Java アノテーション

[Java][SJC-P]
参考・引用

・http://www.ibm.com/developerworks/jp/java/library/j-annotate1/
・http://www.ibm.com/developerworks/library/j-annotate2.html
・http://java.sun.com/j2se/1.5.0/ja/docs/ja/guide/language/annotations.html

・アノテーションは「@」記号で始まり、アノテーション名が後に続きます
・データが必要な時には、name=value の対としてアノテーションにデータを与えます

アノテーション、アノテーション・タイプ

Java 言語の概念で考えてみると、アノテーション･タイプはクラスと似ており、アノテーションは、そのクラスのインスタンスと似ている

アノテーションの種類

アノテーション	内容
マーカー・アノテーション	データは含まず、アノテーション名のみ。eg-@MarkerAnnotation
単一値アノテーション	一つのデータを持っている。eg-@SingleValueAnnotation("my data")
フル・アノテーション	複数のデータ・メンバーを持つ。eg-@FullAnnotation(var1="data value 1", var2="data value 2", var3="data value 3")

一つ以上の値を渡す必要がある時には、以下のことができる。

1. 名前と値の対を使う
2. 中括弧を使って、アノテーション変数に対して値の配列を与える

```
@TODOItems({    // 中括弧は、配列
  @TODO(
    severity=TODO.CRITICAL,
    item="Add functionality to calculate the mean of the student's grades",
    assignedTo="Brett McLaughlin"
  ),
  @TODO(
    severity=TODO.IMPOTANT,
    item="Print usage message to screen if no command-line flags specified",
    assignedTo="Brett McLaughlin"
  ),
  @TODO(
    severity=TODO.LOW,
    item="Roll a new website page with this class's new features",
    assignedTo="Jason Hunter"
  )
})
```

Override アノテーション

・メソッドに対してのみ使用
・このアノテーションで注釈を付けられたメソッドが、スーパークラスにあるメソッドをオーバーライドすることを表す

```
public class OverrideAnnotation {
    @Override
    public String toString() {
        return "OverrideAnnotation";
    }
}
```

例えば、toString を toStrings と打ち間違えると、コンパイルが通らない。

```
C:\work>javac OverrideAnnotation.java
OverrideAnnotation.java:2: メソッドはそのスーパークラスのメソッドをオーバーライドしません。
    @Override
    ^
エラー 1 個
```

Deprecated アノテーション

・@Deprecated を含むクラス自体はコンパイルしても何も特別なことは起こりません
・コンパイルした後で、使用すべきでないメソッドをオーバーライドしたり、呼び出したりして使おうとすると、コンパイラーはエラー・メッセージを出します。
・有効にするには、コンパイラに -deprecation オプションを指定する。

```
public class DeprecatedAnnotation {
    public static void main(String[] args) {
        (new DeprecatedAnnotationTest()).deprecatedMethod();
    }
}
class DeprecatedAnnotationTest {
    @Deprecated
    public void deprecatedMethod() {
        System.out.println("deprecated method");
    }
}
```

```
C:\work>javac -deprecation DeprecatedAnnotation.java
DeprecatedAnnotation.java:3: 警告 : [deprecation] DeprecatedAnnotationTest の deprecatedMethod() は推奨されません。
        (new DeprecatedAnnotationTest()).deprecatedMethod();
             ^
警告 1 個
```

SuppressWarnings アノテーション

・Java5.0 の generics によって、特に Java のコレクションに関して言うと、あらゆる種類のタイプセーフ操作ができるようになった
・ところが今度は generics のために、コレクションがタイプセーフ無しに使われると、コンパイラーは警告を投げるようになった
・Java5.0 用に書かれたコードには便利なのだが、Java 1.4.x やそれ以前のバージョン用に書かれたコードに対して頻繁に警告を受けるようになってしまう

```
import java.util.ArrayList;
import java.util.List;


public class SuppressWarningsAnnotation {
    public static void main(String[] args) {
        List l = new ArrayList();
        l.add("test");
    }
```

```
}
```

上記をコンパイルすると、コンパイルは行われるが、以下の警告

```
C:\work>javac SuppressWarningsAnnotation.java
注 : SuppressWarningsAnnotation.java の操作は、未チェックまたは安全ではありません。
注 : 詳細については、-Xlint:unchecked オプションを指定して再コンパイルしてください。
```

警告を抑制するには、@SuppressWarnings("unchecked") を与える。

```
@SuppressWarnings("unchecked")
public static void main(String[] args) {
    List l = new ArrayList();
    l.add("test");
}
```

・SuppressWarnings での変数の値には配列が使えるため、同じアノテーションで複数の警告を抑えることができる

```
@SuppressWarnings(value={"unchecked", "fallthrough"})
```

独自のアノテーションタイプを定義

・アノテーション型は、通常のクラスと同じように見えるが、いくつかの独特のプロパティを持っている。

@interface 宣言

・新しいアノテーション型は、interface キーワードの先頭に「@」記号が付加されることを除いて、interface に良く似ている。

```
package ann;
public @interface MarkSomething {
}
```

・コンパイルして、クラスパスに置いておけば、利用できます。

```
import ann.MarkSomething;
public class CustomAnnotationTest {
    @MarkSomething
    public void calcSomething(int x, int y) {
    }
}
```

・独自のアノテーションも、組み込みのものと同様に利用できます。
・ただし、import 宣言を行い、@ アノテーション型で、参照できるようにする必要があります。
・もしくは、完全名で指定します。

```
@ann.MarkSomething
public void doSomething() {
}
```

メンバーの追加

・アノテーション型は、メンバー変数を持つことができ、とても利用価値があります。
・特に洗練された利用法を行おうとするときには有用です。
・メタデータは、ただのドキュメントではありません。コード分析ツールに情報を提供することができます。
・アノテーション型のメンバーは、限定された情報を利用して作業の為にセットアップされます。
・メンバー変数や、アクセッサ、ミューテーターメソッドなどを定義する必要はありません。
・その代わり、ひとつのメソッドを定義します。
・データ型は、戻値である必要があります。

```
public @interface MarkSomething {
    String value();
}
```

使用例 -1 (value を省略)

```
@ann.MarkSomething("something message.")
public void doSomething() {
}
```

使用例 -2

```
@ann.MarkSomething(value="something message.")
public void doSomething() {
}
```

※ 使用例 -1 は、メンバーが、1 つで、名前が value の場合しか利用できない。

デフォルト値の設定

・メンバー宣言の後ろに default キーワードを宣言することで、デフォルト値を与えることができる。
・メンバーの型と同じである必要がある。
・enum も以下のように利用できる。

```
public @interface MarkSomething {
    public enum MarkImportant{ ERROR, INFO, DBUG };
    MarkImportant important() default MarkImportant.INFO;
    int level() default 3;
    String message();
}
```

使用例

```
@ann.MarkSomething(
        important=ann.MarkSomething.MarkImportant.DBUG,
        message="critical situation.")
public void doSomething() {
}
```