Android 自定义Lint打造编码规则

|  |  |  |
| --- | --- | --- |
| 版本号 | 描述 | 日期(更新日期)/更新人 |
| 1.0 | 自定义Lint技术 | 2018/11/6 刘志保 |
|  |  |  |
| Github地址 | https://github.com/MMLoveMeMM/AngryPandaLint.git | |
| Reference : | <http://www.paincker.com/android-lint-2-implements>  <https://tech.meituan.com/android_custom_lint.html>[美团技术] | |

预备知识:

<1> : gradlew命令行使用;

<2> : Android 原生Lint的基本使用;

<3> : 推荐会gradle插件plugin开发;

为什么需要自定义Lint?

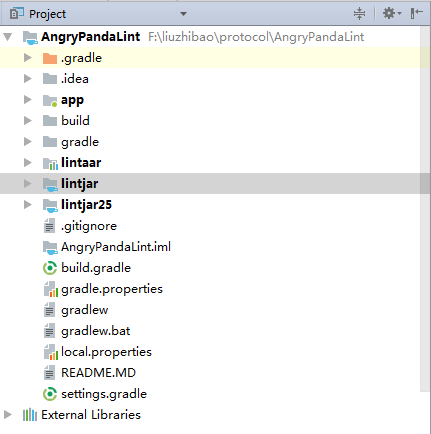
自定义Lint是根据当前的公司的具体项目进行定制的规则,比如说,现在工程中只能使用BHLog类进行日志打印,不允许Android.util.Log类打印,那么整个团队这么多人,虽然事先也约定了BHLog,但是开发过程可能还是不小心使用Android的Log去打印日志,这个时候默认的Lint是无法检查出来的.BHLog是有打印到文本的,但是Log只是在android studio中显示,但是不会打印到文本,这就会导致日志缺失,不利于解bug!以此类推,如果需要在工程中禁止使用什么类和方法都可以通过自定义Lint将其检查出来,规范开发流程和代码一致性.

前提,lintAPI 25版本和lintAPI 24版本区别挺大的,目前建议使用lintAPI 24版本,相关的开发资料较多,lintAPI 25似乎使用起来容易.开源工程中已经包含两个版本的开发,下面的是以lintAPI 25为例.

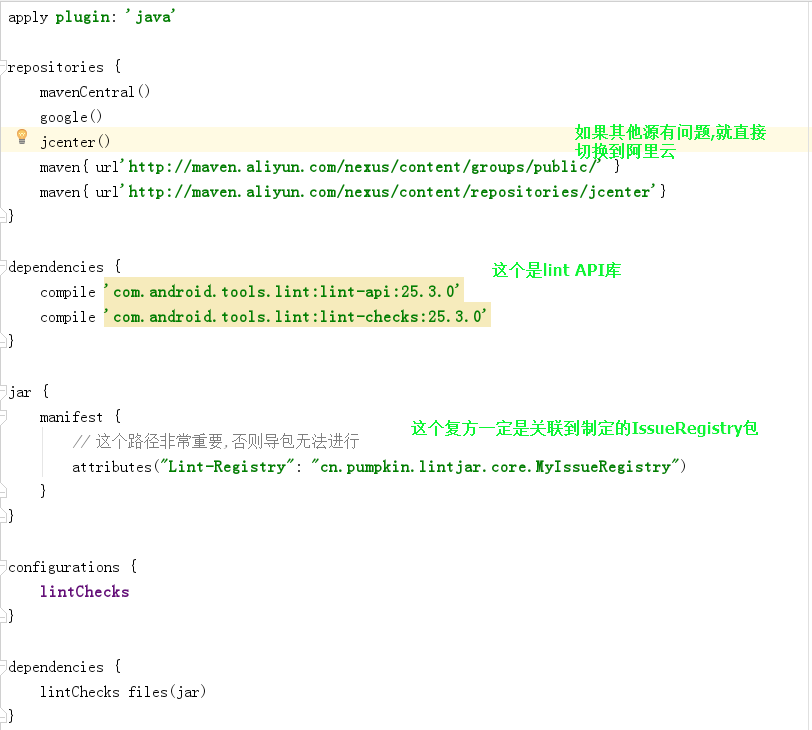
其次,默认lint和自定义lint可以同时存在.

工程从github上面直接下载上面的开源代码,加载到工程,也可以参考上面的网页,自己新建工程,都是新建工程基本功,不过多介绍.

加入到工程后,如下:

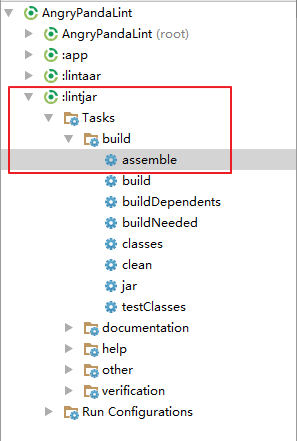


Lint程序在lintjar module中,lintjar的build.gradle脚本如下,其中下面的仓库一定要配置正确,否则依赖的库可能无法下载:

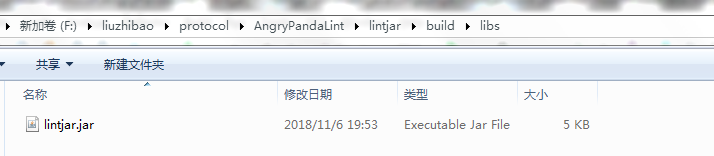


Lintjar module下面的代码参考提交在github开源代码.

程序编写好以后,开始编译这个lintjar,可以双击选中的,如下:



等待片刻,生成在下面的路径下:



下面如何使用这个lintjar.jar是关键,大概有两种方法:

<1> 方法一 :

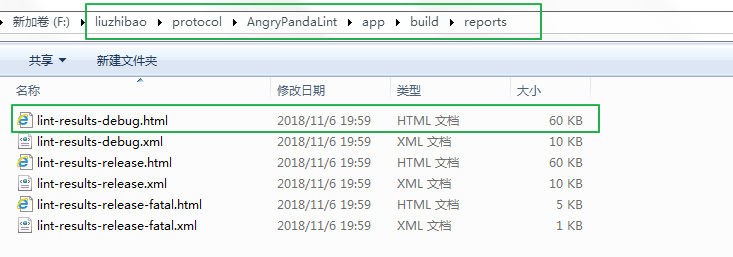
将上面生成的lintjar.jar拷贝到.android/lint目录下(C盘用户文件夹下),默认情况下.android的目录下没有lint文件夹,手动新建lint文件夹,将jar拷贝到该目录下,但是这个需要注意:把jar放在.android目录下,是会影响整个PC机上所有工程的lint功能. 设置了这个以后,AngryPandaLint根目录下,执行 :

-> gradlew lint

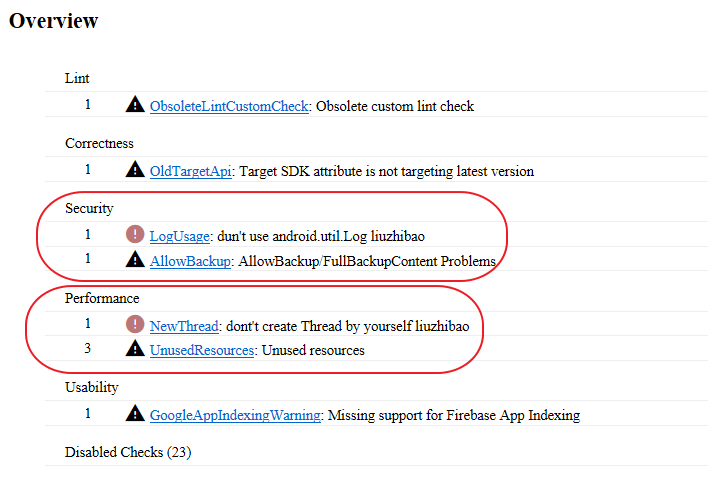
执行方式有多种:

|  |  |
| --- | --- |
| NO. | 方法描述 |
| 1 | 在Android Studio中，自定义Lint可以通过Inspections功能(Analyze - Inspect Code)手动运行 |
| 2 | 在Gradle命令行环境下，可直接用./gradlew lint执行Lint检查 |
| 3 | 手动执行简单易用，但缺乏强制性，容易被开发者遗漏 |

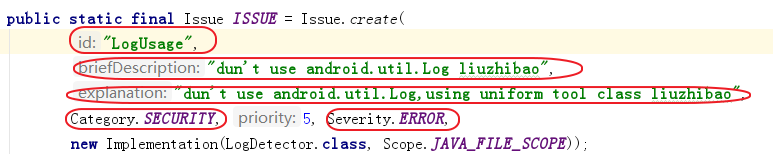
等待片刻,输出lint报告,如下:



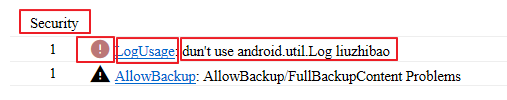
使用浏览器将上面制定的html打开,结果如下:



先看第一个红色的框框:看看下面的自定义lint是如何配置的.



对应下面的各个框框:



其中上面的红色图标是对应Severity.ERROR,这个提示信息即优先级定义:

每个Lint规则都可以配置Sevirity（优先级），包括Fatal、Error、Warning、Information等，我们主要使用Error和Warning，如下。

|  |  |
| --- | --- |
| 优先级别 | 描述 |
| Fatal |  |
| Error | 明确需要解决的问题，包括Crash、明确的Bug、严重性能问题、不符合代码规范等，必须修复 |
| Warning | 包括代码编写建议、可能存在的Bug、一些性能优化等，适当放松要求 |
| Information |  |

其他几个对应程序中的文字;

Category.SECURITY对应的是lint的Security;

<2> 第二种方式是采用LinkedIn提供的,将编译好的lint.jar放到新建的module library中,比如新建一个lintaar module,在这个module中引入编译好的lint.jar,并且配置好变量环境,然后在需要进行检查的工程中引入lintaar module,引入后在工程编译的时候,就会进行lint检查,同样生成报告.

Lint

需要lint检查的模块A

Lintaar

需要lint检查的模块B

需要lint检查的模块…

添加类似如下:

compile project(**path**: **":lintaar"**)

具体参考上面的工程代码.

<3> 第三种方法,就是自己编写gradle plugin自动部署,不需要人为那样拷贝[略];

Lint编译使用,看看自定义lint可以做哪些事情,先熟悉Lint API相关知识:

|  |  |
| --- | --- |
| Lint API类 | 描述 |
| Issue | 表示一个Lint规则。例如调用 Toast.makeText() 方法后，没有调用 Toast.show() 方法将其显示。 |
| IssueRegistry | 用于注册要检查的Issue列表。自定义Lint需要生成一个jar文件，其Manifest指向IssueRegistry类。 |
| Detector | 用于检测并报告代码中的Issue。每个Issue包含一个Detector。 |
| Scope | 声明Detector要扫描的代码范围，例如Java源文件、XML资源文件、Gradle文件等。每个Issue可包含多个Scope。 |
| Scanner | 用于扫描并发现代码中的Issue。每个Detector可以实现一到多个Scanner。自定义Lint开发过程中最主要的工作就是实现Scanner。 |
|  |  |

Scanner类: Lint中包括多种类型的Scanner如下，其中最常用的是扫描Java源文件和XML文件的Scanner;

|  |  |
| --- | --- |
| JavaScanner / JavaPsiScanner / UastScanner | 扫描Java源文件 |
| XmlScanner | 扫描XML文件 |
| ClassScanner | 扫描class文件 |
| BinaryResourceScanner | 扫描二进制资源文件 |
| ResourceFolderScanner | 扫描资源文件夹 |
| GradleScanner | 扫描Gradle脚本 |
| OtherFileScanner | 扫描其他类型文件 |

值得注意的是，扫描Java源文件的Scanner先后经历了三个版本。

最开始使用的是JavaScanner，Lint通过Lombok库将Java源码解析成AST(抽象语法树)，然后由JavaScanner扫描。

在Android Studio 2.2和lint-api 25.2.0版本中，Lint工具将Lombok AST替换为PSI，同时弃用JavaScanner，推荐使用JavaPsiScanner。

PSI是JetBrains在IDEA中解析Java源码生成语法树后提供的API。相比之前的Lombok AST，可以支持Java 1.8、类型解析等。使用JavaPsiScanner实现的自定义Lint规则，可以被加载到Android Studio 2.2+版本中，在编写Android代码时实时执行。

在Android Studio 3.0和lint-api 25.4.0版本中，Lint工具将PSI替换为UAST，同时推荐使用新的UastScanner。

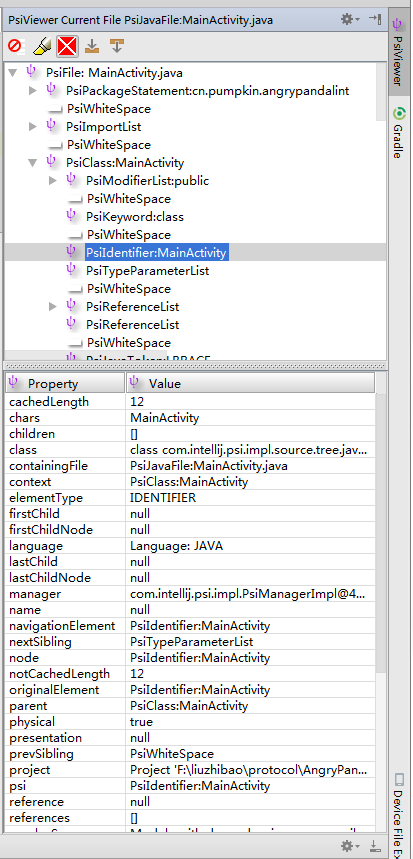
UAST是JetBrains在IDEA新版本中用于替换PSI的API。UAST更加语言无关，除了支持Java，还可以支持Kotlin。

本文目前仍然基于PsiJavaScanner做介绍。根据UastScanner源码中的注释，可以很容易的从PsiJavaScanner迁移到UastScanner。

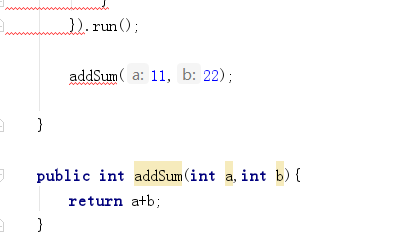
JavaPsiScanner类: JavaPsiScanner中包含6组、12个回调方法(下面标绿的名字对照到psi viewer结构图中);

1. : 当 getApplicablePsiTypes 返回了需要检查的Psi元素类型列表时，类型匹配的Psi元素( PsiElement )就会被 createPsiVisitor 返回的 JavaElementVisitor 检查。
2. : 当 getApplicableMethodNames 返回方法名的列表时，名称匹配的方法调用( PsiMethodCallExpression )就会被 visitMethod 检查。
3. : 当 getApplicableConstructorTypes 返回类名的列表时，类名匹配的构造语句( PsiNewExpression )就会被 visitConstructor 检查。
4. : 当 getApplicableReferenceNames 返回引用名的列表时，名称匹配的引用语句( PsiJavaCodeReferenceElement )就会被 visitReference 检查。
5. : 当 appliesToResourceRefs 返回true时，Java代码中的资源引用（例如 R.layout.main ）就会被 visitResourceReference 检查。
6. : 当 applicableSuperClasses 返回父类名的列表时，父类名匹配的类声明( PsiClass )就会被 checkClass 检查.

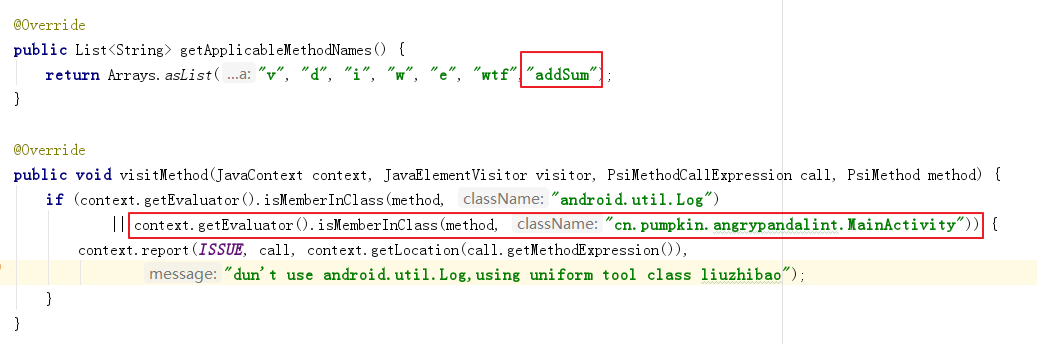
这个地方可能检查工程代码分不清类型,可以让Android studio安装psi插件工具psi viewer,安装好以后,如下(下面选中MainActivity.java):



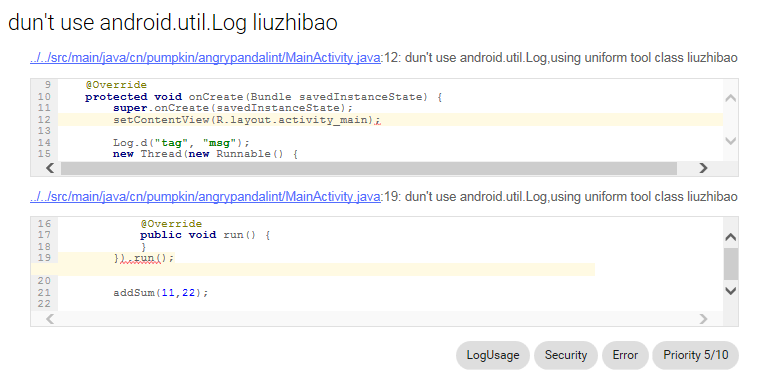
回到工程, LogDetector.java类是扫描工程中Java调用方法的检查.比如我再增加一个方法,MainActivity.java中:



那么这里检查addSum方法不能够被调用,修改LogDetector.java如下:

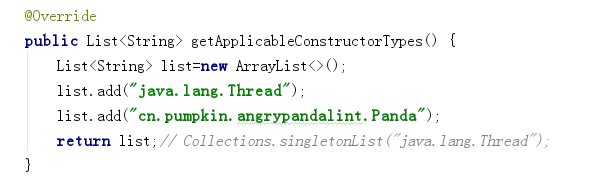


检查结果:



上面同理可以检查Toast对象是否使用,可以检查Toast的makeText方法是否被使用了.

同理,再增加一个构造方法的检查,新建一个Panda类,然后在MainActivity.java中使用,修改NewThreadDetector.java



测试结果:

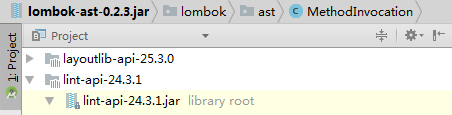


其他的几个结构以此类推.

具体对java和xml资源的检查请参考github提交的开源代码.

下面补充lintAPI 24版本,先要熟悉ast jar架包,了解Java 扫描过程和AST,并且根据扫描结果是否符合规则:

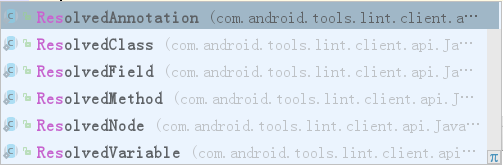
Java程序的扫描工作基本上都是通过javaParser(你要说成JavaScanner也可以)进行的,JavaParser类进行类分析并且生成”树节点”信息,有了这些树节点信息就可以用来比较是否是违禁的用法,比如程序中不准使用System.out.println(…),那么树节点信息列表中有这个信息返回,判断有,就可以report报警,提示开发人员.

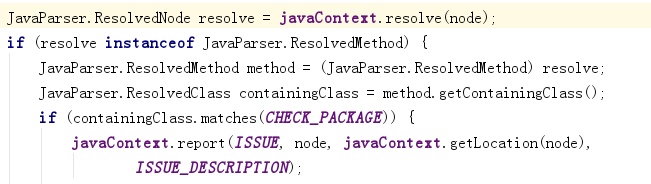


使用需要注意一下几个类:

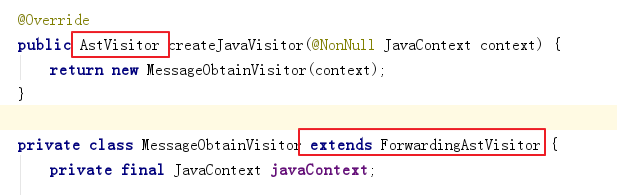
|  |  |
| --- | --- |
| AbstractNode | 这个类有大量的子类,比如ResolvedMethod, ResolvedClass,其他参考下图. |
| MethodInvocation | 需要转换JavaParser.ResolvedNode resolve = **javaContext**.resolve(node); 将其解析成一个com.android.tools.lint.client.api.JavaParser.ResolvedNode节点，ResolvedNode就包含了AST(抽象语法树)节点的类型解析信息. |
|  |  |

AbstractNode子类,这部分网上面基本上没有子类,全部要看lombok-ast.jar的源码:

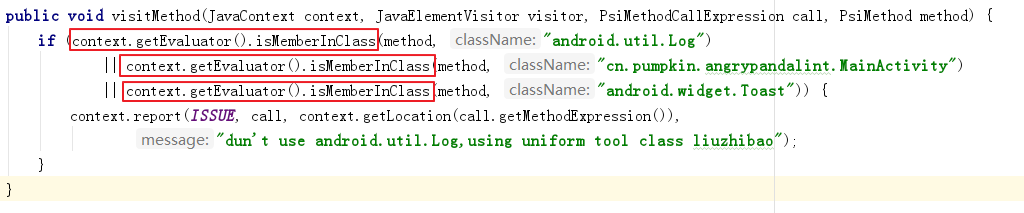


具体使用如下: 

整个程序需要实现一个自定义的AstVistor作为返回即可,这个是lintAPI 24版本绝大部分必须实现的:



在这一点上lintAPI 25就简单多了,只需要学会JavaEvaluator这个类就可以了,这个类可以通过返回的JavaContext. getEvaluator()返回获取.如下图:



轻轻松松的就可以完成比对.