代码规范和Android项目中的一些可用工具

刘聪 18 December 2014

这里主要讲一下关于代码规范的相关问题，和在Android项目中如何利用一些工具进行规范和检查。代码规范不是一个Android项目特有的问题，所以前部分内容是不单针对Android的。

什么是代码规范?

代码规范一般是指在编程过程中的一系列规则规范。

一般来说代码规范可以分为两种。

1. 一是编程语言本身在设计时所规定的一些原则，这类规则大部分都是强制的，像Python里用缩进表示逻辑块，Go里用首字母大小写表示可见度。
2. 另外一种是在一些组织约定的一些规范模式或个人在编写代码时的一些偏好，这种一般都是非强制的。比如大括号是放在方法名的同一行呢还是另起一行，不同的人有不同的想法，我也不知道谁好，所以别问我。

假如是强制的，大家暂时也不能反抗，只能吐嘈吐嘈，我们需要关心和规范的是非强制性规范。

一般情况下代码规范应该包括以下东西:

1. 代码格式和要求：注释，缩进，空格，空行
2. 约定的方法名，变量名，类名等等取名相关问题
3. 通用的一些模式和一些要避免的模式

为什么要有统一的代码规范?

假如你一个人单干，并且基本上也会这么继续下去的话，那这个东西对你不是太重要，随心就好，在这个流程里你就可以随意`continue`, `break`, `return`或`exit`。

我们先来看看以下几种代码规范不好的情况，虽然情况极端，但多少能看出些问题。

1. 一个项目里用Python作为主要编程语言。一部分人用tab做缩进，一部分人用空格做缩进，开发工具也是各种百花齐放。开始的时候大家各自忙着各自的模块，相安无事。有一天，B要在A写的一个文件里加几个方法，然后就会发现有可能程序跑不了，有可能在自己的编辑器里的缩进全乱了。
2. 在一个页面里有很多类似的按钮，A写的时候按勾就记2，按叉就记1。后来B写，按勾记1，按叉记0。再后来C来维护，就傻眼了，到底是要给1或2呢？

那么有一个好的代码规范有什么好处呢？

1. 风格统一，大家读起来舒心
2. 新同事加入项目后，不用再花很多心思去猜测各位前辈们的小技巧，缩短上手时间
3. 看或修改别人的代码，不用再去换IDE，或重新配置了
4. 检查别人代码的时候更加轻松，一般来讲，你比较容易知道那个方法在哪，是干啥的
5. 维护的时候会轻松些，一个项目里维护所处的周期比早期开发要更长，更耗时

简单来说，代码规范就是为了做到车同轨，书同文。最终的目的就是为了节约时间，提高效率，降低沟通成本。

什么是好的代码规范？

1. 好的代码规范是有语言特殊性和项目区别的。比如在Android项目中，必然需要引入一些Java中比较好的范例或模式，不能写着不像Java，反而像Python。假如项目是一个公共库，那么注释的要求就得提高些，因为别人要看的。但对于一些其他项目，这个要求可能不会那么高。
2. 好的代码规范必须得到团队里所有人的接受和认可。就像大家约好了，写字要用看得懂的中文，不能非有人为了耍酷，非要用火星文，看得大家好累。
3. 好的代码规范是可维护且合理的。a) 大家能轻松且明确知道一个东西有

没有违反规范；b) 不能为了执行这个规范把代码写得很长很长，得用非常不人道的方法去实现。

4. 好的代码规范必要小而精简。首先，繁锁而庞大的东西，大家大都记不住且不愿意记。其次，过于繁重反而会让大家花大量的时间在执行规范上。再次，太琐碎的东西会让人很烦，反心由然而起。再再次，扼杀个性或创新……总之，既然目的是要让愉快的写代码，那总不能让人不高兴吧。

套用一句广告来说，**适合自己的才是最好的**。

Android项目里的代码规范

到Android了，觉得无关的群众可以散了。

总体来说，Android的要求应该是要基于Java的。这里推荐Google和Sun的代码规范，大家可以借鉴或修改。

不论是在Android Studio或者Eclipse里，这些集成的IDE都对这些规范有很多人性化的提示，但它们在事后检查或可定制或跨IDE方面有些不足，所以我们需要一些辅助工具。

接下来这里会推荐几个代码分析工具帮助大家检查相关问题，主要环境是Android Studio和Gradle，但在Eclipse和Ant等其他环境下修改应该也能使用。

Checkstyle

checkstyle帮助开发者实现常用JAVA代码规范的自动化检查。它的功能比较丰富，相对配置起来比较复杂，你需要根据自己的需求配置你想检查的东西，比如Annotations，Block Checks，Class Design，Coding，Duplicate Code，Headers，Imports，Javadoc Comments，Metrics，Miscellaneous，Modifiers，Naming Conventions，Regexp，Size Violations，Whitespace。

首先，你要在build.gradle里加入plugin

```
apply plugin: 'checkstyle'
```

其次，由于gradle默认的checkstyle版本比较老，你需要更新版本才可以使用一些比较新的方法

```
checkstyle {
    toolVersion '6.1.1'
    showViolations true
}
```

再次，你需要把定义好的要求转化成config/checkstyle.xml里的一项项定义。比如你在这里就可以找到Google和Sun规范转化过来的xml。你应该按照自己的实际需求来配置这个文件。在checkstyle的官网上，你可以找到各项已经定义好的规则，添加到项目里。例如，这里我们要定义每行最大长度100个字符。

```
<module name="LineLength">
    <property name="max" value="100"/>
    <property name="ignorePattern" value="^package.*|^import.*|a href|h
</module>
```

现在你就可以加入任务了。注意要把刚刚添加的xml文件添加至configFile配置里。同时，你也可以用source指定你的源代码所在目录，并用include指明你希望检查的代码，用exclude排除你希望在检查中忽略的代码。

```
task checkStyle(type: Checkstyle) {
    configFile file("config/checkstyle.xml")
    source fileTree('src')
    include '**/*.java'
    exclude '**/gen/**'
}
```

Findbugs

findbugs是一个分析bytecode并找出其中可疑部分的一个工具。它给项目字节码做一个全面扫描，通过一些通用规则去判断可能潜在的一些问题，比如性能，多线程安全等等。有些时候它也会给出很详细的说明，为什么这种做法不大好，蛮有意思的。例如：

> *DLSYNCHRONIZATIONONSHAREDCONSTANT: Synchronization on interned String*
> *The code synchronizes on interned String.*

```
private static String LOCK = "LOCK";
...
synchronized(LOCK) { ...}
...
```

> *Constant Strings are interned and shared across all other classes loaded by the JVM. Thus, this could is locking on something that other code might also be locking. This could result in very strange and hard to diagnose blocking and deadlock behavior. See* *http://www.javalobby.org/java/forums/t96352.html* *and* *http://jira.codehaus.org/browse/JETTY-352*.

首先，在build.gradle里加入plugin

```
apply plugin: 'findbugs'
```

其次定义任务，这个也非常简单。你需要用classes指定临时生成的classes目录，source指定源代码所在目录。同时，你可以用xml.enabled或html.enabled指定你希望输出的报告格式。

```
    task findbugs(type: FindBugs) {
        ignoreFailures = true
        classes = fileTree('build/intermediates/classes/debug')
        source = fileTree('src')
        classpath = files()
        effort = 'max'
        reports {

            xml.enabled = false
            html.enabled = true
        }
    }
```

PMD

PMD也是一个静态代码分析工具，它主要用来分析一些常见问题。

PMD和Findbugs功能上有很多重叠的地方，二者区别主要体现在分析对象上，Findbugs扫描的是字节码，所以找到问题的级别有可能不一样。推荐大家可以各自选择，也可以两者都用

它有很多定义好的rule，例如在Android里，目前有三项规则:

1. CallSuperFirst，它会检查在Activity或Service里的子类里，是否在错误位轩调用父类onCreate等应该放在方法前的方法。
2. CallSuperLast，和CallSuperFirst，但它会检查一些应该在方法结束时才调用父类实现的情况。
3. DoNotHardCodeSDCard，这也是一个常见错误，你应该用Environment.getExternalStorageDirectory()而不是硬编码去取得扩展存储目录。

目前为止PMD对于Android的检查项并不多，使用它主要是用来检查一些JAVA中的常见错误。

首先，在build.gradle里加入plugin

```
apply plugin: 'pmd'
```

其次定义任务，ruleSets是需要检查的一些规则，大家可以根据需要自己修改。点击这里可以找到所有的预定义规则。

```
task pmd(type: Pmd) {
    source fileTree('src')

    ruleSets = [
        'java-android',
        'java-basic',
        'java-braces',
        'java-clone',
        'java-codesize',
        'java-comments',
        'java-controversial',
        'java-coupling',
        'java-design',
```

```
            'java-empty',
            'java-finalizers',
            'java-imports',
            'java-j2ee',
            'java-javabeans',
            'java-junit',
            'java-logging-jakarta-commons',
            'java-logging-java',
            'java-migrating',
            'java-naming',
            'java-optimizations',
            'java-strictexception',
            'java-strings',
            'java-sunsecure',
            'java-typeresolution',
            'java-unnecessary',
            'java-unusedcode'
        ]

        reports {
            xml.enabled = false
            html.enabled = true
        }
    }
```

总结

工具和方法很多，并且很多工具和方法有重复功能，大家都可以根据需要修改和调整。但需要切记：

1. 制定规范和使用规范的目的是为了让团队能一起愉快的写代码
2. 不能因为过多和繁锁的工具和规范让大家不能愉快的写代码

用户事件的存储与分析

许多时候我们说一款产品的设计是数据驱动的，是指许多产品方面的决策都是把用户行为量化后得出的。一个典例的例子就是注册流程的设计，如果用户需要填写的注册信息较多，一般就会分成多个页面，而产品设计师最关心的就是每个页面的流失率，从而不断的对这个流程作调整以达到信息量与流失率之间的平衡。 为了能够量化用户的行为，前提是要将各种用户事件都保存下来。其中最典型的事件包括user creation, page view和button click，但实际上还有许多其他事件，比如用户更改了状态或是录入了某些数据等等。

目前有许多第三方的服务可以帮助你做这方面的统计，国内有友盟，国外有Google…