Java 8 - 类型注解

理解Java 8 类型注解需理解几个问题:

- 注解在JDK哪个版本中出现的，可以在哪些地方用注解?
- 什么是类型注解?
- 类型注解的作用是什么?
- 为什么会出现类型注解(JSR308)?

什么是类型注解

> 注解大家都知道，从java5开始加入这一特性，发展到现在已然是遍地开花，在很多框架中得到了广泛的使用，用来简化程序中的配置。那充满争议的类型注解究竟是什么? 复杂还是便捷?

1. 在java 8之前，注解只能是在声明的地方所使用，比如类，方法，属性;
2. java 8里面，注解可以应用在任何地方，比如:

创建类实例

```
new @Interned MyObject();
```

类型映射

```
myString = (@NonNull String) str;
```

implements 语句中

```
class UnmodifiableList<T> implements @Readonly List<@Readonly T> { … }
```

throw exception声明

```
void monitorTemperature() throws @Critical TemperatureException { … }
```

需要注意的是，**类型注解只是语法而不是语义，并不会影响java的编译时间，加载时间，以及运行时间，也就是说，编译成class文件的时候并不包含类型注解。**

类型注解的作用

先看看下面代码

```
Collections.emptyList().add("One");
int i=Integer.parseInt("hello");
System.console().readLine();
```

上面的代码编译是通过的，但运行是会分别报UnsupportedOperationException； NumberFormatException；NullPointerException异常，这些都是runtime error；

类型注解被用来支持在Java的程序中做强类型检查。配合插件式的check framework，可以在编译的时候检测出runtime error，以提高代码质量。这就是类型注解的作用了。

check framework是第三方工具，配合Java的类型注解效果就是1+1>2。它可以嵌入到javac编译器里面，可以配合ant和maven使用, 地址是http://types.cs.washington.edu/checker-framework/。 check framework可以找到类型注解出现的地方并检查，举个简单的例子:

```
import checkers.nullness.quals.*;
public class GetStarted {
    void sample() {
        @NonNull Object ref = new Object();
    }
}
```

使用javac编译上面的类

```
javac -processor checkers.nullness.NullnessChecker GetStarted.java
```

编译是通过，但如果修改成

```
@NonNull Object ref = null;
```

再次编译，则出现

```
GetStarted.java:5: incompatible types.
found   : @Nullable <nulltype>
required: @NonNull Object
        @NonNull Object ref = null;
                              ^
1 error
```

类型注解向下兼容的解决方案

如果你不想使用类型注解检测出来错误，则不需要processor，直接javac GetStarted.java是可以编译通过的，这是在java 8 with Type Annotation Support版本里面可以，但java 5,6,7版本都不行，因为javac编译器不知道@NonNull是什么东西，但check framework 有个向下兼容的解决方案，就是将类型注解nonnull用/**/注释起来，比如上面例子修改为

```
import checkers.nullness.quals.*;
public class GetStarted {
    void sample() {
        /*@NonNull*/ Object ref = null;
    }
}
```

这样javac编译器就会忽略掉注释块，但用check framework里面的javac编译器同样能够检测出nonnull错误。 通过类型注解+check framework我们可以看到，现在runtime error可以在编译时候就能找到。

关于JSR 308

JSR 308想要解决在Java 1.5注解中出现的两个问题:

- 在句法上对注解的限制: 只能把注解写在声明的地方
- 类型系统在语义上的限制: 类型系统还做不到预防所有的bug

JSR 308 通过如下方法解决上述两个问题:

- 对Java语言的句法进行扩充，允许注解出现在更多的位置上。包括: 方法接收器(method receivers，译注: 例public int size() @Readonly { … })，泛型参数，数组，类型转换，类型测试，对象创建，类型参数绑定，类继承和throws子句。其实就是类型注解，现在是java 8的一个特性
- 通过引入可插拔的类型系统(pluggable type systems)能够创建功能更强大的注解处理器。类型检查器对带有类型限定注解的源码进行分析，一旦发现不匹配等错误之处就会产生警告信息。其实就是check framework

对JSR308，有人反对，觉得更复杂更静态了，比如

```
@NotEmpty List<@NonNull String> strings = new ArrayList<@NonNull String>()>
```

换成动态语言为

```
var strings = ["one", "two"];
```

有人赞成，说到底，代码才是“最根本”的文档。代码中包含的注解清楚表明了代码编写者的意图。当没有及时更新或者有遗漏的时候，恰恰是注解中包含的意图信息，最容易在其他文档中被丢失。而且将运行时的错误转到编译阶段，不但可以加速开发进程，还可以节省测试时检查bug的时间。

总结

并不是人人都喜欢这个特性，特别是动态语言比较流行的今天，所幸，java 8并不强求大家使用这个特性，反对的人可以不使用这一特性，而对代码质量有些要求比较高的人或公司可以采用JSR 308，毕竟代码才是“最基本”的文档，这句话我是赞同的。虽然代码会增多，但可以使你的代码更具有表达意义。