Android混淆必知必会

日期	功能概要	编写者
2020/5/28	1、ProGuard作用 2、Android开启混淆 3、自定义混淆规则 4、混淆注意事项 5、Android默认混淆规则 6、Android通用混淆规则 7、Android资源压缩规则 8、后记补充	王宝忠

Android中的“混淆”可以分为两部分，一部分是 Java 代码的优化与混淆，依靠 proguard 混淆器来实现；另一部分是资源压缩，将移除项目及依赖的库中未被使用的资源(资源压缩严格意义上跟混淆没啥关系，但一般我们都会放一起讲)。混淆的好处就是它的目的：令 APK 难以被逆向工程，即很大程度上增加反编译的成本。此外，Android 当中的"混淆"还能够在打包时移除无用资源，显著减少 APK 体积。最后，还能以变通方式避免 Android 中常见的 **64k** 方法数引用的限制。

一、ProGuard作用

Proguard是一个Java类文件压缩器、优化器、混淆器、预校验器。压缩环节会检测以及移除没有用到的类、字段、方法以及属性。优化环节会分析以及优化方法的字节码。混淆环节会用无意义的短变量去重命名类、变量、方法。这些步骤让代码更精简，更高效，也更难被逆向（破解）。

压缩（Shrinking）：默认开启，用以减小应用体积，移除未被使用的类和成员，并且会在**优化**动作执行之后再次执行（因为优化后可能会再次暴露一些未被使用的类和成员）。

```
-dontshrink #关闭压缩
```

优化（Optimization）：默认开启，在字节码级别执行优化，让应用运行的更快。由于优化会进行类合并、内联等多种优化，建议关闭改选项，因为根据proguard-android-optimize.txt中的描述，优化可能会造成一些潜在风险，不能保证在所有版本的Dalvik上都正常运行。需使用热修复的应用，建议使用此配置关闭优化。

```
-dontoptimize  #关闭优化
-optimizationpasses 5 #表示proguard对代码进行迭代优化的次数，值在0-7之间，默认1次，Android一般为5
-optimizations !code/simplification/arithmetic,!code/simplification/cast,!field/*,!class/merging/* #优
```

混淆（Obfuscation）：默认开启，增大反编译难度，类和类成员会都会被替换成简短，随机名称，除非用keep保护。

```
-dontobfuscate #关闭混淆
-applymapping filename #根据指定的mapping映射文件进行混淆。
-obfuscationdictionary filename #指定字段、方法名的混淆字典，默认情况下使用abc等字母组合，比如根据自己的喜好
-classobfuscationdictionary filename #指定类名混淆字典。
-packageobfuscationdictionary filename #指定包名混淆字典
```

预校验（Preverify）：默认开启，这个预校验是作用在Java平台上的，Android平台上不需要这项功能，去掉之后还可以加快混淆速度（Android会把class编译成dex，并对dex文件进行校验，对class进行预校验是多余的）。

```
-dontpreverify #关闭预校验
```

二、Android开启混淆

Android Studio自身集成Java语言的ProGuard混淆工具，配合Gradle构建工具使用很简单，只需要在工程应用目录的gradle文件中设置minifyEnabled为true即可。因为开启混淆会使编译时间变长，所以debug模式下不要开启。

```
buildTypes {
    release {
        minifyEnabled true //开启混淆
        proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
    }
}
```

混淆规则是由两个文件共同配置，proguard-android-optimize.txt是默认配置文件，这个文件存放于<Android SDK>/tools/proguard目录下，属于所有项目共用，一般不要修改。我们可以到proguard-rules.pro文件中加入我们的自己混淆规则了 。混淆后默认会在工程目录app/build/outputs/mapping/release（debug）下生成一个mapping.txt文件，这就是混淆规则，我们可以根据这个文件把混淆后的代码反推回源本的代码，所以这个文件很重要，注意保护好。原则上，代码混淆后越乱越无规律越好，但有些地方我们是要避免混淆的，否则程序运行就会出错。

混淆后的类、方法名等等难以阅读，这固然会增加逆向工程的难度，但对追踪线上 crash 也造成了阻碍。我们拿到 crash 的堆栈信息后会发现很难定位，这时需要将混淆反解。

在 <sdk-root>/tools/proguard/ 路径下有附带的的反解工具（Window 系统为 proguardgui.bat，Mac 或 Linux 系统为 proguardgui.sh）。

这里以 Window 平台为例。双击运行 proguardgui.bat 后，可以看到左侧的一行菜单。点击 ReTrace，选择该混淆包对应的 mapping 文件（混淆后在 <module-name>/build/outputs/mapping/release/ 路径下会生成 mapping.txt 文件，它的作用是提供混淆前后类、方法、类成员等的对照表），再将 crash 的 stack trace 黏贴进输入框中，点击右下角的 ReTrace，混淆后的堆栈信息就显示出来了。

以上使用 GUI 程序进行操作，另一种方式是利用该路径下的 retrace 工具通过命令行进行反解，命令是

```
retrace.bat|retrace.sh [-verbose] mapping.txt [<stacktrace_file>]
```

例如：

```
retrace.bat -verbose mapping.txt obfuscated_trace.txt
```

三、自定义混淆规则

proguard中一共有三组六个keep关键字，很多人搞不清楚它们的区别，这里我们通过一个表格来直观地看下，移除是指在**压缩**(Shrinking)时是否会被删除。

保留	防止被移除或者被重命名	防止被重命名
保留类和类成员（变量/方法）	-keep	-keepnames
仅仅保留类成员（变量/方法）	-keepclassmembers	-keepclassmembernames
如果拥有某成员，保留类和类成员（变量/方法）	-keepclasseswithmembers	-keepclasseswithmembernames

keep格式：**-keep [,modifier,...] interface|class|enum classname [extends|implements [@annotationtype] classname]**

```
-keep class com.rush.Test
-keep interface com.rush.InterfaceTest
```

注意区别：建议使用keep class

- keep class表示类或接口
- keep interface仅表示接口

除此之外，proguard中的通配符也比较让人难懂，proguard-android.txt中就使用到了很多通配符，我们来看一下它们之间的区别：

通配符	描述
<field>	匹配类中的所有字段
<method>	匹配类中的所有方法
<init>	匹配类中的所有构造函数
?	匹配单个字符（任意字符），包名分隔符（.）除外
*	匹配任意长度字符（任意字符），但不含包名分隔符(.)。比如说我们的完整类名是com.example.test.MyActivity，使用com.*，或者com.exmaple.*都是无法匹配的，因为*无法匹配包名中的分隔符，正确的匹配方式是com.exmaple.*.*，或者com.exmaple.test.*，这些都是可以的。但如果你不写任何其它内容，只有一个*，那就表示匹配所有的东西。
**	匹配任意长度字符，并且**包含包名分隔符**(.)。比如proguard-android.txt中使用的-dontwarn android.support.**就可以匹配android.support包下的所有内容，包括任意长度的子包。
***	匹配任意**参数类型**。比如void set*(***)就能匹配任意传入的参数类型，*** get*()就能匹配任意返回值的类型。

…	匹配任意长度的任意**参数类型**。比如void test(…)就能匹配任意void test(String a)或者是void test(int a, String b)这些方法。

其他规则

其他规则	描述
-dontusemixedcaseclassnames	不使用大小写混用的类名，默认情况下混淆后的类名可能同时包含大写小字母。这在某些对大小写不敏感的系统（如windowns）上解压时，可能存在文件被相互覆盖的情况。
-keepattributes [attribute_filter]	保留指定属性，多个属性可以用多个-keepattributes配置，也可以用逗号分隔，可以使用? * **通配符，并且可以使用否定符（!）。
-keeppackagenames [package_filter]	不混淆指定的包名，多个包名可以用逗号分隔，可以使用? * **通配符，并且可以使用否定符（!）。 #不混淆泛型，不混淆匿名内部类等 -keepattributes Exceptions,InnerClasses,Signature #抛出异常时保留代码行号 -keepattributes SourceFile,LineNumberTable #使用到注解时也需要keep -keepattributes *Annotation*
-keepparameternames	保留已经被keep的方法的参数类型和参数名称，在混淆library库时非常有用，可供IDE帮助用户进行信息提示和代码自动填充。
-verbose	指定在混淆过程中输出更多信息，配置这个选项后，在遇到异常时，将输出完整的堆栈，而不仅仅是异常消息。
-dontnote [class_filter]	指定不输出潜在的错误或者遗漏，比如拼写错误或者缺失了有用的信息。class_filter是一个正则表达式，匹配到类将不输出这些信息。
-dontwarn [class_filter]	指定一组类，不警告这些类中找不到引用或其它重要的问题。这个选项是很危险的，比如，找不到引用的错误可能导致代码不能正常work。 （在引用一些存在警告的jar包，这个选项比较有用。）
-ignorewarnings	指定输出所以警告信息，但继续进行混淆。同上一选项，慎用。
-printconfiguration [filename]	指定输出整个过程中的所有配置，输出到标准输出流或者指定文件中。这有时候在调度配置时有用。
-dump [filename]	指定在任一处理过程后，输出class文件的结构，可以输出到标准输出流或者指定文件中。

常用混淆规则举例如下：

- 一颗星*：表示只保持该包下的类名，而子包下的类名还是会被混淆

```
-keep class com.thc.test.*
```

- 两颗星**：表示把本包和所含子包下的类名都保持

```
-keep class com.thc.test.**
```

上面两种方式保持类后，会发现类名虽然未混淆，但里面的具体方法和变量命名还是变了

- 既可以保持该包下的类名，又可以保持类里面的内容不被混淆

```
-keep class com.thc.test.*{*;}
```

- 既可以保持该包及子包下的类名，又可以保持类里面的内容不被混淆

```
-keep class com.thc.test.**{*;}
```

- 保持某个特定类名不被混淆（但是内部内容会被混淆）

```
-keep class com.xlpay.sqlite.cache.BaseDaoImpl
```

- 保持某个特定 类名及所有内容不会混淆

```
-keep class com.xlpay.sqlite.cache.BaseDaoImpl{*;}
```

- 保持类中特定内容不混淆，而不是所有的内容可以使用如下

```
-keep class com.thc.gradlestudy.MyProguardBean{
    <init>; #匹配所有构造器
    <fields>;#匹配所有域
    <methods>;#匹配所有方法
 }
```

上面就保持住了MyProguardBean这个类中的所有的构造方法、变量、和方法

- 可以在<fields>，<methods>和<init>前面加上private 、public、native等修饰符来进一步指定不被混淆的内容，或者可以在<init>后面加人参数。

```
-keep class com.xlpay.sqlite.cache.BaseDaoImpl{
     public <methods>;#保持该类下所有的共有方法不被混淆
     public *;#保持该类下所有的共有内容不被混淆
     private <methods>;#保持该类下所有的私有方法不被混淆
     private *;#保持该类下所有的私有内容不被混淆
     public <init>(java.lang.String);#保持该类的String类型的构造方法，仅限于构造方法
 }
```

- 在特定方法后加入参数，限制特定的方法

```
-keep class com.thc.gradlestudy.MyProguardBean{
     public void test(String);#只保留改方法
 }

-keepclassmembers public class * extends android.view.View {
   void set*(***); #保留该类所有set方法
   *** get*(); #保留该类所有get方法
}
```

- 要保留一个类中的内部类不被混淆需要用 $ 符号

```
#保持ProguardTest中的MyClass不被混淆
-keep class com.xlpay.sqlite.cache.ProguardTest$MyClass{*;}
```

- 使用Java的基本规则来保护特定类不被混淆，比如用extends，implement等这些Java规则

```
-keep public class * extends android.app.Activity #不混淆所有activity
-keep class * implements com.example.TestInterface { *; } #不混淆某个接口的实现
```

- 如果不需要保持类名，只需要保持该类下的特定方法保持不被混淆，需要使用keepclassmembers，而不是keep，因为keep方法会保持类名。

```
#保持ProguardTest类下test(String)方法不被混淆
-keepclassmembernames class com.xlpay.sqlite.cache.ProguardTest{
     public void test(java.lang.String);
}
```

- 如果拥有某成员，保留类和类成员（如下就是保留所有含有native方法的类的类名和native方法名，而如果某个类中没有含有native方法，那就还是会被混淆。）

```
-keepclasseswithmember class * {
    native <methods>;
}
```

-

- -libraryjars：引用jar包，相当于类文件中的import，当你需要在proguard脚本中，配置jar包相关的混淆时，就得通过 -liraryjars引入对应jar包

```
#引入第三方jar
-libraryjars libs/android-support-v4.jar
#不混淆第三方jar包中的类
-keep class android.support.v4.** {*;}
```

四、混淆注意事项

- jni方法不可混淆，因为native方法是要完整的包名类名方法名来定义的，不能修改，否则找不到；

```
#保持native方法不被混淆
 -keepclasseswithmembernames class * {
     native <methods>;
 }
```

- 反射用到的类混淆时需要注意：**只要保持反射用到的类名和方法即可，并不需要将整个被反射到的类都进行保持**
- 注解用了反射，所以不能混淆
- AndroidMainfest中的类不混淆，所以四大组件和Application的子类和Framework层下所有的类不要进行混淆。
- 自定义View也是带了包名写在xml布局中，也不能被混淆。
- 实体类不能被混淆，经常伴随着序列化与反序列化操作，或者使用GSON、fastjson等框架解析服务端数据时，会用到反射。
- 使用第三方开源库或者引用其他第三方的SDK包时，如果有特别要求，也需要在混淆文件中加入对应的混淆规则；
- 有用到WebView的JS调用也需要保证写的接口方法不混淆，原因和第一条一样；
- Parcelable的子类和Creator静态成员变量不混淆，否则会产生Android.os.BadParcelableException异常；

```
-keep class * implements Android.os.Parcelable {
    # 保持Parcelable不被混淆
    public static final Android.os.Parcelable$Creator*;
 }
```

- 使用enum类型时需要注意避免以下两个方法混淆，因为enum类的特殊性，以下两个方法会被反射调用，见第

二条规则。

```
-keepclassmembers enum * {
     public static **[] values();
     public static ** valueOf(java.lang.String);
 }
```

- 建议：发布一款应用除了设minifyEnabled为ture，你也应该设置zipAlignEnabled为true，像Google Play强制要求开发者上传的应用必须是经过zipAlign的，zipAlign可以让安装包中的资源按4字节对齐，这样可以减少应用在运行时的内存消耗。

五、Android默认混淆规则

官方的proguard-android.txt文件，位于sdk/tools/proguard目录下，这是一份默认的混淆规则，适用于所有项目，所以自定义的混淆规则可以不用再重复添加了，规则如下：

```
-optimizations !code/simplification/arithmetic,!code/simplification/cast,!field/*,!class/merging/*
-optimizationpasses 5
-allowaccessmodification #优化时允许访问并修改有修饰符的类和类的成员

-dontoptimize #不进行优化
-dontpreverify #不进行预校验

-dontusemixedcaseclassnames #不使用大小写混合类名
-dontskipnonpubliclibraryclasses #指定不去忽略非公共库的类
-dontskipnonpubliclibraryclassmembers#指定不去忽略非公共库的类
-verbose #混淆过程中打印日志

-keepattributes *Annotation* #保留注解
-keep public class com.google.vending.licensing.ILicensingService #keep google license服务接口
-keep public class com.android.vending.licensing.ILicensingService

-keepclasseswithmembernames class * {  #保留native方法及其所属类
    native <methods>;
}

-keepclassmembers public class * extends android.view.View { #保留自定义view的get/set方法
   void set*(***);
   *** get*();
}
# 保留继承自Activity中所有包含public void *(android.view.View)签名的方法，如onClick
-keepclassmembers class * extends android.app.Activity {
   public void *(android.view.View);
}

# 保留枚举中的values和valueOf方法
-keepclassmembers enum * {
    public static **[] values();
    public static ** valueOf(java.lang.String);
```

```
}
# 保留Parcelable的CREATOR成员
-keepclassmembers class * implements android.os.Parcelable {
  public static final android.os.Parcelable$Creator CREATOR;
}
#保留R文件的静态字段
-keepclassmembers class **.R$* {
    public static <fields>;
}
#不输出support包中的警告
-dontwarn android.support.**

#保留keep相关注解
-keep class android.support.annotation.Keep
-keep @android.support.annotation.Keep class * {*;}

-keepclasseswithmembers class * {
    @android.support.annotation.Keep <methods>;
}

-keepclasseswithmembers class * {
    @android.support.annotation.Keep <fields>;
}

-keepclasseswithmembers class * {
    @android.support.annotation.Keep <init>(...);
}

```

6、Android通用混淆规则

除了默认混淆规则外，有些规则是每个项目都是需要的，这里整理出一份通用规则，以后写混淆文件，可以直接拷贝使用，除非有特殊的需求，基本不用改动。如下规则可以安装

AndroidProguard Pro插件实现一键生成。

```
#----------------------------------------基本不用动区域----------------------------------------
#--------------------------------基本指令区--------------------------------
-keepattributes *Annotation*,InnerClasses #不混淆匿名内部类，注解
-keepattributes Signature #保留泛型
-keepattributes SourceFile,LineNumberTable #保留行号
-keepparameternames #保留方法参数名
#----------------------------------------------------------------------------

#--------------------------------默认保留区--------------------------------
-keep public class * extends android.app.Activity
-keep public class * extends android.app.Application
-keep public class * extends android.app.Service
-keep public class * extends android.content.BroadcastReceiver
-keep public class * extends android.content.ContentProvider
```

```
-keep public class * extends android.app.backup.BackupAgentHelper
-keep public class * extends android.preference.Preference
-keep public class * extends android.view.View
-keep public class com.android.vending.licensing.ILicensingService
-keep class android.support.** {*;}

-keep public class * extends android.view.View{
    *** get*();
    void set*(***);
    public <init>(android.content.Context);
    public <init>(android.content.Context, android.util.AttributeSet);
    public <init>(android.content.Context, android.util.AttributeSet, int);
}
-keepclasseswithmembers class * {
    public <init>(android.content.Context, android.util.AttributeSet);
    public <init>(android.content.Context, android.util.AttributeSet, int);
}

-keepclassmembers class * implements java.io.Serializable {
    static final long serialVersionUID;
    private static final java.io.ObjectStreamField[] serialPersistentFields;
    private void writeObject(java.io.ObjectOutputStream);
    private void readObject(java.io.ObjectInputStream);
    java.lang.Object writeReplace();
    java.lang.Object readResolve();
}
#R文件类不混淆
-keep class **.R$* {
 *;
}

-keepclassmembers class * {
    void *(*Event);
}
# support-v4
#https://stackoverflow.com/questions/18978706/obfuscate-android-support-v7-widget-gridlayout-issue
-dontwarn android.support.v4.**
-keep class android.support.v4.app.** { *; }
-keep interface android.support.v4.app.** { *; }
-keep class android.support.v4.** { *; }


# support-v7
-dontwarn android.support.v7.**
-keep class android.support.v7.internal.** { *; }
-keep interface android.support.v7.internal.** { *; }
-keep class android.support.v7.** { *; }

# support design
#@link http://stackoverflow.com/a/31028536
-dontwarn android.support.design.**
```

```
-keep class android.support.design.** { *; }
-keep interface android.support.design.** { *; }
-keep public class android.support.design.R$* { *; }
#---------------------------------------------------------------------------

#---------------------------------webview------------------------------------
-keepclassmembers class fqcn.of.javascript.interface.for.Webview {
   public *;
}
-keepclassmembers class * extends android.webkit.WebViewClient {
    public void *(android.webkit.WebView, java.lang.String, android.graphics.Bitmap);
    public boolean *(android.webkit.WebView, java.lang.String);
}
-keepclassmembers class * extends android.webkit.WebViewClient {
    public void *(android.webkit.WebView, jav.lang.String);
}
#---------------------------------------------------------------------------
#-------------------------------------------------------------------------------------------
```

需要自己添加的规则，例如：

```
#---------------------------------实体类---------------------------------

#---------------------------------第三方包-------------------------------

#---------------------------------反射相关的类和方法-----------------------

#---------------------------------与js互相调用的类-----------------------
```

七、Android资源压缩规则

Android 中，编译器为我们提供了另外一项强大的功能：资源的压缩。资源压缩能够帮助我们移除项目及依赖仓库中未使用到的资源，有效地降低了apk包的大小。由于资源压缩与代码混淆是协同工作，所以，如果需要开启资源的压缩，切记要先开启代码混淆，否则会出现以下问题：

```
android {
    buildTypes {
        release {
            minifyEnabled true //先开启混淆
            shrinkResources true //开启资源压缩
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }
}
```

（1）自定义要保留的资源

当我们开启了资源压缩之后，系统会默认替我们移除所有未使用的资源，假如我们需要保留某些特定的资源，可以在我们项目中创建一个被 <resources> 标记的 XML 文件（如 res/raw/keep.xml），并在 tools:keep 属性中指定每个要保留的资源，在 tools:discard 属性中指定每个要舍弃的资源。这两个属性都接受逗号分隔的资源名称列表。同样，我们可以使用字符 * 作为通配符。如：

```
<?xml version="1.0" encoding="utf-8"?>
<resources xmlns:tools="http://schemas.android.com/tools"
    tools:keep="@layout/activity_video*,@layout/dialog_update_v2"
    tools:discard="@layout/unused_layout,@drawable/unused_selector" />
```

（2）启用严格检查模式

正常情况下，资源压缩器可准确判定系统是否使用了资源。不过，如果您的代码（包含库）调用 Resources.getIdentifier()，这就表示您的代码将根据动态生成的字符串查询资源名称。这时，资源压缩器会采取防御性行为，将所有具有匹配名称格式的资源标记为可能已使用，无法移除。例如，以下代码会使所有带 img_ 前缀的资源标记为已使用：

```
String name = String.format("img_%1d", angle + 1);
res = getResources().getIdentifier(name, "drawable", getPackageName());
```

我们可以设置 `tools:shrinkMode` 为 `strict` 来开启严格模式，使只有确实被使用的资源被保留。

（3）移除备用资源

Gradle 资源压缩器只会移除未被应用引用的资源，这意味着它不会移除用于不同设备配置的备用资源。必要时，我们可以使用 Android Gradle 插件的 resConfigs 属性来移除您的应用不需要的备用资源文件（常见的有用于国际化支持的 strings.xml，适配用的 layout.xml 等）：

```
android {
    defaultConfig {
        ...
        //保留中文和英文国际化支持 其他未显式声明的语言资源将被移除
        resConfigs "en", "zh"
    }
}
```

八、后记补充

（1）例如：classA和classB，在加混淆的情况下多种结果：

1. 如果classA没有被keep，则不会看到classA的class文件
2. 如果classA没有被keep，classB被保持，同时classB引用到了classA，这个时候能够看到被混淆的classA的class文件，如显示为a
3. 如果classA中通过反射，获取到classB，那么classB的类名及反射用到的方法必须keep住

4. jar包混淆，暴露出的类、方法、方法的参数需要keep住
5. 情况说明：工程Demo依赖了 小米渠道的依赖，小米依赖又依赖了Common，对Common进行混淆但是不对小米渠道混淆，那么小米的依赖中使用到的Common中的类都需要keep住

（2）关于反射（待验证）

并不是所有会被反射引用的类都必须keep，在progurad过程中能直接分析到引用的类会被proguard做相应的处理：

Class.forName的类名"SomeClass"被混淆后自动替换

Class.forName("SomeClass")

建议，所有会被反射引用的类都做好keep（虽然有些反射能被正确处理）。 如native方法，四大组件，接口model，枚举，序列化类等。

（3）打AAR混淆包

library打aar混淆包，只需要添加基本的规则，和自己代码相关混淆设置（包括libs里的jar）；不需要设置引入的第三方AAR和maven依赖的包，并且第三方aar只能用compileOnly依赖，否则第三方包部分类会被打进aar（不打混淆包没事）。如下是通用规则

```
-dontoptimize
# 不进行预校验,Android不需要,可加快混淆速度。
-dontpreverify
# 屏蔽警告
-ignorewarnings
#不进行压缩，防止类被移除
-dontshrink

#----------------------------------基本指令区----------------------------------
-keepattributes *Annotation*,InnerClasses #不混淆匿名内部类，注解
-keepattributes Signature #保留泛型
-keepattributes SourceFile,LineNumberTable #保留行号
-keepparameternames #保留方法参数名
-repackageclasses '自定义包名，可为空'#没有被保持的类会被打进该包中
```

（4）打APK混淆包

依赖的远程库，本地AAR包，本地library，jar包等都会被混淆，混淆规则是由各种依赖库中consumerProguardFiles定义的规则合集，然后再统一混淆。并不是依赖库先按自己的规则混淆，然后再合并按主规则再混淆一次。总共一起就混淆一次，如果依赖库已经是混淆过的，可以keep排除掉。