**Аннотация методов, annotation**

Аннотация "annotation" в языке Java – это специальная форма синтетических метаданных, которая может быть добавлена в исходный код. **Аннотации** используются для анализа кода, компиляции или выполнения. Аннотированы могут быть пакеты, классы, методы, переменные и параметры.

**Аннотация** выполняет следующие функции :

* предоставляет необходимую информацию для компилятора;
* предоставляет информацию различным инструментам для генерации другого кода, конфигураций и т. д.;
* может быть использована во время работы кода.

А по большому счету применяются либо как комментарии, либо для последующих действий \ отсутствия действий с классом\методом в зависимости от того есть ли у него соответствующая аннотация или нет (типа сортировки).

1. **Встроенные аннотации : @Override, @Deprecated, @SuppressWarnings**

Встроенные аннотации отслеживаются средой разработки IDE и применяются к java-коду метода :

|  |  |
| --- | --- |
| @Override | Проверка переопределения метода. IDE вызывает предупреждение компиляции, если метод не найден в родительском классе. |
| @Deprecated | IDE отмечает, что метод устарел и вызывает предупреждение компиляции, если метод используется. |
| @SuppressWarnings | Аннотация указывает IDE подавить предупреждения компиляции. |

1. **Аннотации, применяемые к другим аннотациям : @Retention, @Documented, @Target, @Inherited**

|  |  |
| --- | --- |
| @Retention | Определяет, как отмеченная аннотация будет храниться — в коде, в скомпилированном классе или во время работы кода. |
| @Documented | Отмечает аннотацию для включения в документацию. |
| @Target | Отмечает аннотацию как ограничивающую, какие элементы java-аннотации могут быть к ней применены. |
| @Inherited | Отмечает, что аннотация может быть расширенна подклассами аннотируемого класса. |

Первоначально в платформе Java имелся механизм, предваряющий механизм аннотаций — например, модификатор *transient* или тэг *@deprecated*. В сентябре 2002 года сообществу Java представлен документ JSR-175, описывающий основные тезисы по аннотациям. Он был утвержден в 2004 году. Аннотации стали доступны в самом языке начиная с версии Java 5.0 и описаны в JSR-269. В версии Java 6 аннотации были интегрированы в компилятор javac.

**Пример аннотации :**

public class Animal

{

public void speak() {

}

}

public class Cat extends Animal

{

@Override // Аннотация говорит о том, что этот метод переопределен

public void speak() {

System.out.println("Meow.");

}}

1. **Написание собственных аннотаций**

**Синтаксис аннотации, @interface**

Аннотации представляют из себя дескрипторы, включаемые в текст программы, и используются для хранения метаданных программного кода, необходимых на разных этапах жизненного цикла программы. Информация, хранимая в аннотациях, может использоваться соответствующими обработчиками для создания необходимых вспомогательных файлов или для маркировки классов, полей и т.д. То есть, аннотации могут быть применены к декларациям классов, полей, методов, ну и конечно же аннотаций.

Для описания новой аннотации используется ключевое слово **@interface**. Например :

public @interface Description {

String title();

int version() default 1;

String text();

}

Пример использования аннотации *Description* :

@Description(title="title", version=2, text="text")

public class Sample {

// ...

}

Пример аннотации с параметрами:

import java.lang.annotation.\*;

@Target(value=ElementType.FIELD)

@Retention(value= RetentionPolicy.RUNTIME)

public @interface Name

{

String name();

String type() default "string";

}

В данном примере **аннотация** включает в себя несколько полей (name, type), которые можно задать как обязательными, так и необязательными. В последнем случае подставляется *default* значение поля.

Из синтаксиса аннотации следует, что саму аннотацию можно пометить несколькими параметрами. В качестве типов параметров аннотации могут использоваться только примитивные типы, перечисления и класс String. Если у аннотации нет элементов, ее называют маркером (marker annotation type). В этом случае при использовании аннотации круглые скобки можно не писать.

**Параметры аннотации**

**@Retention**

Параметр @Retention позволяет определить жизненный цикл аннотации : будет она присутствовать только в исходном коде, в скомпилированном файле, или она будет также видна и в процессе выполнения. Выбор нужного типа аннотации **@Retention** зависит от того, как будет использоваться данная аннотацию. Например, генерировать что-то побочное из исходных кодов, или в процессе выполнения "стучаться" к классу через [reflection](http://java-online.ru/java-reflection.xhtml).

|  |  |
| --- | --- |
| RetentionPolicy.SOURCE | аннотация используется на этапе компиляции и должна отбрасываться компилятором |
| RetentionPolicy.CLASS | аннтоация будет записана в class-файл компилятором, но не должна быть доступна во время выполнения (runtime) |
| RetentionPolicy.RUNTIME | аннотация будет записана в class-файл и доступна во время выполнения через reflection |

**@Target**

Параметр @Target указывает, что именно должно быть помечено аннотацией. Это может быть поле, метод, тип и т.д. Для этого следует использовать параметры к аннотации.

|  |  |
| --- | --- |
| @Target(ElementType.PACKAGE) | только для пакетов |
| @Target(ElementType.TYPE) | только для классов |
| @Target(ElementType.CONSTRUCTOR) | только для конструкторов |
| @Target(ElementType.METHOD) | только для методов |
| @Target(ElementType.FIELD) | только для атрибутов(переменных) класса |
| @Target(ElementType.PARAMATER) | только для параметров метода |
| @Target(ElementType.LOCAL\_VARIABLE) | только для локальных переменных |

В случае, если необходимо, что бы **аннотация** использовалась больше чем для одного типа параметров, то можно указать @Target следующим образом:

@Target({ ElementType.PARAMETER, ElementType.LOCAL\_VARIABLE })

В данном случае аннотацию можно использовать только для параметров метода и для локальных переменных.

**@Documented**

Параметр @Documented указывает, что помеченные таким образом аннотацией класс/метод/поле должны быть добавлены в [javadoc](http://java-online.ru/java-javadoc.xhtml). Например, класс, помеченный аннотацией без @Documented, будет выглядеть так:

public class TestClass extends java.lang.Object

А если в описание аннотации добавить @Documented, получим:

@ControlledObject(name="name")

public class TestClass extends java.lang.Object

Пример своей аннотации.

Попробуем теперь написать рабочий пример с использованием аннотаций.  
Представим себе, что у нас есть какой-то самодельный проект, который на вход получает класс, специально заанотированный, чтобы проект мог управлять жизненным циклом объектов этого класса, и пусть там будут аннотации StartObject, StopObject для описания методов класса, и ControlledObject для описания самого класса. Последней аннотации дадим еще поле name, путь там хранится якобы имя для поиска.  
Аннотации будут выглядеть так:

@Target(value=ElementType.METHOD)

@Retention(value= RetentionPolicy.RUNTIME)

public @interface StartObject {

}

@Target(value=ElementType.METHOD)

@Retention(value= RetentionPolicy.RUNTIME)

public @interface StopObject {

}

@Target(value=ElementType.TYPE)

@Retention(value= RetentionPolicy.RUNTIME)

public @interface ControlledObject {

String name();

}

Напишем модуль, проверяющий подходит ли класс для загрузки в наш гипотетический проект или нет.  
Сперва определим сам проверяемый класс.

@ControlledObject(name="biscuits")

public class Cookies {

@StartObject

public void createCookie(){

//бизнес логика

}

@StopObject

public void stopCookie(){

//бизнес логика

}

}

Для того, чтобы работать с классом, сначала необходимо загрузить класс в контекст приложения. Используем:  
Class cl = Class.forName("org.annotate.test.classes.Cookies"); - это устаревшее написание, сейчас правильно писать = org.annotate.test.classes.Cookies.class;

так мы получили в объект “cl” все метаданные класса “Cookies” и теперь их из этого объекта (применяя к нему разные методы) можем получить.

Далее, через механизм reflection мы получаем доступ к полям и аннотациям класса.  
Проверим наличие аннотированных методов в классе и аннотации на самом классе:

if(!cl.isAnnotationPresent(ControlledObject.class)){ - ищем у класса установленную аннотацию “ControlledObject”

System.err.println("no annotation");

} else {

System.out.println("class annotated ; name - " + cl.getAnnotation(ControlledObject.class));

}

boolean hasStart=false;

boolean hasStop=false;

Method[] method = cl.getMethods();

for(Method md: method){ - дальше будем искать у конкретных методов установленные им конкретные аннотации

if(md.isAnnotationPresent(StartObject.class)) {hasStart=true;}

if(md.isAnnotationPresent(StopObject.class)) {hasStop=true;}

if(hasStart && hasStop){break;}

}

System.out.println("Start annotaton - " + hasStart + "; Stop annotation - " + hasStop );

Запустив, на выходе мы получим:  
Start annotaton — true; Stop annotation — true.  
Если попробовать убрать одну из аннотаций, то вывод сообщит о несоответствии требованиям.

**Еще одно Использование аннотации**

Предположим, нам нужно ограничить доступ к некоторым функциям веб-приложения для разных пользователей. Иными словами необходимо реализовать права (permissions). Для этого можно добавить следующее перечисление в класс пользователя:

public class User {

public static enum Permission {

USER\_MANAGEMENT, CONTENT\_MANAGEMENT

}

private List<Permission> permissions;

public List<Permission> getPermissions() {

return new ArrayList<Permission>(permissions);

}

// ...

}

Создадим аннотацию, которую будем использовать для проверки прав доступа :

@Retention(RetentionPolicy.RUNTIME)

public @interface PermissionRequired {

User.Permission value();

}

Предположим, что у нас есть некоторое действие, право на выполнение которого мы хотим ограничить, например, *UserDeleteAction*. Для этого добавляем аннотацию на это действие следующим образом:

@PermissionRequired(User.Permission.USER\_MANAGEMENT)

public class UserDeleteAction {

public void invoke(User user) { /\* \*/ }

}

Теперь используя [reflection](http://java-online.ru/java-reflection.xhtml), можно принимать решение, разрешать или не разрешать выполнение определенного действия :

User user = ...;

Class<?> actionClass = ...;

PermissionRequired permissionRequired = actionClass.getAnnotation(PermissionRequired.class);

if (permissionRequired != null){

if (user != null && user.getPermissions().contains(permissionRequired.value())){

// выполнить действие

}

}

1. **Анализатор аннотаций**

Создадим класс анализатора, который будет определять аннотации и выполнять некоторые действия, связанные с аннотируемыми параметрами. Необходимо иметь в виду, что если используется более чем одна пользовательская аннотации, то целесообразно иметь отдельный анализатор для каждой аннотации.

Что должен делать анализатор? Он использует **reflection** для доступа к аннотируемым данным. Пример анализатора для класса @Test:

public class AnnotationAnalyzer {

public void parse(Class<?> clazz) throws Exception {

Method[] methods = clazz.getMethods();

int pass = 0;

int fail = 0;

for (Method method : methods) {

if (method.isAnnotationPresent(Test.class)) {

try {

// вызов аннотируемого метода

method.invoke(null);

pass++;

} catch (Exception e) {

fail++;

}}}}}

Сочетание использования аннотации и [reflection](http://java-online.ru/java-reflection.xhtml) позволяет выполнить определенную проверку и вызвать метод на исполнение через **invoke**. Анализатор готов к использованию. Для использования атрибутов аннотации расширим код.

public class AnnotationAnalyzer {

public void analyze(Class<?> clazz) throws Exception {

Method[] methods = clazz.getMethods();

int pass = 0;

int fail = 0;

for (Method method : methods) {

if (method.isAnnotationPresent(Test.class)) {

// Получаем доступ к атрибутам

Test test = method.getAnnotation(Test.class);

Class expected = test.expected();

try {

method.invoke(null);

pass++;

} catch (Exception e) {

if (Exception.class != expected) {

fail++;

} else {

pass++;

}}}}}}

После получения доступа к атрибуту аннотации определяем ее значение. В нашем случае это значение типа Class, так как *expected* — это ожидаемая ошибка и мы будем получать *exception*.

Пример использования класса анализа аннотации:

public class Demo

{

public static void main(String [] args)

{

AnnotationAnalyzer analyzer = new AnnotationAnalyzer();

analyzer.analyze(MyTest.class);

}

}