Java Annotations – Annotations in Java

APRIL 2, 2018 BY PANKAJ — 33 COMMENTS

Annotations in java provide information about the code. Java annotations have no direct effect on the code they annotate. In java annotations tutorial, we will look into following;

1. see built-in Java annotation example
2. how to write custom annotation
3. annotations usage and how to parse annotations using reflection

Table of Contents [hide]

Java Annotations

Annotations are introduced in Java 1.5 and now it's heavily used in Java EE frameworks like Hibernate, Jersey, Spring.

Java Annotation is metadata about the program embedded in the program itself. It can be parsed by the annotation parsing tool or by compiler. We can also specify annotation availability to either compile time only or till runtime also.

Before java annotations, program metadata was available through java comments or by javadoc but annotation offers more than that. Annotations metadata can be available at runtime too and annotation parsers can use it to determine the process flow.

For example, in Jersey webservice we add PATH annotation with URI string to a method and at runtime jersey parses it to determine the method to invoke for given URI pattern.

Java Custom Annotation

Creating custom annotation in java is similar to writing an interface, except that it interface keyword is prefixed with @ symbol. We can declare methods in annotation.

Let's see java custom annotation example and then we will discuss it's features and important points.

```
package com.journaldev.annotations;
```

```
import java.lang.annotation.Documented;
import java.lang.annotation.ElementType;
import java.lang.annotation.Inherited;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Documented
@Target(ElementType.METHOD)
@Inherited
@Retention(RetentionPolicy.RUNTIME)
public @interface MethodInfo{
        String author() default "Pankaj";
        String date();
        int revision() default 1;
        String comments();
}
```

Some important points about java annotations are:

1. Annotation methods can't have parameters.
2. Annotation methods return types are limited to primitives, String, Enums, Annotation or array of these.
3. Java Annotation methods can have default values.
4. Annotations can have meta annotations attached to them. Meta annotations are used to provide information about the annotation.

Meta annotations in java

There are four types of meta annotations:

1. **@Documented** – indicates that elements using this annotation should be documented by javadoc and similar tools. This type should be used to annotate the declarations of types whose annotations affect the use of annotated elements by their clients. If a type declaration is annotated with Documented, its annotations become part of the public API of the annotated elements.
2. **@Target** – indicates the kinds of program element to which an annotation type is applicable. Some possible values are TYPE, METHOD, CONSTRUCTOR, FIELD etc. If Target meta-annotation is not present, then annotation can be used on any program element.
3. **@Inherited** – indicates that an annotation type is automatically inherited. If user queries the annotation type on a class declaration, and the class declaration has no annotation for this type, then the class's superclass will automatically be queried for the annotation type. This process will be repeated until an annotation for this type is found, or the top of the class hierarchy (Object) is reached.

4. **@Retention** – indicates how long annotations with the annotated type are to be retained. It takes RetentionPolicy argument whose Possible values are SOURCE, CLASS and RUNTIME

Built-in annotations in Java

Java Provides three built-in annotations.

1. `@Override` – When we want to override a method of Superclass, we should use this annotation to inform compiler that we are overriding a method. So when superclass method is removed or changed, compiler will show error message. Learn why we should always use java override annotation while overriding a method.
2. `@Deprecated` – when we want the compiler to know that a method is deprecated, we should use this annotation. Java recommends that in javadoc, we should provide information for why this method is deprecated and what is the alternative to use.
3. `@SuppressWarnings` – This is just to tell compiler to ignore specific warnings they produce, for example using raw types in java generics. It's retention policy is SOURCE and it gets discarded by compiler.

Java Annotations Example

Let's see a java example showing use of built-in annotations in java as well as use of custom annotation created by us in above example.

```
package com.journaldev.annotations;

import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.List;

public class AnnotationExample {

	public static void main(String[] args) {
	}

	@Override
	@MethodInfo(author = "Pankaj", comments = "Main method", date = "Nov 17
2012", revision = 1)
	public String toString() {
		return "Overriden toString method";
	}

	@Deprecated
	@MethodInfo(comments = "deprecated method", date = "Nov 17 2012")
	public static void oldMethod() {
```

```
                System.out.println("old method, don't use it.");
```

I believe above java annotation example is self explanatory and showing use of annotations in different cases.

Java Annotations Parsing

We will use Reflection to parse java annotations from a class. Please note that Annotation Retention Policy should be *RUNTIME* otherwise it's information will not be available at runtime and we wont be able to fetch any data from it.

```
package com.journaldev.annotations;

import java.lang.annotation.Annotation;
import java.lang.reflect.Method;

public class AnnotationParsing {

        public static void main(String[] args) {
                try {
                        for (Method method : AnnotationParsing.class.getClassLoader()

.loadClass(("com.journaldev.annotations.AnnotationExample")).getMethods()) {
                                // checks if MethodInfo annotation is present for the
method
                                if
(method.isAnnotationPresent(com.journaldev.annotations.MethodInfo.class)) {
                                        try {
                                                // iterates all the annotations
available in the method
                                                for (Annotation anno :
method.getDeclaredAnnotations()) {
```

Output of the above program is:

```
Annotation in Method 'public java.lang.String
com.journaldev.annotations.AnnotationExample.toString()' :
@com.journaldev.annotations.MethodInfo(author=Pankaj, revision=1, comments=Main
method, date=Nov 17 2012)
Method with revision no 1 = public java.lang.String
com.journaldev.annotations.AnnotationExample.toString()
Annotation in Method 'public static void
```

```
com.journaldev.annotations.AnnotationExample.oldMethod()' : @java.lang.Deprecated()
Annotation in Method 'public static void
com.journaldev.annotations.AnnotationExample.oldMethod()' :
@com.journaldev.annotations.MethodInfo(author=Pankaj, revision=1, comments=deprecated
method, date=Nov 17 2012)
Method with revision no 1 = public static void
com.journaldev.annotations.AnnotationExample.oldMethod()
Annotation in Method 'public static void
com.journaldev.annotations.AnnotationExample.genericsTest() throws
java.io.FileNotFoundException' : @com.journaldev.annotations.MethodInfo(author=Pankaj,
revision=10, comments=Main method, date=Nov 17 2012)
```

Reflection API is very powerful and used widely in Java, J2EE frameworks like Spring, Hibernate, JUnit, check out **Reflection in Java**.

That's all for the java annotations example tutorial, I hope you learned something from it.

Java Annotations Updates:

1. Servlet Specs 3.0 introduced use of annotations for Servlet Configuration and init parameters, read more at **Java Servlet Tutorial**.
2. We can use annotations in Struts 2 to configure it's action classes and result pages, check working example at **Struts 2 Hello World Annotation Example**.

Reference: Oracle website

« PREVIOUS
Java Enum – Enum in Java

NEXT »
Java String to int, Java int to String

About Pankaj
If you have come this far, it means that you liked what you are reading. Why not reach little more and connect with me directly on **Google Plus**, **Facebook** or **Twitter**. I would love to hear your thoughts and opinions on my articles directly.

Recently I started creating video tutorials too, so do check out my videos on **Youtube**.

FILED UNDER: JAVA

Comments

hossein says
OCTOBER 7, 2017 AT 3:59 AM
Hi
thanks for this tutorial.
Please go for writtring an annotation processor
thanks alot
Reply

priya says
SEPTEMBER 25, 2017 AT 2:31 AM
All the concepts are easily explained. Post some more concepts of java 8 i need to know some basics, Thanks for sharing.
Reply

Anurag Singh says
JULY 17, 2017 AT 7:23 AM
very – very nice article about java annotation.
the example is also very nice and simple

very useful post
thank you for this post
Reply

Victor says
OCTOBER 8, 2016 AT 12:44 AM
good sample with completed example, compared to other tutorials, which got various inconveniences to learner, like java files lumped together, or missing "import" some classes, etc.
good work ..
Reply

Royceroy says
AUGUST 25, 2016 AT 4:22 AM
HI Buddy thanks for sharing useful and helpful information about Annotation.The Target annotation specifies which elements of our code can have annotations of the defined type. A concrete example will help explain this concept. Add the Target annotation to the Task annotation we defined earlier in the last post, as shown here:
https://www.mindstick.com/Articles/12141/annotations-in-java-target-and-retention
Reply

Shiraz says
JUNE 28, 2016 AT 2:30 PM
Good explanation. Succinct and to the point without being verbose. My first working annotation! Thank you.
Reply

Mann says
JANUARY 7, 2016 AT 10:24 AM
Hey Hi,
I'm new to Java world and am trying to explore more thus am not sure if my question is worth your time –
i want to initialize a bean with some parameters using java annotations. I'm able to achieve the same using XML though.
Background –
I'm trying to fetch the student (bean) details from classs (bean) where i want to initialize the studentid, student name and student age at bean initialization time.

Any help here will be a great support

Reply

Key Naugh says

DECEMBER 28, 2015 AT 9:19 AM

Hi Pankaj,

Question:

I currently have the situation that I can't import the 3rd party package "com.journaldev.annotations" in my class so I have to load it using reflection. In this case, can you guide me how to use reflection to implement the toString() below with the @MethodInfo annotation? Thank you so much.

@Override

@MethodInfo(author = "Pankaj", comments = "Main method", date = "Nov 17 2012", revision = 1)

public String toString() {

return "Overriden toString method";

}

Reply

Palak says

SEPTEMBER 21, 2015 AT 3:09 AM

Hi, please tell me how to do the same with method parammeters

Reply

naresh says

APRIL 1, 2015 AT 10:18 PM

Please Provide the Spring Boot Tutorial to my Mail…

Thanking You

Reply

Nasirkhan S. Sirajbhai says

FEBRUARY 19, 2015 AT 10:27 AM

Thanks for this tutorial, I have one question though, How Annotation parser validates or evaluates custom annotation, I mean how custom annotation parser gets activated?

Thanks

Reply

Khan says

DECEMBER 1, 2014 AT 10:45 PM

Thanks for good explination

Reply

dhaval says

NOVEMBER 19, 2014 AT 11:20 PM

hi pankaj how i can search custom annotation for method in java source code?i am parsing all java classes .i want method name that have some custom annotation .can you please help me??if possible put some example to search custom annotation in java code

Reply

Purpose of @Deprecated annotation in Java says

AUGUST 14, 2014 AT 10:30 AM

Hi,

Thanks for the tutorial.

Reply

Vijay says

AUGUST 11, 2014 AT 11:55 PM

Hi Pankaj,

your tutorial is really nice.I have one doubt,In the class AnnotationExample.java, we have 3 methods,1. toString() 2.oldMethod() 3.genericsTest(), and all these methods have 2 annotations, but during parsing in AnnotationParsing.java , we are getting 2 annotations for oldMethod() and 1 annotation for toString() as well as genericsTest(). do we have any reason for missing @override and @suppresswarnings in the output?

Reply

Pankaj says

AUGUST 12, 2014 AT 3:22 AM

That is because of Renention Policy, it should be Runtime to be accessible while running the program.

Reply

Vijay says

AUGUST 12, 2014 AT 7:30 AM

Got it ! Thanks Pankaj .You're doing a great job.Keep it up !!

Reply

Dheeraj Kumar says

AUGUST 1, 2014 AT 9:38 PM

Thank you.This is very good.Thank You Again……

Reply

Binh Thanh Nguyen says

JUNE 26, 2014 AT 6:10 PM

Thanks, nice post

Reply

Velu says

NOVEMBER 21, 2013 AT 8:37 AM

Useful & Detailed tutorial Thanks!!!

Reply

Siddu says

AUGUST 27, 2013 AT 5:07 AM

Could you tell me the difference between these below code

List list=new ArrayList();

ArrayList list=new ArrayList();

When to use this. Cold you provide real time examples on this.

Reply

Sohi says

AUGUST 27, 2013 AT 8:27 AM

Hi ,

List is the Interface that ArrayList implements. We use interfaces as Reference variable for concrete classes when we define common methods or contract for all of its implementation.

For example:
Suppose we have two implementations of List Interface:

```
LinkedList linkList = new LinkedList();
ArrayList arrList = new ArrayList();
```

then we declare method to do some work for each kind of list which do same work i.e. iterating over elements in list:
Method 1.

```
public void doSomeCalculation(ArrayList arrList){
for(String s : arrList)
{
System.out.println(“String is :”+ s);
}
}
```

Method 2.

```
public void doSomeCalculation1(LinkedList linkLIst){
for(String s : linkLIst)
{
System.out.println(“String is :”+ s);
}
}
```

Here we call methods:

```
public void CallToMEthod(){
doSomeCalculation(arrList);
doSomeCalculationLinked(linkList);
}
```

Important thing is we defined two separate methods to do same work for ArrayList and LinkedList.
If you try to access
doSomeCalculation on linkedlist or doSomeCalculationLinked on arrList
It will give error of compatibility.
So here will use concept of polymorphism(great feature of OO):
We know that List is Interface of Array List and Linked lIst:
we will declare reference variable of Interface type:

```
LIst arrList = new ArrayList();
List linkList = new LinkedList();
```

then we will declare only one method to iterate both types of list:

```
public void doSomeCalculation(List list) {
for(String s : list)
{
System.out.println(“String is :”+ s);
}
}
public void CallToMEthod(){
doSomeCalculation(arrList);
doSomeCalculation(linkList );
```

}
Here we use same method for both types of lists.
public void doSomeCalculation(List list) {
for(String s : list)
{
System.out.println("String is :"+ s);
}
}
This method will check at run time what kind of list is being passed and call that list type's Iterator.
Hope this will help you:-)
Reply

Pankaj says
AUGUST 28, 2013 AT 12:44 AM
Nice explanation Sohi, I liked it.
Reply

Sohi says
JANUARY 21, 2014 AT 10:04 AM
Thanks Pankaj. Your tutorials are very easy and simple to understand. Can you pls add some for Hadoop?:-)
Reply

Saravana says
MARCH 13, 2014 AT 1:35 AM
Yes, Please post some good articles for Hadoop, I'm gonna work on that in few months. □

ashutosh says
AUGUST 23, 2015 AT 8:36 PM
thanks Pankaj,
can you please guide me how to use annotation with jdk1.8 beacuse when we use annotation in eclipse with jdk 1.8 then we are getting an error as change project compliance and jre to 1.8 only with simple example.
i will be thankful for you
Reply

Siddu says
AUGUST 27, 2013 AT 5:04 AM
How to write business logic code in annotation in java?
Reply

Pankaj says
AUGUST 28, 2013 AT 12:43 AM
Annotations are metadata about the program, you can't put business logic in annotations.
Reply

Siddu says
AUGUST 28, 2013 AT 4:40 AM
Thanks, Ex @Override annotion, How it checks method being overriden properly by sub class without having any logic in @override. Could you please explain if time permits.
Reply

Velu says
NOVEMBER 21, 2013 AT 8:46 AM
That is kind of runtime scanning. http://stackoverflow.com/questions/10205261/how-to-create-custom-annotation-with-code-behind
Reply

Anonymous says
JULY 23, 2013 AT 1:43 PM
… and I meant on the subject "CUSTOM annotation creation" not standard/built-in usage (that we know and there is no need for documentation that is on Oracle website). I have learned not much here. I know how to use annotations, reflection and write main method with some System printouts. It would be much more useful to explain and show how custom annotation gets processed and write for example custom annotation that create a proxy method or sets a value on annotated variable. So called "one-liner-focused-example".
Reply

Pankaj says

JULY 24, 2013 AT 3:36 AM

As quoted by you "I have learned not much here", I am happy that at least you learned something. I am not sure what are you trying to say here, I have provided the sample code to create custom annotation and shown how can we use it in our program. Real life examples are usually more complex and idea here is to understand annotations with simple and easy to understand code.

Reply

Anonymous says

JULY 23, 2013 AT 1:36 PM

I see way too much fun with annotations (including Oracle flaky documentation that cannot stay focused on bulletpoints on the subject). Can we have in simple points steps to create annotation and process it required by Java with simplest example and no funky games with processors, testers and who-knows-what? I mean it is good to know it, but that's next after we become proficient… especially when in commercial programming we do not play in academic way – not time for this fun, we program for living according to certain steps that are supposed to help simplicity and not scientific complexity (maintiainable code principle).

Reply

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Save my name, email, and website in this browser for the next time I comment.

POST COMMENT

Search for tutorials...

DOWNLOAD ANDROID APP

GET IT ON Google Play

CORE JAVA TUTORIAL

Java 10 Tutorials Java 9 Tutorials Java 8 Tutorials Java 7 Tutorials Core Java Basics OOPS Concepts Data Types and Operators String Manipulation Java Arrays Annotation and Enum Java Collections Java IO Operations Java Exception Handling MultiThreading and Concurrency Regular Expressions Advanced Java Concepts

RECOMMENDED TUTORIALS

Java Tutorials

- Java IO
- Java Regular Expressions
- Multithreading in Java
- Java Logging
- Java Annotations
- Java XML
- Collections in Java
- Java Generics
- Exception Handling in Java

- Java Reflection
- Java Design Patterns
- JDBC Tutorial

Java EE Tutorials

- Servlet JSP Tutorial
- Struts2 Tutorial
- Spring Tutorial
- Hibernate Tutorial
- Primefaces Tutorial
- Apache Axis 2
- JAX-RS
- Memcached Tutorial

Java Course

Java Beginner Tutorials

Free Java Update

Learn Java Programming

Online Java Training

Java Updates & Games

Java Books For Beginners

Java Source Code