Аннотации в Java

Выполнили студенты гр. 5030102/20201
Грушин А.Д.

Смирнова А.П.

Содержание

1. Основы аннотации
2. Типы аннотаций, используемые в языке Java
3. Аннотации, применимые к другим аннотациям
4. Пользовательские аннотации
5. Применение пользовательских аннотаций
6. Источники

Основы аннотации

Аннотации, форма метаданных, предоставляют данные о программе, которые не являются частью самой программы.

Аннотации не влияют на работу кода, который они аннотируют.

Возможное применение аннотаций:

- Информация для компилятора - аннотации могут использоваться компилятором для обнаружения ошибок или подавления предупреждений.
- Обработка во время компиляции и развертывания - программные инструменты могут обрабатывать информацию аннотаций для генерации кода, XML файлов и т.д.
- Обработка во время выполнения - некоторые аннотации доступны для обработки во время выполнения.

В простейшем виде аннотация выглядит следующим образом: @Entity.

Символ @ указывает компилятору, что то, что следует за ним, является аннотацией.

Пример:

```
@Override
void mySuperMethod { ... }
```

Аннотации могут содержать параметры, которые передают дополнительную информацию, которая помогает аннотации выполнить свою задачу.

Параметры указываются внутри круглых скобок ( ).

Пример:

```
@Author (
name = "Benjamin Franklin",
date = "3/27/2003"
)
class MyClass { ... }
```

Также верна следующая запись:

```
@Author ( name = "Benjamin Franklin" )
class MyClass { ... }
```

Если аннотация содержит один параметр, то имя параметра можно опустить:

```
@Author ( "Benjamin Franklin" )
class MyClass { ... }
```

Если параметры не содержатся вовсе, то скобки можно опустить.

Можно использовать несколько аннотаций к одному объявлению:

```
@Author ( name = "Jane Doe" )
@EBook
class MyClass { ... }
```

Типы аннотаций, используемые в Java

- @Deprecated - указывает на то, что помеченный элемент устарел и больше не должен использоваться. Компилятор выдает предупреждение каждый раз, когда программа использует элемент с аннотацией.
- @Override - информирует компилятор о том, что элемент предназначен для переопределения элемента, объявленного в суперклассе.

- @SuppressWarnings - сообщает компилятору о необходимости подавить определенные предупреждения. Например, при использовании устаревшего метода :)
- @SafeVarargs - утверждает, что код не выполняет потенциально опасных операций со своим параметром. При использовании этого типа аннотаций непроверенные предупреждения подавляются.

Аннотации, применимые к другим аннотациям

Такие аннотации называются мета-аннотациями. Они определяют где и как может быть использована аннотация.

- @Retention - указывает, как будет храниться помеченная аннотация.
- @Documented - указывает на то, что при ее использовании, элементы должны быть задокументированы с помощью Javadoc.

- @Target - ограничивает типы элементов, к которым может быть применена аннотация.
- @Inherited - указывает, что тип аннотации может быть унаследован от суперкласса.
- @Repeatable - позволяет использовать аннотацию несколько раз для одного элемента.

Пользовательские аннотации

Можно создавать свои аннотации.

Рассмотрим аннотацию @Company:

```
@Company ( name = "ABC", city = "XYZ" )
class CustomAnnotatedEmployee { ... }
```

Создать пользовательскую аннотацию можно с помощью ключевого слова @interface:

public @interface Company { ... }

Добавим мета-аннотации для указания информации об области действия и о типах элементов, к которым аннотация может быть применена:

@Target(ElementType.TYPE) //может быть применена к любому элементу класса

@Retention(RetentionPolicy.RUNTIME) //доступна в рантайме

public @interface Company { ... }

Осталось добавить атрибуты/параметры в аннотацию:

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
public @interface Company {
String name();
String city();
}
```

Применение пользовательских аннотаций

Пользовательские аннотации можно применять в различных сценариях программирования. Чтобы упростить задачи, повысить читаемость кода или автоматизировать рутинные процессы.

Рассмотрим примеры, как можно использовать пользовательские аннотации.

1. Используем аннотацию для маркировки методов, вызовы которых следует логировать:

```
@Target(ElementType.METHOD) //применяется только к методам
@Retention(RetentionPolicy.RUNTIME) //доступна в рантайме
public @interface Loggable

public class Service {
    @Loggable
    public void process() { ... }

    public void nonAnnotatedMethod() { ... }
}
```

Важно заметить.

Анотация @Loggable говорит о том, что помеченный метод необходимо логировать. Но это просто метка - Java не знает, что с ней делать.

Чтобы логирование сработало нужно написать код, который будет узнавать об аннотации и выполнит соответствующие действия. Такой код называется обработчиком аннотации.

Пометить метод аннотацией недостаточно!

Вызовем метод через базовый обработчик:

```
Method method = service.getClass().getMethod("process"); //получим
   необходимый метод
//проверим метод на наличие аннотации
if (method.isAnnotationPresent(Loggable.class)) {
   System.out.println("Loggable: Calling process()"); //выполняется
   дополнительная логика
   method.invoke(service); //вызывается метод
}
```

2. Используем аннотацию для проверки значения поля на соответствие заданным значениям:

@Target(ElementType.FIELD) //применяется только к полям
@Retention(RetentionPolicy.RUNTIME) //доступна в рантайме

public @interface MinAge { int value(); } //имеет параметр для хранения минимального возраста

```
public class User {
    @MinAge(18) // Минимальный возраст - 18
    public int age;

    public User(int age) { this.age = age; }
}
```

```
User user = new User(16); // Укажем возраст, который не соответствует минимальному
Field ageField = user.getClass().getField("age"); //получим значение поля age

// Проверяем, есть ли на поле аннотация @MinAge
if (ageField.isAnnotationPresent(MinAge.class)) {
        MinAge minAge = ageField.getAnnotation(MinAge.class);
        int minAgeValue = minAge.value(); // Получаем значение параметра аннотации
        int userAge = (int) ageField.get(user); // Получаем текущее значение поля age

        if (userAge < minAgeValue) {
          System.out.println("Validation failed: Age must be at least " + minAgeValue);
        } else {
          System.out.println("Validation passed: Age is " + userAge);
        }
}
```

3. Используем аннотацию для хранения конфигурационных данных, например, параметров подключения к БД:

```
@Target(ElementType.TYPE) // Применяется к классам
@Retention(RetentionPolicy.RUNTIME) // Доступна в рантайме
public @interface DatabaseConfig {
    String url();
    String username();
    String password();
}
@DatabaseConfig( url = "jdbc:mysql://localhost:3306/mydb", username = "root",
    password = "password" )
public class DatabaseService { ... }
```

```
Class<DatabaseService> clazz = DatabaseService.class;


if (clazz.isAnnotationPresent(DatabaseConfig.class)) { // Проверяем
  наличие аннотации
        DatabaseConfig config = clazz.getAnnotation(DatabaseConfig.class);
  // Получаем параметры
        System.out.println("URL: " + config.url());
        System.out.println("Username: " + config.username());
        System.out.println("Password: " + config.password());
}
```

Источники

- Аннотации в Java и их обработка / Хабр
- Урок: Аннотации (Уроки по Java™ > изучению языка Java)