid I don't have a java solution, though... – Joe Kington May 6, 2011 at 17:17

@joe, in the end all i want is for the thumbnails to 'look right'. if possible, id like to solve this issue by somehow making the browser realize that the thing is oriented. – hvgotcodes May 7, 2011 at 18:55

9 Answers

Sorted by: Highest score (default)

77

+50

If you want to rotate your images, I would suggest to use the metadata extractor library http://code.google.com/p/metadata-extractor/. You can get the image information with the following code:

```
// Inner class containing image information
public static class ImageInformation {
    public final int orientation;
    public final int width;
    public final int height;

    public ImageInformation(int orientation, int width, int height) {
        this.orientation = orientation;
        this.width = width;
        this.height = height;
    }

    public String toString() {
        return String.format("%dx%d,%d", this.width, this.height,
this.orientation);
    }
}


public static ImageInformation readImageInformation(File imageFile)  throws
IOException, MetadataException, ImageProcessingException {
    Metadata metadata = ImageMetadataReader.readMetadata(imageFile);
    Directory directory =
metadata.getFirstDirectoryOfType(ExifIFD0Directory.class);
    JpegDirectory jpegDirectory =
metadata.getFirstDirectoryOfType(JpegDirectory.class);

    int orientation = 1;
    try {
        orientation = directory.getInt(ExifIFD0Directory.TAG_ORIENTATION);
    } catch (MetadataException me) {
        logger.warn("Could not get orientation");
    }
    int width = jpegDirectory.getImageWidth();
    int height = jpegDirectory.getImageHeight();

    return new ImageInformation(orientation, width, height);
}
```

Then given the orientation you retrieve, you can rotate and/or flip the image to the right orientation. The Affine transform for the EXIF orientation is given by the following method:

```
// Look at http://chunter.tistory.com/143 for information
public static AffineTransform getExifTransformation(ImageInformation info) {

    AffineTransform t = new AffineTransform();

    switch (info.orientation) {
    case 1:
        break;
    case 2: // Flip X
        t.scale(-1.0, 1.0);
        t.translate(-info.width, 0);
        break;
    case 3: // PI rotation
        t.translate(info.width, info.height);
        t.rotate(Math.PI);
        break;
    case 4: // Flip Y
        t.scale(1.0, -1.0);
        t.translate(0, -info.height);
        break;
    case 5: // - PI/2 and Flip X
        t.rotate(-Math.PI / 2);
        t.scale(-1.0, 1.0);
        break;
    case 6: // -PI/2 and -width
        t.translate(info.height, 0);
        t.rotate(Math.PI / 2);
        break;
    case 7: // PI/2 and Flip
        t.scale(-1.0, 1.0);
        t.translate(-info.height, 0);
        t.translate(0, info.width);
        t.rotate(  3 * Math.PI / 2);
        break;
    case 8: // PI / 2
        t.translate(0, info.width);
        t.rotate(  3 * Math.PI / 2);
        break;
    }
    }

    return t;
}
```

The rotation of the image would be done by the following method:

```
public static BufferedImage transformImage(BufferedImage image, AffineTransform
transform) throws Exception {

    AffineTransformOp op = new AffineTransformOp(transform,
AffineTransformOp.TYPE_BICUBIC);

    BufferedImage destinationImage = op.createCompatibleDestImage(image,
(image.getType() == BufferedImage.TYPE_BYTE_GRAY) ? image.getColorModel() : null
);
    Graphics2D g = destinationImage.createGraphics();
    g.setBackground(Color.WHITE);
    g.clearRect(0, 0, destinationImage.getWidth(),
destinationImage.getHeight());
    destinationImage = op.filter(image, destinationImage);
    return destinationImage;
}
```

In a server environment, don't forget to run with `-Djava.awt.headless=true`

Share Improve this answer Follow

edited May 23, 2016 at 16:03
Matěj Kripner
74 2 8

answered May 15, 2011 at 18:50
Antoine Martin
1,257 11 6

1 exactly what i did, except i used the thumbnailarator library, which has a rotate method in it. Since you took the time to show code for a bounty, you get it.
– hvgotcodes May 15, 2011 at 18:56

Note that in `readImageInformation` , `directory` (and possibly also `jpegDirectory`) can be `null` . – Samuel Aug 10, 2012 at 11:39

Thanks for the answer, it's almost working for me. Correct me if I'm mistaken though, but should the colormodel line in transformImage be: BufferedImage destinationImage = op.createCompatibleDestImage(image, (image.getType() == BufferedImage.TYPE_BYTE_GRAY)? null : image.getColorModel());
– jsaven Feb 2, 2013 at 5:12

27

The Thumbnailator library honors EXIF orientation flags. To read an image at full size with correct orientation:

```
BufferedImage image = Thumbnails.of(inputStream).scale(1).asBufferedImage();
```

Share Improve this answer Follow

edited Apr 6, 2016 at 19:52

answered Sep 30, 2014 at 21:07
dnault
8,809 1 36 56

Fantastic! I was not aware of this hidden capability. It works perfectly for auto-rotating images as they're read in. And it was way easier than working through metadata-extractor. – Jeff Oct 21, 2015 at 2:31

2 Unfortunately, Thumbnailator's image quality after rotation is poor. github.com/coobird/thumbnailator/issues/101 – DylanYi Feb 11, 2020 at 2:26

9

This can be done surprisingly easily by using the image part of JavaXT core library :

```
// Browsers today can't handle images with Exif Orientation tag
Image image = new Image(uploadedFilename);
// Auto-rotate based on Exif Orientation tag, and remove all Exif tags
image.rotate();
image.saveAs(permanentFilename);
```

That's it!

I have tried Apache Commons Imaging, but that was a mess. JavaXT is way more elegant.

Share Improve this answer Follow

answered Jan 24, 2014 at 13:51
Per Lindberg
747 9 8

1 Sadly javaxt doesnt have a maven repo as far as I can tell (maybe I missed it?), which means I would need to do a bunch of custom build stuff just to use em :(– Gus May 10, 2014 at 16:52

1 Unfortunately, the JavaXT core library does not rotate the image correctly in some cases. It works on some images, but not on others. One image that works has ExifVersion=Exif Version 2.1, one image that does not work has ExifVersion=Exif Version 2.2. Perhaps that's the problem, JavaXT core does not handle version 2.2. I don't know. – Per Lindberg Jun 10, 2014 at 12:38

2 Also, image.saveAs() uses memory-mapped file, so the result file can be empty or locked in Windows. Saving via a byte array seems to work better. But I'm throwing out JavaXT anyway. – Per Lindberg Jun 10, 2014 at 12:57

1 javaxt is available for maven: mvnrepository.com/artifact/javaxt/javaxt-core – yglodt Dec 17, 2018 at 13:46

@PerLindberg, see my answer below. The hybrid solution works for both Exif 2.1 and Exid 2.2 and leverages your recommendation. – Ted Gulesserian May 6, 2019 at 15:39

4

My solution is a combination of @PerLindberg's answer as well as @AntoineMartin's. I tried the other answers with Java 8 on Windows 10 and none seemed to do the trick. @AntoinMartin's com.drew.imaging solution was slow and image turned out black and white and full of artifacts. @PerLindberg's JavaXT solution did not read Exif 2.2 data.

1) Use com.drew.imaging to read exif information:

```
// Inner class containing image information
public static class ImageInformation {
    public final int orientation;
    public final int width;
    public final int height;

    public ImageInformation(int orientation, int width, int height) {
        this.orientation = orientation;
        this.width = width;
        this.height = height;
    }

    public String toString() {
        return String.format("%dx%d,%d", this.width, this.height,
this.orientation);
    }
}

public ImageInformation readImageInformation(File imageFile)  throws
IOException, MetadataException, ImageProcessingException {
    Metadata metadata = ImageMetadataReader.readMetadata(imageFile);
    Directory directory =
metadata.getFirstDirectoryOfType(ExifIFD0Directory.class);
    JpegDirectory jpegDirectory =
metadata.getFirstDirectoryOfType(JpegDirectory.class);

    int orientation = 1;
    if (directory != null) {
        try {
            orientation = directory.getInt(ExifIFD0Directory.TAG_ORIENTATION);
        } catch (MetadataException me) {
            logger.warn("Could not get orientation");
        }
        int width = jpegDirectory.getImageWidth();
        int height = jpegDirectory.getImageHeight();
        return new ImageInformation(orientation, width, height);
    } else {
        return null;
    }
}
```

2) Use JavaXT to perform the rotation based on Exif data.

```
public void rotateMyImage(String imageDownloadFilenme);
    File imageDownloadFile =  new File(imgageDownloadFilenme);
    Image image = new Image(imgageDownloadFilenme);
    ImageInformation imageInformation = readImageInformation(imageDownloadFile);
    if (imageInformation != null) {
        rotate(imageInformation, image);
    }
    image.saveAs(imgageDownloadFilenme);
}

public void rotate(ImageInformation info, Image image) {

    switch(info.orientation) {
        case 1:
            return;
        case 2:
            image.flip();
            break;
        case 3:
            image.rotate(180.0D);
            break;
        case 4:
            image.flip();
            image.rotate(180.0D);
            break;
        case 5:
            image.flip();
            image.rotate(270.0D);
            break;
        case 6:
            image.rotate(90.0D);
            break;
        case 7:
            image.flip();
            image.rotate(90.0D);
            break;
        case 8:
            image.rotate(270.0D);
    }

}
```

Share Improve this answer Follow

edited May 6, 2019 at 15:40

answered May 6, 2019 at 15:20

Ted Gulesserian
277 2 8

Alternatively, you could use JavaXT with TwelveMonkeys. TwelveMonkeys provides a JPEG ImageIO plugin that allows javaxt.io.Image to handle most, if not all Exif metadata. More info and workaround described here. – Peter Nov 3, 2019 at 15:56

3

Exif seems to be hard to write because of proprietary stuff in it. However, you can consider another option

Read original but only write orientation tag to thumbnails.

Apache Sanselan seems to have nice collection of tools to do it.

http://commons.apache.org/proper/commons-imaging/

Look at ExifRewriter class, for example.

Share Improve this answer Follow

edited Mar 8, 2013 at 19:49

quietmint
14.1k 7 49 73

answered May 14, 2011 at 1:21

Alex Gitelman
24.6k 7 53 49

3

As *dnault* mentioned in previous comment, Thumbnaliator library resolves the issue. But you should use correct input/output formats to avoid color change on this automatic rotation.

```
ByteArrayOutputStream baos = new ByteArrayOutputStream();
ByteArrayInputStream in = new ByteArrayInputStream(file.getContents());
Thumbnails.of(in)
    .scale(1)
    .toOutputStream(baos);
byte[] bytes = baos.toByteArray();
```

Share Improve this answer Follow

answered Sep 10, 2018 at 8:26
Anton Petrovskyi
452 5 19

1

If you just want it to look right. You can just add a "rotate" -PI/2 (-90 degrees), PI/2 (90 degrees), or PI (+180 degrees) as necessary depending on the 'orientation' you've already extracted. The browser or any other program will correctly display the image as the orientation will have been applied and the metadata stripped from the thumbnail output.

Share Improve this answer Follow

answered May 12, 2011 at 2:49
karmakaze
35.8k 1 33 33

karmakaze -- yes, i think I am going to have to do that -- you are talking about on the server? I am worried that different cameras might have different metadata -- is that valid? Also, are there any image formats other than jpg that will require this? – hvgotcodes May 12, 2011 at 12:29

According to Wikipedia, Exif is for JPEG and TIFF image files, as well as some audio file formats. It is not supported in JPEG 2000, PNG, or GIF. Many native formats used by digital cameras will have Exif tags. – karmakaze May 13, 2011 at 0:25

0

The rotation functions above rotate the image data. And there is one more thing to do. You must set the width and height of BufferedImage. When rotating 90 degrees and 270 degrees, the width and height must be interchanged.

```
    BufferedImage transformed;
    switch (orientation) {
        case 5:
        case 6:
        case 7:
        case 8:
            transformed = new BufferedImage(image.getHeight(), image.getWidth(),
image.getType());
            break;
        default:
            transformed = new BufferedImage(image.getWidth(), image.getHeight(),
image.getType());
    }
```

Share Improve this answer Follow

answered May 26, 2021 at 0:12

HeeGu

1 1

-1

Based on the answers of Antoine Martin I created an own class for correcting the orientation of a given jpeg image (in my case as an input stream) based on the exif information of the image. With his solution I had the problem, that the colors of the resulting image were wrong, therefore I created this one. For retrieving the metadata of the image I used the metadata-extractor library.

I hope it will help some people.

```
public class ImageOrientationUtil {

/**
 * Checks the orientation of the image and corrects it if necessary.
 * <p>If the orientation of the image does not need to be corrected, no
operation will be performed.</p>
 * @param inputStream
 * @return
 * @throws ImageProcessingException
 * @throws IOException
 * @throws MetadataException
 */
public static BufferedImage correctOrientation(InputStream inputStream) throws
ImageProcessingException, IOException, MetadataException {
    Metadata metadata = ImageMetadataReader.readMetadata(inputStream);
    if(metadata != null) {
        if(metadata.containsDirectoryOfType(ExifIFD0Directory.class)) {
            // Get the current orientation of the image
            Directory directory =
metadata.getFirstDirectoryOfType(ExifIFD0Directory.class);
            int orientation =
directory.getInt(ExifIFD0Directory.TAG_ORIENTATION);

            // Create a buffered image from the input stream
            BufferedImage bimg = ImageIO.read(inputStream);


            // Get the current width and height of the image
            int[] imageSize = {bimg.getWidth(), bimg.getHeight()};
            int width = imageSize[0];
            int height = imageSize[1];

            // Determine which correction is needed
            AffineTransform t = new AffineTransform();
            switch(orientation) {
```

```
            case 1:
                // no correction necessary skip and return the image
                return bimg;
            case 2: // Flip X
                t.scale(-1.0, 1.0);
                t.translate(-width, 0);
                return transform(bimg, t);
            case 3: // PI rotation
                t.translate(width, height);
                t.rotate(Math.PI);
                return transform(bimg, t);
            case 4: // Flip Y
                t.scale(1.0, -1.0);
                t.translate(0, -height);
                return transform(bimg, t);
            case 5: // - PI/2 and Flip X
                t.rotate(-Math.PI / 2);
                t.scale(-1.0, 1.0);
                return transform(bimg, t);
            case 6: // -PI/2 and -width
                t.translate(height, 0);
                t.rotate(Math.PI / 2);
                return transform(bimg, t);
            case 7: // PI/2 and Flip
                t.scale(-1.0, 1.0);
                t.translate(height, 0);
                t.translate(0, width);
                t.rotate(  3 * Math.PI / 2);
                return transform(bimg, t);
            case 8: // PI / 2
                t.translate(0, width);
                t.rotate(  3 * Math.PI / 2);
                return transform(bimg, t);
            }
        }
    }

    return null;
}

/**
 * Performs the tranformation
 * @param bimage
 * @param transform
 * @return
 * @return
 * @throws IOException
 */
private static BufferedImage transform(BufferedImage bimage, AffineTransform
transform) throws IOException {
    // Create an transformation operation
    AffineTransformOp op = new AffineTransformOp(transform,
AffineTransformOp.TYPE_BICUBIC);

    // Create an instance of the resulting image, with the same width, height
and image type than the referenced one
    BufferedImage destinationImage = new BufferedImage( bimage.getWidth(),
bimage.getHeight(), bimage.getType() );
    op.filter(bimage, destinationImage);

   return destinationImage;
}
}
```

Share Improve this answer Follow

answered Jul 22, 2017 at 9:18

Florian
120 8

This will not work as you are reading the input stream twice, once to obtain the `Metadata` and then to create the `BufferedImage`. You would need to either copy the stream or use a variant that can be reset and read again. The destination image may also have the incorrect bounds for something like a 90 degree CW or CCW rotation unless it is square, you need to obtain the new bounds from the transform operation when creating the destination image. – Robert Hunt Jan 4, 2018 at 10:58

Will work only for square images since the dimensions of rotated-image are kept same as original-image. – nishantbhardwaj2002 Nov 16, 2018 at 4:26