Java中的注解（Annotations in Java）

Java项目的CodeQL数据库包含有关附加到程序元素的所有注释的信息。

关于使用注释（About working with annotations）

注释由以下CodeQL类表示：

- 类Annotatable表示所有可能附有注释的实体（即，包，引用类型，字段，方法和局部变量）。
- 类AnnotationType表示Java注释类型，例如java.lang.Override;。 注释类型是接口。
- 类AnnotationElement表示注释元素，即注释类型的成员。
- 类Annotation表示诸如@Override之类的注释； 批注值可以通过成员谓词getValue访问。

例如，Java标准库定义了一个注释SuppressWarnings，该注释指示编译器不要发出某些警告：

```
package java.lang;

public @interface SuppressWarnings {
    String[] value;
}
```

SuppressWarnings被表示为AnnotationType，其值是唯一的AnnotationElement。
SuppressWarnings的典型用法是此注释，以防止发出有关使用原始类型的警告：

```
class A {
    @SuppressWarnings("rawtypes")
    public A(java.util.List rawlist) {
    }
}
```

表达式@SuppressWarnings（“ rawtypes”）表示为注释。 字符串文字“ rawtypes”用于初始化注释元素的值，并且可以使用getValue谓词从注释中提取其值。
然后，我们可以编写此查询来查找附加到构造函数的所有@SuppressWarnings批注，并返回批注本身及其值元素的值：

```
import java

from Constructor c, Annotation ann, AnnotationType anntp
where ann = c.getAnAnnotation() and
    anntp = ann.getType() and
    anntp.hasQualifiedName("java.lang", "SuppressWarnings")
select ann, ann.getValue("value")
```

再举一个例子，该查询查找所有只有一个注解元素的注解类型，该注解元素的名称值为：

```
import java

from AnnotationType anntp
where forex(AnnotationElement elt |
    elt = anntp.getAnAnnotationElement() |
    elt.getName() = "value"
)
select anntp
```

示例：查找缺少的@Override注释

在Java的较新版本中，建议（尽管不是必需的）建议您使用@Override注释来注释覆盖另一个方法的方法。 这些注释（由编译器检查）用作文档，也可以帮助您避免在打算覆盖的地方意外重载。

例如，考虑以下示例程序：

```
class Super {
    public void m() {}
}

class Sub1 extends Super {
    @Override public void m() {}
}
```

```
8
9  class Sub2 extends Super {
10     public void m() {}
11 }
```

在这里，Sub1.m和Sub2.m都覆盖了Super.m，但是只有Sub1.m带有@Override注释。

现在，我们将开发一个查询来查找Sub2.m之类的方法，该方法应使用@Override进行注释，但不能使用。

首先，让我们编写一个查询，以查找所有@Override注释。 注释是表达式，因此可以使用getType访问它们的类型。 另一方面，注释类型是接口，因此可以使用hasQualifiedName查询其合格名称。 因此，我们可以像这样实现查询：

```
1 import java
2
3 from Annotation ann
4 where ann.getType().hasQualifiedName("java.lang", "Override")
5 select ann
```

与往常一样，最好在Java项目的CodeQL数据库上尝试此查询，以确保它确实产生一些结果。 在前面的示例中，它应该在Sub1.m上找到注释。 接下来，我们将@Override注释的概念封装为CodeQL类：

```
1 class OverrideAnnotation extends Annotation {
2     OverrideAnnotation() {
3         this.getType().hasQualifiedName("java.lang", "Override")
4     }
5 }
```

这使得编写查询来查找覆盖另一个方法的方法非常容易，但是没有@Override注解：我们使用谓词覆盖来确定一个方法是否覆盖另一个方法，并使用getAnAnnotation（可在任何Annotatable上使用）谓词来 检索一些注释。

```
1 import java
2
3 from Method overriding, Method overridden
4 where overriding.overrides(overridden) and
5     not overriding.getAnAnnotation() instanceof OverrideAnnotation
```

```
select overriding, "Method overrides another method, but does not
  have an @Override annotation."
```

示例：查找对不推荐使用的方法的调用（Example: Finding calls to deprecated methods）

再举一个例子，我们可以编写一个查询来查找对标有@Deprecated注解的方法的调用。
例如，考虑以下示例程序：

```
class A {
    @Deprecated void m() {}

    @Deprecated void n() {
        m();
    }

    void r() {
        m();
    }
}
```

在此，A.m和A.n都标记为已弃用。 方法n和r都调用m，但是请注意n本身已被弃用，因此我们可能不应该对此调用发出警告。
与前面的示例一样，我们将首先定义一个用于表示@Deprecated批注的类：

```
class DeprecatedAnnotation extends Annotation {
    DeprecatedAnnotation() {
        this.getType().hasQualifiedName("java.lang", "Deprecated")
    }
}
```

现在我们可以定义一个类来表示不赞成使用的方法：

```
class DeprecatedMethod extends Method {
    DeprecatedMethod() {
        this.getAnAnnotation() instanceof DeprecatedAnnotation
    }
```

```
}
```

最后，我们使用这些类查找对不赞成使用的方法的调用，不包括本身在不赞成使用的方法中出现的调用：

```
import java

from Call call
where call.getCallee() instanceof DeprecatedMethod
    and not call.getCaller() instanceof DeprecatedMethod
select call, "This call invokes a deprecated method."
```

在我们的示例中，此查询在A.r中标记对A.m的调用，但在A.n中不标记对A.m的调用。

优化（Improvements）

Java标准库提供了另一个注释类型java.lang.SupressWarnings，可用于禁止某些类别的警告。 特别是，它可以用于关闭有关不赞成使用的方法的调用的警告。 因此，有必要改进我们的查询，以忽略标有@SuppressWarnings（“ deprecated”）的内部方法对不赞成使用的方法的调用。
例如，考虑以下稍微更新的示例：

```
class A {
    @Deprecated void m() {}

    @Deprecated void n() {
        m();
    }

    @SuppressWarnings("deprecated")
    void r() {
        m();
    }
}
```

在这里，程序员已明确禁止有关A.r中不推荐使用的调用的警告，因此我们的查询不应再标记对A.m的调用。
为此，我们首先引入一个用于表示所有@SuppressWarnings批注的类，其中在禁止显示的警告列表中出现了不赞成使用的字符串：

```
class SuppressDeprecationWarningAnnotation extends Annotation {
    SuppressDeprecationWarningAnnotation() {
        this.getType().hasQualifiedName("java.lang", "SuppressWarn
ings") and
        this.getAValue().(Literal).getLiteral().regexpMatch(".*dep
recation.*")
    }
}
```

在这里，我们使用getAValue（）检索任何注释值：实际上，注释类型SuppressWarnings仅具有单个注释元素，因此每个@SuppressWarnings注释仅具有单个注释值。 然后，确保它是一个文字，使用getLiteral获取其字符串值，并使用正则表达式匹配检查它是否包含不赞成使用的字符串。

为了在现实世界中使用，必须对检查进行一些概括：例如，OpenJDK Java编译器允许@SuppressWarnings（“ all”）注释禁止显示所有警告。 我们可能还想通过将正则表达式更改为“。* \\ bdeprecation \\ b。*”来确保不赞成使用的是整个单词而不是另一个单词的一部分。

现在，我们可以扩展查询以筛选出带有SuppressDeprecationWarningAnnotation的方法中的调用：

```
import java

// Insert the class definitions from above

from Call call
where call.getCallee() instanceof DeprecatedMethod
    and not call.getCaller() instanceof DeprecatedMethod
    and not call.getCaller().getAnAnnotation() instanceof Suppress
DeprecationWarningAnnotation
select call, "This call invokes a deprecated method."
```