1 Configuration file structure 配置文件结构

1.1 Input tag输入标记

1.2 Classpath tag类路径标记

1.3 Keep-names tag保持名称标记

1.3.1 class tag类标记

1.3.2 field tag字段标记

1.3.3 method tag 方法标记

1.4 Watermark tag水印标记

1.5 Expiry tag期满标记

1.6 Property tag属性标记

1.6.1 General properties一般属性

1.6.1.1 log-file 日志文件

1.6.1.2 random-seed随机种子

1.6.2 String encryption properties字符串加密属性

1.6.2.1 string-encryption字符串加密

1.6.2.2 string-encryption-type字符串加密类型

1.6.2.3 string-encryption-version字符串加密版本

1.6.2.4 string-encryption-ignored-strings字符串-加密-忽略-字符串

1.6.3 Control flow obfuscation properties控制流模糊处理属性

1.6.3.1 control-flow-obfuscation控制流模糊处理

1.6.3.2 extensive-flow-obfuscation 广泛流混淆

1.6.4 Renaming properties 重命名属性

1.6.4.1 default-package 默认包

1.6.4.2 force-default-package 强制-默认-包

1.6.4.3 packages-naming 包命名

1.6.4.4 classes-naming 类命名

1.6.4.5 methods-naming 命名方法

1.6.4.6 fields-naming 字段命名

1.6.4.7 classes-naming-prefix类命名前缀

1.6.4.8 methods-naming-prefix方法命名前缀

1.6.4.9 fields-naming-prefix字段命名前缀

1.6.4.10 local-variables-naming本地变量命名

1.6.4.11 skip-renaming跳过重命名

1.6.4.12 update-resource-names 更新-资源-名称

1.6.4.13 update-resource-contents 更新资源的内容

1.6.5 Other properties其他属性

1.6.5.1 line-numbers行号

1.6.5.2 generics泛型

1.6.5.3 inner-classes 内部类

1.6.5.4 member-reorder成员重新排序

1.6.5.5 finalize完成

1.6.5.6 version-marker版本标记

1.6.5.7 synthetize-methods合成方法

1.6.5.8 synthetize-fields合成字段

1.6.5.9 remove-toString 删除 toString方法

1.6.5.10 remove-calls 删除函数调用

1.6.6 Incremental obfuscation properties增量模糊处理属性

1.6.6.1 incremental-obfuscation增量模糊处理

1.6.6.2 unique-renaming独特重命名

1.7 Ignore-classes tag忽略类标记

2 Annotations 注释

3 Android obfuscation android 混淆

3.1 Using Android Studio 使用 Android Studio

3.2 Using Ant使用 Ant

4 Eclipse IDE plugin eclipse IDE 插件

# 1 Configuration file structure

Allatori configuration file is a XML file with the following structure:

<config>

<[input](http://www.allatori.com/doc.html" \l "input) basedir="input-jars" single-jar="application.jar">

<[jar](http://www.allatori.com/doc.html" \l "input-jar) in="app.jar" out="app-obf.jar"/>

<jar in="input/\*.jar" out="output/\*.jar"/>

<[dir](http://www.allatori.com/doc.html" \l "input-dir) in="in-dir" out="out-dir"/>

</input>

<[classpath](http://www.allatori.com/doc.html" \l "classpath) basedir="library-jars">

<!-- Adding library.jar to the classpath -->

<[jar](http://www.allatori.com/doc.html" \l "classpath-jar) name="library.jar"/>

<!-- Adding all jars in the lib directory to the classpath -->

<jar name="lib/\*.jar"/>

<!-- Adding all jars in the lib2 directory and its subdirectories to the classpath -->

<jar name="lib2/\*\*/\*.jar"/>

</classpath>

<[keep-names](http://www.allatori.com/doc.html" \l "keep-names)>

<[class](http://www.allatori.com/doc.html" \l "keep-names-class) [template](http://www.allatori.com/doc.html" \l "keep-names-class-template)="class SomeClass"/>

<class template="class \* instanceof java.io.Serializable"/>

<class template="class com.package.\*"/>

<class [access](http://www.allatori.com/doc.html" \l "keep-names-access)="protected+">

<field access="protected+"/>

<method access="protected+"/>

</class>

<class template="class com.company.abc.\*">

<[field](http://www.allatori.com/doc.html" \l "keep-names-field) [template](http://www.allatori.com/doc.html" \l "keep-names-field-template)="public int \*"/>

<[method](http://www.allatori.com/doc.html" \l "keep-names-method) [template](http://www.allatori.com/doc.html" \l "keep-names-method-template)="public get\*(\*)"/>

<method template="public set\*(\*)"/>

</class>

</keep-names>

<[watermark](http://www.allatori.com/doc.html" \l "watermark) key="secure-key-to-extract-watermark" value="Customer: John Smith"/>

<[expiry](http://www.allatori.com/doc.html" \l "expiry) date="2017/01/01" string="EXPIRED!"/>

<!-- Configuration properties, all properties are optional -->

<!-- General properties, we recommend to use these two properties -->

<[property](http://www.allatori.com/doc.html" \l "property) name="[log-file](http://www.allatori.com/doc.html" \l "property-log-file)" value="renaming-log.xml"/>

<property name="[random-seed](http://www.allatori.com/doc.html" \l "property-random-seed)" value="type anything here"/>

<!-- String encryption -->

<property name="[string-encryption](http://www.allatori.com/doc.html" \l "property-string-encryption)" value="enable"/>

<property name="[string-encryption-type](http://www.allatori.com/doc.html" \l "property-string-encryption-type)" value="fast"/>

<property name="[string-encryption-version](http://www.allatori.com/doc.html" \l "property-string-encryption-version)" value="v4"/>

<property name="[string-encryption-ignored-strings](http://www.allatori.com/doc.html" \l "property-string-encryption-ignored-strings)" value="patterns.txt"/>

<!-- Control flow obfuscation -->

<property name="[control-flow-obfuscation](http://www.allatori.com/doc.html" \l "property-control-flow-obfuscation)" value="enable"/>

<property name="[extensive-flow-obfuscation](http://www.allatori.com/doc.html" \l "property-extensive-flow-obfuscation)" value="normal"/>

<!-- Renaming -->

<property name="[default-package](http://www.allatori.com/doc.html" \l "property-default-package)" value="com.package"/>

<property name="[force-default-package](http://www.allatori.com/doc.html" \l "property-force-default-package)" value="enable"/>

<property name="[packages-naming](http://www.allatori.com/doc.html" \l "property-packages-naming)" value="abc"/>

<property name="[classes-naming](http://www.allatori.com/doc.html" \l "property-classes-naming)" value="compact"/>

<property name="[methods-naming](http://www.allatori.com/doc.html" \l "property-methods-naming)" value="compact"/>

<property name="[fields-naming](http://www.allatori.com/doc.html" \l "property-fields-naming)" value="compact"/>

<property name="[local-variables-naming](http://www.allatori.com/doc.html" \l "property-local-variables-naming)" value="optimize"/>

<property name="[update-resource-names](http://www.allatori.com/doc.html" \l "property-update-resource-names)" value="enable"/>

<property name="[update-resource-contents](http://www.allatori.com/doc.html" \l "property-update-resource-contents)" value="enable"/>

<!-- Other -->

<property name="[line-numbers](http://www.allatori.com/doc.html" \l "property-line-numbers)" value="obfuscate"/>

<property name="[generics](http://www.allatori.com/doc.html" \l "property-generics)" value="remove"/>

<property name="[inner-classes](http://www.allatori.com/doc.html" \l "property-inner-classes)" value="remove"/>

<property name="[member-reorder](http://www.allatori.com/doc.html" \l "property-member-reorder)" value="enable"/>

<property name="[finalize](http://www.allatori.com/doc.html" \l "property-finalize)" value="disable"/>

<property name="[version-marker](http://www.allatori.com/doc.html" \l "property-version-marker)" value="anyValidIdentifierName"/>

<property name="[synthetize-methods](http://www.allatori.com/doc.html" \l "property-synthetize-methods)" value="all"/>

<property name="[synthetize-fields](http://www.allatori.com/doc.html" \l "property-synthetize-fields)" value="all"/>

<property name="[remove-toString](http://www.allatori.com/doc.html" \l "property-remove-toString)" value="enable"/>

<property name="[remove-calls](http://www.allatori.com/doc.html" \l "property-remove-calls)" value="com.package.Logger.debug"/>

<!-- Incremental obfuscation -->

<property name="[incremental-obfuscation](http://www.allatori.com/doc.html" \l "property-incremental-obfuscation)" value="input-renaming-log.xml"/>

</config>

**Note 1.** All relative paths are resolved against configuration file location.   
**Note 2.** If you run Allatori from Ant, then you can reference properties defined in the Ant build file using standard Ant syntax: ${PropertyName}.

## 1.1 Input tag输入标记

The input tag is used to set jar (war, ear) files that should be obfuscated. It should contain at least one nested jar or dir tag to set input and output files.

输入的标记用来设置应进行模糊处理的jar（war、ear）文件。它应该包含至少一个嵌套的 jar 或 dir 标记来设置输入和输出文件。

The input tag has two optional attributes: 输入的标记有两个可选属性︰

|  |  |
| --- | --- |
| **Attribute** | **Value** |
| basedir | Optional. Relative paths to jar files will be resolved against the specified directory. By default, relative paths are resolved against configuration file location.  可选。jar 文件的相对路径将针对指定的目录进行解析。默认情况下，相对路径是相对配置文件所在目录而言。 |
| single-jar | Optional. Allatori will create an additional output jar file with all obfuscated classes.  可选的。Allatori 将创建额外的输出 jar 文件，包含了经过模糊处理的所有类。 |

The nested jar tag has two required attributes: 嵌套的 jar 标记具有两个必需的属性︰

|  |  |
| --- | --- |
| **Attribute** | **Value** |
| in | Required. The name of the jar file to obfuscate.  必填。要进行模糊处理的 jar 文件的名称。 |
| out | Required. The name of the output jar file. It can be the same as in, in such case the jar will be overwritten with its obfuscated version.  必填。输出 jar 文件的名称。它可以作为中，在这种情况下，jar 将覆盖其模糊处理的版本相同。 |

The nested dir tag has two required attributes:

|  |  |
| --- | --- |
| **Attribute** | **Value** |
| in | Required. The name of the directory with class files to obfuscate.  必填。需要进行模糊处理的类文件目录的名称。 |
| out | Required. The name of the output directory for obfuscated class files. The output directory will not be emptied before writing new files to avoid accidental removal of data in the case of typo in the config. You should empty the output directory before running Allatori (i.e. using Ant's delete task).  必填。经过混淆处理的类文件的输出目录名称。写入新文件之前，输出目录将不会被清空，以避免在配置在错字的情况下意外的数据清除。你应该在运行allatori之前清空输出目录（也就是用 Ant 的删除任务）。 |

Example:

<input basedir="input-jars" single-jar="application.jar">

<jar in="app.jar" out="app-obf.jar"/>

<jar in="input/\*.jar" out="output/\*.jar"/>

<dir in="in-dir" out="out-dir"/>

</input>

## 1.2 Classpath tag类路径标记

The classpath tag is used to set the classpath for the obfuscated application. It contains nested jar tags with the names of jar files. It is not necessary to reference all library jars needed by your application, but missing classpath elements may result in a weaker obfuscation. Allatori will warn you about all missing classes during the obfuscation process.

类路径标记用来设置经过混淆处理的应用程序的类路径。它包含嵌套的 jar标记，该标记包含 jar 文件的名称。没有必要引用应用程序所需的所有jar库文件，但缺少类路径元素可能会导致较弱的混淆。Allatori 将警告您混淆处理过程缺少的所有类。

The classpath tag has one optional attribute: 类路径标记都有一个可选属性︰

|  |  |
| --- | --- |
| **Attribute** | **Value** |
| basedir | Optional. Relative paths to jar files will be resolved against the specified directory. By default, relative paths are resolved against configuration file location.  是可选的。jar 文件的相对路径将针对指定的目录进行解析。默认情况下，相对路径是相对配置文件所在目录而言。 |

The nested jar tag has one required attribute: 嵌套的 jar 标记都有一个必需的属性︰

|  |  |
| --- | --- |
| **Attribute** | **Value** |
| name | Required. The name of jar file to add to the classpath. Wildcard syntax is allowed: "\*" matches any characters in the file name; "\*\*" recurses into subdirectories.  必填。要添加到类路径中的 jar 文件的名称。允许通配符语法:"\*"匹配任何字符的文件名;"\* \*"递归到其子目录。 |

Example:

<classpath basedir="library-jars">

<!-- Adding library.jar to the classpath将 library.jar 添加到类路径中-->

<jar name="library.jar"/>

<!-- Adding all jars in the lib directory to the classpath将 lib 目录中的所有 jar 都添加到类路径中-->

<jar name="lib/\*.jar"/>

<!-- Adding all jars in the lib2 directory and its subdirectories to the classpath -->

<!-- Lib2 目录和其子目录中的所有 jar 都添加到类路径中-->

<jar name="lib2/\*\*/\*.jar"/>

</classpath>

## 1.3 Keep-names tag保持名称标记

The keep-names tag is used to set names of classes, methods and fields that should not be renamed during the obfuscation process. If the obfuscated application is a library, then you should keep all public API. For stand-alone applications, you should keep at least the main class's name. You should also keep names of classes and methods which are used via reflection.

keep-names标记用来设置类、方法和字段在混淆处理过程中不会被重命名。如果经过模糊处理的应用程序是一个库，那么你就应该保持所有的公共 API。对于独立应用程序，你应该至少保持主类名称。您还应该保留通过反射使用的类和方法的名称。  
  
You can use [annotations](http://www.allatori.com/doc.html" \l "annotations) for more accurate control of renamed elements. Annotations override configuration file settings.

您可以使用[annotations](http://www.allatori.com/doc.html" \l "annotations)更精确的控制重命名元素。Annotations重写配置文件设置。

The keep-names tag contains any number of the following nested tags:   
    field tag to specify fields that should not be renamed;   
    method tag to specify methods that should not be renamed;   
    class tag to specify classes that should not be renamed. Can, in turn, contain nested field and method tags.

keep-names标记包含任意数量的下列嵌套标记︰

field标记来指定字段不应重命名;

method标记为指定方法不应重命名;

class标记为指定类不应重命名。可以反过来，包含字段和方法的嵌套的标签。

These nested tags set rules for matching names of classes, fields and methods. Matched names will not be renamed. All these tags can have either access or template attribute.

这些嵌套的标记设置规则用于匹配类、字段和方法的名称的匹配。匹配的名称则不会被重命名。所有这些标记都可以具有access或template的属性。

The access attribute matches elements by access levels and can have the following values:

access属性匹配元素的访问级别，可以具有下列值︰

|  |  |
| --- | --- |
| **Value** | **Description** |
| private | Matches classes, fields or methods with private access.  匹配私有访问权限的类、字段或方法 |
| private+ | Matches classes, fields or methods with private or wider access.  匹配私有或更宽访问权限的类、字段或方法。 |
| package | Matches classes, fields or methods with package access.  匹配包访问权限的类、字段或方法。 |
| package+ | Matches classes, fields or methods with package or wider access.  匹配包访问或更宽权限的类、字段或方法。 |
| protected | Matches classes, fields or methods with protected access.  匹配protected访问权限的类、字段或方法。 |
| protected+ | Matches classes, fields or methods with protected or wider access.  匹配protected访问或更宽权限的类、字段或方法。 |
| public | Matches classes, fields or methods with public access.  匹配公共访问权限的类、字段或方法。 |

The template attribute is similar to Java language syntax. It has different formats for class, field and method tags.

template属性是类似于 Java 语言的语法。它具有不同格式的类、 字段和方法的标记。

### 1.3.1 class tag类标记

The class tag is used to match classes. It has the following attributes:

类标记用于匹配的类。它具有下列属性︰

|  |  |
| --- | --- |
| **Attribute** | **Value** |
| access | Required\*. Sets matching rule. Possible values are [above](http://www.allatori.com/doc.html" \l "keep-names-access).  需要\*. 设置匹配规则。 可能的值如上。 |
| template | Required\*. Sets matching rule. Its format is described below.  需要\*. 设置匹配规则。其格式如下所述。 |
| ignore | Optional. If set to "true" or "yes", then matched classes will be renamed, but nested method and field tags will be processed as usual. It allows to keep names of some fields and methods without keeping the name of the class.  可选。如果设置为"true"或"yes"，那么匹配的类会被更改名称，但会像往常一样处理嵌套的方法和字段标记。它允许保留一些字段和方法的名称而不保留类的名称。 |
| stop | Optional. If set to "true" or "yes", then Allatori will stop applying any further rules to the matched classes.  可选。如果设置为"true"或"yes"，则 Allatori 将停止任何进一步的匹配类规则的应用。 |

 \* Either access or template attribute is required. 访问或模板的属性是必需的。

template attribute of the class tag has the following format:

template属性的class标记具有以下格式︰

[@annotation] [modifiers] (class | interface) classname [extends classname]

[implements classname]

[instanceof classname]

The '\*' symbol in class or type name matches any number of characters.

' \*' 中类或类型名称的符号匹配任意数量的字符。

Examples:

|  |  |
| --- | --- |
| **Value** | **Description** |
| class \* | Matches all classes and interfaces. 匹配所有类和接口。 |
| interface \* | Matches all interfaces. 匹配所有接口。 |
| public class \* | Matches all public classes and interfaces.  匹配所有公共类和接口。 |
| protected+ class \* | Matches all protected and public classes and interfaces.  匹配所有受保护的和公共的类和接口。 |
| class \*abc\* | Matches all classes containing "abc" in their fully qualified name.  匹配其完全限定名称包含"abc"的所有类。 |
| class com.abc.\* | Matches all classes in the com.abc package and its subpackages.  匹配的 com.abc 包和它的子包中的所有类。 |
| class \*.abc.\* | Matches all classes in all "abc" packages and their subpackages.  匹配所有的"abc"软件包和它们的子包中的所有类。 |
| class \* extends java.util.Enumeration | Matches all classes extending java.util.Enumeration.  匹配继承java.util.Enumeration的所有类。 |
| class \* extends \*.Enumeration | Matches all classes extending Enumeration.  匹配继承Enumeration的所有类。 |
| class \* instanceof java.io.Serializable | Matches all classes that are instances of java.io.Serializable.  匹配java.io.Serializable实例的所有类。 |
| class \* implements \*.MouseListener | Matches all classes implementing MouseListener.  匹配实现 MouseListener 的所有类。 |
| @java.lang.Deprecated class \* | Matches all deprecated classes. 匹配所有弃用的类。 |

### 1.3.2 field tag字段标记

The field tag is used to match fields. If field tag is nested in the class tag, then it is applied only to classes matched by the parent class tag. If parent tag is keep-names, then it is applied to all classes. The field tag has the following attributes:

field标记用来匹配字段。如果field标记嵌套在class标记中，然后它是只适用于由父类标记匹配的类。如果父标记是keep-names，则它被应用于所有的类。字段标记具有下列属性︰

|  |  |
| --- | --- |
| **Attribute** | **Value** |
| access | Required\*. Sets matching rule. Possible values are [above](http://www.allatori.com/doc.html" \l "keep-names-access).  需要\*. 设置匹配规则。 可能的值如上。 |
| template | Required\*. Sets matching rule. Its format is described below.  需要\*. 设置匹配规则。其格式如下所述。 |

\* Either access or template attribute is required. 访问或模板的属性是必需的。

template attribute of the class tag has the following format:

template属性的class标记具有以下格式︰

[@annotation] [modifiers] [type] fieldname [instanceof classname]

The '\*' symbol in field or type name matches any number of characters.

' \*' 中字段或类型名称的符号匹配任意数量的字符。

Examples:

|  |  |
| --- | --- |
| **Value** | **Description** |
| \* | Matches all fields. 匹配的所有字段。 |
| private \* | Matches all private fields. 匹配所有私有字段。 |
| private+ \* | Matches all fields. 匹配的所有字段。 |
| protected+ \* | Matches all protected and public fields.  匹配所有受保护和公共字段。 |
| static \* | Matches all static fields. 匹配所有的静态字段。 |
| public static \* | Matches all public static fields. 匹配所有公共静态字段。 |
| public int \* | Matches all public integer fields. 匹配所有公共整数字段。 |
| java.lang.String \* | Matches all String fields. 匹配所有字符串字段。 |
| java.lang.\* \* | Matches all fields with type in java.lang package.  匹配java.lang 包中类型的所有字段。 |
| abc\* | Matches all fields which names start with "abc".  匹配名称以"abc"开头的所有字段。 |
| private abc\* | Matches all private fields which names start with "abc".  匹配名称以"abc"开头的所有私有字段。 |
| \* instanceof java.io.Serializable | Matches all serializable fields. 匹配所有序列化的字段。 |
| @java.lang.Deprecated \* | Matches all deprecated fields. 匹配所有过时的字段。 |

### 1.3.3 method tag方法标记

The method tag is used to match methods. If method tag is nested in the class tag, then it is applied only to classes matched by the parent class tag. If parent tag is keep-names, then it is applied to all classes. The method tag has the following attributes:

method标记用于匹配的方法。如果method标记嵌套class标记中，那么它是只适用于由父类标记匹配的类。如果父标记是keep-names，则它被应用于所有的类。方法标记具有下列属性︰

|  |  |
| --- | --- |
| **Attribute** | **Value** |
| access | Required\*. Sets matching rule. Possible values are [above](http://www.allatori.com/doc.html" \l "keep-names-access).  需要\*. 设置匹配规则。 可能的值如上。 |
| template | Required\*. Sets matching rule. Its format is described below.  需要\*. 设置匹配规则。其格式如下所述。 |
| parameters | Optional. If set to "keep", then names of method's parameters will not be changed. Useful for public API methods.  可选。如果设置为" keep "，则方法的参数的名称将不会更改。对于公共 API 方法很有用。 |

\* Either access or template attribute is required. 访问或模板的属性是必需的。

template attribute of the class tag has the following format:

template属性的class标记具有以下格式︰

[@annotation] [modifiers] [type] methodname(arguments)

The '\*' symbol in method or type name matches any number of characters. The '\*' symbol in arguments matches any single argument. Use "\*\*" to match any number of arguments.

' \*' 中method或类型名称的符号匹配任意数量的字符。参数中，' \*'符号匹配任何单个参数。使用"\* \*"，匹配任意数量的参数。

Examples:

|  |  |
| --- | --- |
| **Value** | **Description** |
| \*(\*\*) | Matches all methods. 匹配所有方法。 |
| private \*(\*\*) | Matches all private methods. 匹配所有的私有方法。 |
| private+ \*(\*\*) | Matches all methods. 匹配所有方法。 |
| protected+ \*(\*\*) | Matches all protected and public methods.  匹配所有受保护和公共方法。 |
| private+ \*(\*) | Matches all methods with exactly one argument.  匹配具有一个参数的所有方法。 |
| private+ \*(\*,\*) | Matches all methods with exactly two arguments.  匹配具有正好两个参数的所有方法。 |
| private+ \*(java.lang.String) | Matches all methods with exactly one argument of String type.  匹配具有一个参数的字符串类型的所有方法。 |
| private+ \*(java.lang.String,\*\*) | Matches all methods with String as the first argument.  匹配第一个参数为字符串的所有方法。 |
| private+ \*(java.lang.\*) | Matches all methods with exactly one argument which type is in java.lang package.  匹配具有恰好一个java.lang 包中类型的所有方法。 |
| public get\*(\*\*) | Matches all public methods which names start with "get".  匹配以名称" get "开始的所有公共方法。 |
| public \*abc\*(\*\*) | Matches all public methods which names contain "abc".  匹配以名称中包含"abc"的所有公共方法。 |
| private+ int \*(\*\*) | Matches all methods with int return type.  匹配返回类型为int的所有方法。 |
| @java.lang.Deprecated \*(\*\*) | Matches all deprecated methods. 匹配所有已过时的方法。 |

Example:

<keep-names>

<!-- Stops applying further rules to classes in the com.company.abc package,

therefore all classes, methods and fields in this package will be renamed -->

<!--停止进一步将规则应用于 com.company.abc 包中的类

因此所有的类、 方法和字段在此程序包将被重命名-->

<class template="class com.company.abc.\*" stop="true"/>

<!-- Further rules instruct Allatori not to rename matched elements -->

<!—更多的规则命令Allatori不要重命名匹配的元素 -->

<!-- Matches classes with the name "Main" in any package匹配任何包中名称为“Main”的类-->

<class template="class \*.Main"/>

<!-- Matches classes with the name ending with "Bean" 匹配以“Bean”结尾的名称的类 -->

<class template="class \*Bean">

<!-- Matches all fields匹配所有字段-->

<field access="private+"/>

<!-- Matches public integer fields匹配公共整数字段-->

<field template="public int \*"/>

<!-- Matches all static fields匹配所有静态字段-->

<field template="static \*"/>

<!-- Matches protected and public String fields匹配受保护和公共字符串字段-->

<field template="protected+ java.lang.String \*"/>

<!-- Matches all methods匹配所有方法-->

<method template="private+ \*(\*\*)"/>

<!-- Matches all getter methods匹配所有getter方法-->

<method template="private+ get\*(\*\*)"/>

<!-- Matches all methods with String argument,

parameter names of these methods will not be changed -->

<!--匹配使用String参数所有方法，这些方法的参数名称不会更改-->

<method template="private+ \*(java.lang.String)" parameters="keep"/>

</class>

<!-- Matches serialization members匹配序列化成员-->

<class template="class \* instanceof java.io.Serializable">

<field template="static final long serialVersionUID"/>

<method template="void writeObject(java.io.ObjectOutputStream)"/>

<method template="void readObject(java.io.ObjectInputStream)"/>

<method template="java.lang.Object writeReplace()"/>

<method template="java.lang.Object readResolve()"/>

</class>

<!-- Matches applets匹配applet -->

<class template="class \* instanceof java.applet.Applet"/>

<!-- Matches servlets匹配servlets-->

<class template="class \* instanceof javax.servlet.Servlet"/>

<!-- Matches midlets 匹配midlets -->

<class template="class \* instanceof javax.microedition.midlet.MIDlet"/>

</keep-names>

## 1.4 Watermark tag水印标记

The watermark tag is used to set key and value for [watermarking](http://www.allatori.com/features/watermarking.html). The tag has two attributes:

水印标记用于设置[watermarking](http://www.allatori.com/features/watermarking.html)的键和值。 该标签有两个属性：

|  |  |
| --- | --- |
| **Attribute** | **Value** |
| key | Required. The key used to embed a watermark into the application using steganography techniques.  必填。用于使用隐写（信息隐藏）技术将水印嵌入应用程序的密钥。 |
| value | Required for adding watermark. Any string that will be embedded into the application jars. It can be copyright, customer name, company name or any other information that uniquely identifies the build. A watermark can be used to identify owners of the software or track the origin of a pirated copy.  需要用来添加水印。 为将被嵌入到应用程序jar中的任何字符串。 它可以是版权，客户名称，公司名称或任何其他唯一标识构建的信息。 水印可用于识别软件的所有者或跟踪盗版拷贝的来源。 |

Example:

<watermark key="secure-key-to-extract-watermark" value="Customer: John Smith"/>

Watermark can also be added without obfuscation to already obfuscated or non-obfuscated jars. Complete examples of adding and extracting watermarks can be found in the tutorial that comes with Allatori distribution.

水印也可以在不混淆的情况下添加到已经混淆或未混淆的jar文件中。 添加和提取水印的完整示例可以在Allatori发行版的教程中找到。

## 1.5 Expiry tag到期标签

The expiry tag is used to set expiry date to your application. Expiry date checks are inserted into many methods, not just main method, therefore cannot be easily removed. This feature can be used to obfuscate libraries that do not even have main method. The tag has two required attributes:

expiry标签用于设置应用程序的到期日期。 到期日期检查插入到许多方法中，而不仅仅是主要方法，因此不能被轻易删除。此功能可以用于混淆甚至没有主方法的库。标记具有两个必需属性：

|  |  |
| --- | --- |
| **Attribute** | **Value** |
| date | Required. Expiry date in the yyyy/mm/dd format.  必填。到期日期以yyyy / mm / dd格式。 |
| string | Required. Any string message for the exception thrown if the application is run after the specified expiry date.  必填。如果应用程序在指定的到期日期之后运行，则抛出的异常的任何字符串消息。 |

Example:

<expiry date="2017/01/01" string="EXPIRED!"/>

Complete example of using expiry date can be found in the tutorial that comes with Allatori distribution.

使用过期日期的完整示例可以在Allatori发行版的教程中找到。

## 1.6 Property tag属性标记

The property tag is used to set different obfuscation properties. The tag has two required attributes - **name** and **value**:

属性标记用来设置不同的混淆处理属性。标签有两个必需的属性的名称和值︰

<property name="property-name" value="property-value"/>

### 1.6.1 General properties一般属性

#### 1.6.1.1 log-file日志文件

|  |  |
| --- | --- |
| Value | Description |
| filename | Allatori will write obfuscation log to the specified file. If the property is not set, then log file is not created.  Allatori 将混淆处理日志写入指定的文件。如果不设置该属性，则不会创建日志文件。  Relative paths are resolved against configuration file location. 相对路径是相针对于配置文件的路径位置。 |

The log file is used to restore original stack trace from the obfuscated one. It keeps obfuscated-to-original mapping of names and line numbers.

日志文件用于从混淆后的文件还原原始堆栈跟踪。它保留了混淆文件和原始文件之间的关于名称和行号的映射。  
Examples:

<property name="log-file" value="log.xml"/>

<property name="log-file" value="logs/file.xml"/>

#### 1.6.1.2 random-seed随机种子

|  |  |
| --- | --- |
| **Value** | **Description** |
| any string | String to initialize random number generator. 要初始化随机数生成器的字符串。 |

The default value is current time in milliseconds. 默认值为当前时间 （毫秒）。  
  
By default, if you run Allatori twice on the same input jars, it will produce different output jars. Renamed classes, fields and methods will have different names, fields and methods will be reordered differently, etc.

默认情况下，如果您运行 Allatori 两次相同的输入 jar，它会产生不同的输出 jar。重命名的类、字段和方法会有不同的名字，字段和方法将会以不同的方式排序，等等。

This is done to make analyzing two versions of the obfuscated application extremely difficult. If you need the same output jars over several consecutive runs of Allatori, then you need to set the random seed. We recommend changing the random seed for different public releases of your application.

这样做的目的是为了使分析两个版本的混淆处理后的应用程序变得极其困难。如果你需要几次连续运行的 Allatori，生成相同的jar文件，那么您需要设置随机种子。我们建议对于不同的公共版本，改变您的应用程序的随机种子。  
Example:

<property name="random-seed" value="any text here"/>

### 1.6.2 String encryption properties字符串加密属性

#### 1.6.2.1 string-encryption字符串加密

|  |  |
| --- | --- |
| **Value** | **Description** |
| enable | *(default)* All string literals that can be safely changed with the encrypted value will be encrypted. Allatori will add a method to decrypt strings on run-time.  （默认值）可以安全地更改与加密后的值的所有字符串将被都加密。Allatori 将在运行阶段添加一个方法来解密字符串。 |
| disable | String encryption is disabled. 字符串加密被禁用。 |
| maximum | All string literals will be encrypted. See limitations below.  所有字符串将被都加密。请参见下面的限制。 |
| maximum-with-warnings | All string literals will be encrypted. Every string comparison using == operator will produce a warning, and you'll be able to replace these comparisons with equals() calls.  所有字符串将被都加密。每个字符串比较使用 == 运算符将产生一个警告，你就能够使用 equals（）调用替换这些比较。 |

Sometimes, strings are compared this way: 有时，字符串以下面的方式比较︰

String myString = "Hello";

...

public boolean test() {

return myString == "Hello";

}

Although it is a bad practice to compare strings using == operator instead of equals method, the method in the example above will return true, because JVM caches String objects to reuse them within the same class. However, after string encryption the method will look something like this:

虽然比较字符串使用 == 运算符而不使用equals 方法是不好的做法，上面示例中的方法将返回 true，因为 JVM 会缓存字符串对象以便重用它们。然而，字符串加密后方法会看起来像这样︰

public boolean test() {

return myString == new String("Hello");

// "Hello" string isn't encrypted to make this example more clear

//"你好"字符串不被加密，以使此示例更清楚

}

This version of the method will return false, because compared objects are different.

此版本的方法将返回 false，因为比较的对象不同。

If you set string-encryption property to enable, then Allatori will not encrypt strings which are compared using == operator, and your application will work correctly.

如果您设置string-encryption属性为enable，那么Allatori不会加密使用==运算符进行比较的字符串，您的应用程序将正常工作。

If you are always using equals method to compare strings, then set string-encryption property to maximum.

如果你总是使用 equals 方法来比较字符串，然后将string-encryption属性设置为maximum。

Example:

<property name="string-encryption" value="enable"/>

String encryption can be enabled/disabled in the specified classes using [annotations](http://www.allatori.com/doc.html" \l "annotations) or apply2class attribute. The apply2class attribute has the same format as [template](http://www.allatori.com/doc.html" \l "keep-names-class-template) attribute of the class tag. Here is an example:

字符串加密可在指定的使用[annotations](http://www.allatori.com/doc.html" \l "annotations)或 apply2class 属性的类中启用/禁用。Apply2class 属性和class标记下的[template](http://www.allatori.com/doc.html" \l "keep-names-class-template)属性具有相同的格式。下面是一个示例︰

<!-- Disabling string encryption for classes in com.abc packagecom.abc 包中的类的字符串加密禁用-->

<property name="string-encryption" value="disable" apply2class="class com.abc.\*"/>

<!-- Enabling string encryption for all other classes所有其他类的字符串加密启用-->

<property name="string-encryption" value="enable"/>

#### 1.6.2.2 string-encryption-type字符串加密类型

|  |  |
| --- | --- |
| **Value** | **Description** |
| fast | *(default)* Allatori will use very fast string encryption algorithm.  （默认值）Allatori 将使用非常快速的字符串加密算法。 |
| strong | Allatori will use strong and tricky string encryption algorithm. It is, however, slower.  Allatori 将使用强和棘手的字符串加密算法。然而，它是速度较慢。 |

[String encryption](http://www.allatori.com/doc.html" \l "property-string-encryption) should be enabled for this property to take effect.

string-encryption应该启用，此属性才会生效。

Example:

<property name="string-encryption-type" value="strong"/>

String encryption type can be applied to the specified classes using [annotations](http://www.allatori.com/doc.html" \l "annotations) or apply2class attribute. The apply2class attribute has the same format as [template](http://www.allatori.com/doc.html" \l "keep-names-class-template) attribute of the class tag. Here is an example:

可以将字符串加密类型应用于使用[annotations](http://www.allatori.com/doc.html" \l "annotations)或 apply2class 属性指定的类。Apply2class 属性和class标记下的[template](http://www.allatori.com/doc.html" \l "keep-names-class-template)属性具有相同的格式。下面是一个示例︰

<!-- Setting strong string encryption type for classes in com.abc package-->

<!--在 com.abc 包中设置类强字符串加密类型-->

<property name="string-encryption-type" value="strong" apply2class="class com.abc.\*"/>

<!-- Setting fast string encryption type for all other classes -->

< ！ -- 为所有其他类设置快速的字符串加密类型-->

<property name="string-encryption-type" value="fast"/>

#### 1.6.2.3 string-encryption-version字符串加密版本

|  |  |
| --- | --- |
| **Value** | **Description** |
| v4 | *(default)* New string encryption algorithm will be used.  （默认值）将使用新的字符串加密算法。 |
| v3 | Allatori will use string encryption algorithm from the 3.X version.  Allatori 将使用字符串加密算法3.X 版本。 |

[String encryption](http://www.allatori.com/doc.html" \l "property-string-encryption) should be enabled for this property to take effect.

string-encryption应该启用，此属性才会生效。

It doesn't mean that v3 algorithm is outdated. We'll make changes to both v3 and v4 algorithms from time to time to keep them fresh. This property was introduced as the key ideas of v3 and v4 algorithms are very different.

这并不意味着，v3 算法是过时。我们会经常更新v3 和 v4 的算法以保持算法活力。此属性介绍了作为关键构思的 v3 和 v4 算法有很大不同。

Example:

<property name="string-encryption-version" value="v3"/>

#### 1.6.2.4 string-encryption-ignored-strings字符串加密忽略字符串

|  |  |
| --- | --- |
| **Value** | **Description** |
| filename | Text file containing string literals which should be excluded from obfuscation. These strings will not be encrypted.  文本文件，其中包含字符串文本，应从混淆处理中排除。这些字符串不会被加密。 |

The specified text file contains string templates. Each line is a new template, '\*' matches any number of any characters. If template line starts with "*regex:*", then it is processed as a regular expression:

指定的文本文件中包含字符串的模板。每一行都是一个新的模板，' \*' 匹配任何数量的任何字符。如果模板行开头为则" *regex*:"，那么该行作为正则表达式进行处理︰

Copyright\*

All Rights Reserved\*

\*CompanyName\*

regex:\d+

Example:

<property name="string-encryption-ignored-strings" value="patterns.txt"/>

### 1.6.3 Control flow obfuscation properties控制流混淆处理属性

#### 1.6.3.1 control-flow-obfuscation控制流混淆处理

|  |  |
| --- | --- |
| **Value** | **Description** |
| enable | *(default)* Allatori will alter the code of the methods. It will not change the application behaviour at run-time, but will make the decompilation process much harder. Generally, control flow obfuscation also makes application smaller and faster.  （默认值）Allatori 将改变方法的代码。它不会改变在运行过程中的应用程序行为，但是会使反编译过程困难得多。一般来说，控制流混淆处理也使应用程序更小更快。 |
| disable | Control flow obfuscation is disabled. 控制流混淆处理处于禁用状态。 |

Example:

<property name="control-flow-obfuscation" value="enable"/>

Control flow obfuscation can be enabled/disabled in the specified classes using [annotations](http://www.allatori.com/doc.html" \l "annotations) or apply2class attribute. The apply2class attribute has the same format as [template](http://www.allatori.com/doc.html" \l "keep-names-class-template) attribute of the class tag. Here is an example:

控制流混淆处理可在指定的使用[annotations](http://www.allatori.com/doc.html" \l "annotations)或 apply2class 属性的类中启用/禁用。Apply2class 属性和class标记下的[template](http://www.allatori.com/doc.html" \l "keep-names-class-template)属性具有相同的格式。下面是一个示例︰

<!-- Disabling control flow obfuscation for classes in com.abc package -->

<!-- 禁用com.abc 包中的类的控制流模糊处理-->

<property name="control-flow-obfuscation" value="disable" apply2class="class com.abc.\*"/>

<!-- Enabling control flow obfuscation for all other classes -->

<!--对于所有其他类启用控制流模糊处理-->

<property name="control-flow-obfuscation" value="enable"/>

#### 1.6.3.2 extensive-flow-obfuscation广泛流混淆处理

|  |  |
| --- | --- |
| **Value** | **Description** |
| normal | *(default)* Allatori will use control flow obfuscation techniques, which make the obfuscated application a bit bigger and slower. However, Allatori will minimize the number of such code transformations.  （默认值）Allatori 将使用控制流模糊处理技术，使经过模糊处理的应用程序，有点大，速度较慢。然而，Allatori 将尽量减少这类代码转换的数量。 |
| disable | Extensive control flow obfuscation is disabled.  广泛的控制流模糊处理处于禁用状态。 |
| maximum | Allatori will fully use control flow obfuscation techniques, which make the obfuscated application a bit bigger and slower.  Allatori 将完全使用控制流模糊处理技术，使经过混淆处理的应用程序文件有点大，运行速度较慢。 |

[Control flow obfuscation](http://www.allatori.com/doc.html" \l "property-control-flow-obfuscation) should be enabled for this property to take effect.

control-flow-obfuscation应启用，此属性才会生效。  
  
Example:

<property name="extensive-flow-obfuscation" value="maximum"/>

Extensive control flow obfuscation can be applied to the specified classes using [annotations](http://www.allatori.com/doc.html" \l "annotations) or apply2class attribute. The apply2class attribute has the same format as [template](http://www.allatori.com/doc.html" \l "keep-names-class-template) attribute of the class tag. Here is an example:

广泛流混淆处理可在指定的使用[annotations](http://www.allatori.com/doc.html" \l "annotations)或 apply2class 属性的类中启用/禁用。Apply2class 属性和class标记下的[template](http://www.allatori.com/doc.html" \l "keep-names-class-template)属性具有相同的格式。下面是一个示例︰

<!-- Using "maximum" value for classes in com.abc package -->

<!--为 com.abc 包中的类使用" maximum "值-->

<property name="extensive-flow-obfuscation" value="maximum" apply2class="class com.abc.\*"/>

<!-- Using "normal" value for all other classes -->

<!--对于所有其他类使用" normal "值 -->

<property name="extensive-flow-obfuscation" value="normal"/>

### 1.6.4 Renaming properties重命名属性

#### 1.6.4.1 default-package默认包

|  |  |
| --- | --- |
| **Value**值 | **Description**描述 |
| package name包名称 | Full package name, existing or new. 完整软件包名称，现有的包或新生成的包。 |

If all classes in some package are renamed, then Allatori will move them to the default package. To move absolutely all renamed classes to the default package, you should enable the force-default-package property. The usage of "" as the default package will reduce the size of the resulting jar.

如果重命名一些包中的所有类，然后 Allatori 会将它们移到默认包。将绝对所有重命名的类移动到默认包，您应该启用“强制默认的包属性”。使用""作为默认包将减少由此产生的 jar 的大小。

Examples: 例子︰

<property name="default-package" value=""/>

<property name="default-package" value="com.company.product"/>

#### 1.6.4.2 force-default-package 强制-默认-包

|  |  |
| --- | --- |
| Value值 | Description描述 |
| disable | (default) Only classes from packages where all classes are renamed will be moved to the default package. （默认） 只有所有类都重命名的包里的类会被移到默认的包。 |
| enable | Absolutely all renamed classes will be moved to the default package.  绝对所有重命名的类将会移到缺省包 |

Default package should be set for this property to take effect.

“default-package默认包”应设置之后，此属性才会生效。

Example:

<property name="force-default-package" value="enable"/>

#### 1.6.4.3 packages-naming 包命名

|  |  |
| --- | --- |
| **Value** | **Description** |
| abc | (default) Packages will be renamed to 'a', 'b', 'c', 'd', ..., 'aa', 'ab', etc. Names will have lower-case letters only.  （默认）包将被重命名为 'a'，'b'，'c'，有 '，......，aa、 ab 等。名称会仅有小写字母。 |
| 123 | Packages will be renamed to '1', '2', '3', ..., '00', '01', etc.  包将被重命名为 ' 1'，' 2'，'3'，......，' 00'，' 01'，等。 |

Example:

<property name="packages-naming" value="abc"/>

#### 1.6.4.4 classes-naming 类命名

|  |  |
| --- | --- |
| **Value** | **Description** |
| compact | *(default)* Allatori will use single-character names as much as possible, thus making the resulting jar smaller. Classes may have mixed-case names differing only in case (a.class and A.class). Jar files allow mixed-case file names, while Windows file system does not, so it will be tricky to unzip some classes (a.class will overwrite A.class on Windows when unzipped). The jar file with mixed-case file names will work fine on all platforms including Windows.  （默认）Allatori 将尽可能多地使用单字符名称，从而使生成 jar 较小。类可能有差别仅在于大小 （a.class和A.class） 的大小写混合的名字。Jar 文件允许混合大小写的文件名，而 Windows 文件系统没有，所以在解压一些类时才会棘手（Windows 解压缩时a.class会覆盖A.class）。类文件名称大小写混合的 jar 文件将在包括 Windows在内的所有平台上运行良好。 |
| iii | All names will have the same length and differ in case only - iiii, iiiI, iiIi, etc. The resulting jar file will be bigger comparing to other renaming options.  所有名称将具有相同的长度，只是大小写不同，例如-iiii、 iiiI、 iiIi 等。由此产生的 jar 文件要比其他重命名选项产生的jar文件大。 |
| abc | Classes will be renamed to 'a', 'b', 'c', 'd', ..., 'aa', 'ab', etc. Names will have lower-case letters only.  类将重命名为 'a'，'b'，'c'，’d '，......，aa、 ab，等等。名称中仅有小写字母。 |
| 123 | Classes will be renamed to '1', '2', '3', ..., '00', '01', etc.  类将重命名为 ' 1'，' 2'，'3'，......，' 00'，' 01'，等。 |
| windows | Allatori will use names prohibited on Windows ('con', 'prn', 'aux', 'nul', etc.) as class names. It is ok to have con.class in the jar, but this class cannot be unzipped on Windows. Classes may also have mixed-case names differing only in case. The jar file with such file names will work fine on all platforms including Windows. This option makes the resulting jar file bigger comparing to compact or abc naming.  Allatori 将使用Windows系统禁止的 （如'con'、 'prn'、 aux、 'nul' 等） 的名称作为类名。它运行类con.class 存在于jar文件里面，但此类却无法在 Windows系统上解压缩。类也可能在不同的情况下有大小写混合名称。包含此类文件的Jar 文件名称将在所有平台上包括 Windows系统运行良好。相比较compact或 abc naming而言，此选项使生成的 jar 文件更大。 |
| custom(filename.txt) | Names will be constructed using the provided text file. Each line of the file represents a single name element. If the file has two lines '0' and '1', then the generated names would be '0', '1', '00', '01', '10', '11', '000', etc.  将根据提供的文本文件来构造名称。文件的每一行表示一个单一的名称元素。如果该文件有两行 '0' 和 '1'，那么，生成的名称就是 ' 0'，' 1'，' 00'，'01' ，’10'，’11 '，' 000'，等。 |
| unique | All renamed classes will have unique names. There will be no matching class names in different packages. Can be combined with other classes naming options.  重命名的所有类将都具有唯一的名称。在不同的包中不会有匹配的类名称。可以结合其他类命名选项。 |
| keep-$-sign | Renamed classes will retain Java inner class naming notation, i.e. classes Foo and Foo$Bar will be renamed to a and a$b. By default, Allatori will rename Foo and Foo$Bar to a and b. Can be combined with other classes naming options.  重命名的类将保留 Java 内部类命名符号，亦即类 Foo 和 Foo$Bar 将被重命名为a 和a$b。默认情况下，Allatori 将重命名 Foo和Foo$Bar为a和 b。可以结合其他类命名选项。 |

Example:

<property name="classes-naming" value="abc"/>

#### 1.6.4.5 methods-naming 方法命名

|  |  |
| --- | --- |
| **Value** | **Description** |
| compact | *(default)* Allatori will use single-character names as much as possible, thus making the resulting jar smaller.  （默认）Allatori 将使用单字符名称尽可能多地，从而使生成 jar 较小。 |
| iii | All names will have the same length and differ in case only - iiii, iiiI, iiIi, etc. The resulting jar file will be bigger comparing to other renaming options.  所有方法名称将具有相同的长度，只是大小写不同，例如-iiii、 iiiI、 iiIi 等。由此产生的 jar 文件要比其他重命名选项产生的jar文件大。 |
| abc | Methods will be renamed to 'a', 'b', 'c', 'd', ..., 'aa', 'ab', etc.  方法将会被重命名为 'a'，'b'，'c'，有 '，......，aa、 ab，等等。 |
| 123 | Methods will be renamed to '1', '2', '3', ..., '00', '01', etc.  方法将会被重命名为 ' 1'，' 2'，'3'，......，' 00'，' 01'，等。 |
| keywords | Allatori will use Java reserved keywords ('if', 'for', 'int', etc.) as method names. Such naming is legal in class file format, but can confuse many decompilers. However, this makes the resulting jar file bigger comparing to compact naming.  Allatori 将使用 Java 保留关键字 （'if'，'for'，'int' 等）作为方法名称。这种命名在类文件格式上是合法的，却可以迷惑很多编译器。然而，这使得生成的 jar 文件相比较compact naming生成的jar文件而言要大。 |
| custom(filename.txt) | Names will be constructed using the provided text file. Each line of the file represents a single name element. If the file has two lines '0' and '1', then the generated names would be '0', '1', '00', '01', '10', '11', '000', etc.  将根据提供的文本文件来构造名称。文件的每一行表示一个单一的名称元素。如果该文件有行 '0' 和 '1'，那么生成的名称就是 ' 0'，' 1'，' 00'，'01' ，’10年'，’11'，' 000'，等。 |
| unique | Shortcut for [unique-renaming](http://www.allatori.com/doc.html" \l "property-unique-renaming) property. Can be combined with other methods naming options. If any two methods have the same name and signature, then these methods will be renamed to the same new name. If any two methods have different name/signature, then these methods will have different names after renaming. It ensures consistency during subsequent incremental obfuscation runs.  [unique-renaming](http://www.allatori.com/doc.html" \l "property-unique-renaming)属性的快捷方式。可以结合其他方法命名选项。如果任何两种方法具有相同名称和签名，然后这些方法会被重命名为相同的新名称。如果任何两种方法具有不同的名称或则签名，然后这些方法之后会有不同的名称重命名。在随后的增量模糊处理（[Incremental obfuscation properties](http://www.allatori.com/doc.html" \l "properties-incremental-obfuscation)）运行期间，它可以确保一致性。 |

Example:

<property name="methods-naming" value="keywords"/>

#### 1.6.4.6 fields-naming 字段命名

|  |  |
| --- | --- |
| **Value** | **Description** |
| compact | *(default)* Allatori will use single-character names as much as possible, thus making the resulting jar smaller.  （默认）Allatori 将使用单字符名称尽可能多地，从而使生成 jar 较小。 |
| iii | All names will have the same length and differ in case only - iiii, iiiI, iiIi, etc. The resulting jar file will be bigger comparing to other renaming options.  所有方法名称将具有相同的长度，只是大小写不同，例如-iiii、 iiiI、 iiIi 等。由此产生的 jar 文件要比其他重命名选项产生的jar文件大。 |
| abc | Fields will be renamed to 'a', 'b', 'c', 'd', ..., 'aa', 'ab', etc.  方法将会被重命名为 'a'，'b'，'c'，有 '，......，aa、 ab，等等。 |
| 123 | Fields will be renamed to '1', '2', '3', ..., '00', '01', etc.  字段将会被重命名为 ' 1'，' 2'，'3'，......，' 00'，' 01'，等。 |
| keywords | Allatori will use Java reserved keywords ('if', 'for', 'int', etc.) as field names. Such naming is legal in class file format, but can confuse many decompilers. However, this makes the resulting jar file bigger comparing to compact naming.  Allatori 将使用 Java 保留关键字 （'if'，'for'，'int' 等）作为方法名称。这种命名在类文件格式上是合法的，却可以迷惑很多编译器。然而，这使得生成的 jar 文件相比较compact naming生成的jar文件而言要大。 |
| custom(filename.txt) | Names will be constructed using the provided text file. Each line of the file represents a single name element. If the file has two lines '0' and '1', then the generated names would be '0', '1', '00', '01', '10', '11', '000', etc.  将根据提供的文本文件来构造名称。文件的每一行表示一个单一的名称元素。如果该文件有行 '0' 和 '1'，那么生成的名称就是 ' 0'，' 1'，' 00'，'01' ，’10年'，’11'，' 000'，等。 |

Example:

<property name="fields-naming" value="keywords"/>

#### 1.6.4.7 classes-naming-prefix 类命名前缀

|  |  |
| --- | --- |
| Value | Description |
| any string | The specified string will be used as a name prefix for all renamed classes.  指定的字符串将作为所有重命名类的名称前缀。 |

Example:

<property name="classes-naming-prefix" value="c\_"/>

Possible usage is specifying MainClass$ as a prefix: 可能的用法就是：指定MainClass$为前缀

<property name="classes-naming-prefix" value="MainClass$"/>

Some decompilers will see renamed classes as inner classes of MainClass.

一些编译器会把重命名的类作为 MainClass 的内部类。

#### 1.6.4.8 methods-naming-prefix 方法命名前缀

|  |  |
| --- | --- |
| Value | Description |
| any string | The specified string will be used as a name prefix for all renamed methods.  指定的字符串将作为所有重命名方法的名称前缀。 |

Example:

<property name="methods-naming-prefix" value="m\_"/>

#### 1.6.4.9 fields-naming-prefix 字段命名前缀

|  |  |
| --- | --- |
| Value | Description |
| any string | The specified string will be used as a name prefix for all renamed fields.  指定的字符串将作为所有重命名字段的名称前缀。 |

Example:

<property name="fields-naming-prefix" value="f\_"/>

#### 1.6.4.10 local-variables-naming局部变量命名

|  |  |
| --- | --- |
| Value | Description |
| optimize | *(default)* Allatori performs optimizations to reduce the total number of local variables in a method. The remaining local variables will have the same name (*single-name* renaming option). This is the default and recommended option.  （默认）Allatori 执行优化，从而降低方法中的局部变量总数。其余的本地变量将具有相同的名称 （*single-name*重命名选项）。这是默认设置和推荐选项。 |
| single-name | Almost all local variables will have the same name. It is allowed by the Java virtual machine but can confuse a lot of decompilers.  几乎所有的本地变量将具有相同的名称。这是Java 虚拟机所允许的，但可以迷惑很多反编译器。 |
| abc | Local variables will be renamed to unique names 'a', 'b', 'c', 'd', etc.  本地变量将被重命名为唯一名称 'a'，'b'，'c'，’d'，等等。 |
| remove | The original local variable names will be removed. It can reduce the size of the resulting jar.  原始的本地变量名称将被删除。它可以减少由此产生的 jar 的大小。 |
| keep-parameters | Parameter names will be kept unchanged, all other local variables will be renamed. It is useful for methods that are part of the public API. There is also an option to keep parameters only in the specified methods using [method](http://www.allatori.com/doc.html" \l "keep-names-method) tag in the keep-names section.  参数名称将保持不变，所有其他本地变量将被重命名。这对公共API函数来说是非常有用的。还有一个可以保持参数的选项，就是在keep-names部分中使用[method](http://www.allatori.com/doc.html" \l "keep-names-method)标签为特定方法保持参数。 |
| keep | All local variable names will be kept unchanged. This option is not recommended.  所有局部变量的名称将会维持不变。不建议使用此选项。 |

Example:

<property name="local-variables-naming" value="single-name"/>

The default single name for *single-name* and *optimize* renaming options is 'a'. You can change it using one of the following lines:

默认*single-name*和*optimize* 命名选项中的单个命名是 'a'。您可以更改它使用以下行之一︰

<property name="local-variables-naming" value="optimize:ANY\_OTHER\_NAME"/>

<property name="local-variables-naming" value="optimize:int"/>

<property name="local-variables-naming" value="single-name:4"/>

#### 1.6.4.11 skip-renaming跳过重命名

|  |  |
| --- | --- |
| Value | Description |
| disable | *(default)* Allatori performs renaming of classes, methods and fields according to [keep-names](http://www.allatori.com/doc.html" \l "keep-names) rules.  （默认）Allatori 执行重命名的类、方法和字段根据[keep-names](http://www.allatori.com/doc.html" \l "keep-names)保持名称规则。 |
| enable | All classes, methods and fields will not be renamed. Local variables naming is controlled separately by [local-variables-naming](http://www.allatori.com/doc.html" \l "property-local-variables-naming) property. String encryption, flow obfuscation, etc. will be applied normally according to the settings in the configuration file.  所有的类、 方法和字段将不被重命名。局部变量命名由[local-variables-naming](http://www.allatori.com/doc.html" \l "property-local-variables-naming)属性分别控制。根据配置文件中的设置，字符串加密、 流模糊处理等可以正常应用。 |

Example:

<property name="skip-renaming" value="enable"/>

#### 1.6.4.12 update-resource-names更新-资源-名称

|  |  |
| --- | --- |
| Value | Description |
| disable | *(default)* Resource file names will not be changed. （默认）资源文件的名称不会改变。 |
| enable | Resource files will be renamed to reflect changes in class names. If a resource file name is based on a class name, and that class is renamed, then the resource file will also be renamed.  资源文件将被重命名，以反在映类名称上的更改。如果资源文件的名称基于类名，并且该类进行了重命名，资源文件的名称也会被重命名。 |

Example:

<property name="update-resource-names" value="enable"/>

#### 1.6.4.13 update-resource-contents更新资源的内容

|  |  |
| --- | --- |
| Value | Description |
| disable | *(default)* Resource file contents will not be changed.  （默认）资源文件的内容不会改变。 |
| enable | Resource contents will be updated to reflect changes in class names.  资源内容将更新以反映在类名称的更改。 |

Example:

<property name="update-resource-contents" value="enable"/>

The property can be applied to the specified files using apply2file attribute:

该属性可以应用于通过apply2file 属性指定的文件︰

<property name="update-resource-contents" value="enable" apply2file="\*.xml"/>

### 1.6.5 Other properties其他属性

#### 1.6.5.1 line-numbers行号

|  |  |
| --- | --- |
| Value | Description |
| obfuscate | (default) Debug information is obfuscated and cannot be used without further transformation. Allatori has a special utility, which allows reconstructing the original stack trace with the help of an obfuscated one.  （默认）调试信息被混淆和不能被使用没有进一步的转变。Allatori 有一个特别的实用工具，这使得通过混淆后的文件可以重建原来的堆栈跟踪。  Reported stack trace will look like this:堆栈跟踪报告看起来像这：  java.lang.NullPointerException  at com.company.c.a(m:61)  at com.company.b.b(w:94)  at com.company.b.a(w:83)  at com.company.a.a(n:75)  After transforming it with Allatori Stack Trace Utility, stack trace will look like this:  经过Allatori堆栈跟踪实用工具的转换后, 堆栈跟踪报告将如下所示︰  java.lang.NullPointerException  at com.company.Util.createTestException(Util.java:38)  at com.company.TraceTest.testNullObject(TraceTest.java:53)  at com.company.TraceTest.allTraceTests(TraceTest.java:14)  at com.company.Main.runTest(Main.java:27)  Using this option makes the resulting jar smaller. 使用此选项使生成 jar 更小。 |
| remove | This option can be used, when the size of your application matters really much.  当您应用程序的大小很重要时，可以使用此选项。  Reported stack trace will look like this: 报告的堆栈跟踪将看起来像这：  java.lang.NullPointerException  at com.company.c.a(Unknown Source)  at com.company.b.b(Unknown Source)  at com.company.b.a(Unknown Source)  at com.company.a.a(Unknown Source) |
| keep | Leaves debug information without modifications. This option can be helpful for internal testing of your application. In other cases it's better to choose other options.  留下没有修改的调试信息。此选项对应用程序的内部测试很有帮助。在其他情况下，最好选择其他选项。  Reported stack trace will look like this:  java.lang.NullPointerException  at com.company.c.a(Util.java:38)  at com.company.b.b(TraceTest.java:53)  at com.company.b.a(TraceTest.java:14)  at com.company.a.a(Main.java:27) |

Example:

<property name="line-numbers" value="obfuscate"/>

#### 1.6.5.2 generics泛型

|  |  |
| --- | --- |
| **Value** | **Description** |
| keep | *(default)* If you are using reflection to determine generic types or need to compile other classes using the obfuscated jar as a library, then you need to keep generic type signatures.  （默认）如果你通过反射来确定泛型类型，或需要使用经过混淆处理的 jar作为一个库来编译其他类，您需要保持泛型类型签名。 |
| remove | Generic types information will be removed, i.e. a vector of strings Vector<String> will be seen as a generic Vector. It doesn't affect the performance, makes jar smaller, and is the recommended option.  泛型类型信息将被删除，亦即一个字符串向量 Vector <String>将被看作是一个通用的向量。它不会影响性能，使jar文件较小，并且是推荐的选项。 |

Example:

<property name="generics" value="remove"/>

This property can be set for the specified classes using apply2class attribute. The apply2class attribute has the same format as [template](http://www.allatori.com/doc.html" \l "keep-names-class-template) attribute of the class tag. Here is an example:

可以使用apply2class属性为指定的类设置此属性。Apply2class 属性和class标记下的[template](http://www.allatori.com/doc.html" \l "keep-names-class-template)属性具有相同的格式。下面是一个示例︰

<!-- Keeping generic types information in com.abc package's classes 在com.abc包的类中保持泛型类型信息-->

<property name="generics" value="keep" apply2class="class com.abc.\*"/>

<!-- Removing generic types information in all other classes在所有其他类中删除泛型类型信息-->

<property name="generics" value="remove"/>

#### 1.6.5.3 inner-classes内部类

|  |  |
| --- | --- |
| Value | Description |
| keep | *(default)* Java compiler adds informational attributes with inner classes names. These attributes will be kept in obfuscated classes.  （默认）Java 编译器添加内部类名称的信息属性。这些属性将保存在经过混淆处理的类中。 |
| remove | Informational attributes will be removed, class hierarchy will be harder to restore. It doesn't affect the performance, makes jar smaller, and is the recommended option.  信息属性将被删除，类层次结构将很难恢复。它不会影响性能，使jar文件较小，并且是推荐的选项。 |

Example:

<property name="inner-classes" value="remove"/>

This property can be set for the specified classes using apply2class attribute. The apply2class attribute has the same format as [template](http://www.allatori.com/doc.html" \l "keep-names-class-template) attribute of the class tag. Here is an example:

可以使用apply2class属性为指定的类设置此属性。Apply2class 属性和class标记下的[template](http://www.allatori.com/doc.html" \l "keep-names-class-template)属性具有相同的格式。下面是一个示例︰

<!-- Keeping inner classes information in com.abc package's classes在com.abc包的类中保持内部类信息-->

<property name="inner-classes" value="keep" apply2class="class com.abc.\*"/>

<!-- Removing inner classes information in all other classes 在所有其他类中删除内部类信息-->

<property name="inner-classes" value="remove"/>

#### 1.6.5.4 member-reorder成员重新排序

|  |  |
| --- | --- |
| Value | Description |
| enable | *(default)* Usually, developers place related methods and fields one after the other in the source file. This sequence is kept after the compilation process. Allatori will shuffle fields and methods.  （默认）通常情况下，开发者可以将相关的方法和字段一个借一个的放在源文件中。这个顺序是在编译过程后保存。Allatori 将会对所有的字段和方法重新排序。 |
| disable | Member reordering is disabled. 禁用成员重新排序。 |

Example:

<property name="member-reorder" value="enable"/>

#### 1.6.5.5 finalize最终化

|  |  |
| --- | --- |
| **Value** | **Description** |
| disable | *(default)* Class finalizing is disabled. （默认）类确定将被禁用。 |
| enable | Classes with no subclasses (leaf classes) will be declared final. This feature should be used only for obfuscating stand-alone applications. It can make your application run faster.  没有子类（叶子类）的类将声明为 final。此功能应仅用于混淆处理独立的应用程序。它可以使您的应用程序运行更快。 |

Example:

<property name="finalize" value="enable"/>

#### 1.6.5.6 version-marker版本标记

|  |  |
| --- | --- |
| Value | Description |
| valid Java identifier name  有效的 Java 标识符名称 | Allatori will use the given identifier as a name for some renamed methods and fields. This will mark the obfuscated class files. You can use it to mark the demo version of your product. For instance, Allatori demo version is marked with "ALLATORI\_DEMO" string. Note that demo version of Allatori marks obfuscated jars and adds "ALLATORI\_DEMO\_" to whatever you use as a value of this property.  Allatori 将使用给定的标识符作为一些改名的方法和字段的名称。这将标记混淆处理后的类文件。你可以使用它来标记您的产品的演示版本。例如，Allatori 演示版本被标有"ALLATORI\_DEMO"字符串。注意Allatori的demo版本标记混淆后的jar文件，不管你设置什么值，都会把"ALLATORI\_DEMO\_"添加到属性的值中。 |

Example:

<property name="version-marker" value="THIS\_IS\_DEMO\_VERSION"/>

#### 1.6.5.7 synthetize-methods合成方法

|  |  |
| --- | --- |
| Value | Description |
| private | *(default)* All private methods will be marked as synthetic.  （默认）所有的私有方法将标记作为合成。 |
| all | All methods will be marked as synthetic. 所有的方法将标记作为合成。 |
| package | All package visible methods will be marked as synthetic.  所有包的可见方法将都标记作为合成。 |
| protected | All protected methods will be marked as synthetic.  所有protected方法将标记作为合成。 |
| public | All public methods will be marked as synthetic. 所有public方法将标记作为合成。 |
| disable | Allatori will not mark methods as synthetic. Allatori不会标记作合成方法。 |

Some decompilers do not output synthetic methods.   
  
Example:

<property name="synthetize-methods" value="all"/>

The property can be used more than once: 可以不止一次使用该属性︰

<property name="synthetize-methods" value="private"/>

<property name="synthetize-methods" value="package"/>

<property name="synthetize-methods" value="protected"/>

The property can be applied to the specified classes using apply2class attribute. The apply2class attribute has the same format as [template](http://www.allatori.com/doc.html" \l "keep-names-class-template) attribute of the class tag. Here is an example:

可以使用apply2class属性为指定的类设置此属性。Apply2class 属性和class标记下的[template](http://www.allatori.com/doc.html" \l "keep-names-class-template)属性具有相同的格式。下面是一个示例︰

<property name="synthetize-methods" value="all" apply2class="class com.abc.\*"/>

<property name="synthetize-methods" value="private"/>

#### 1.6.5.8 synthetize-fields合成属性（字段）

|  |  |
| --- | --- |
| **Value** | **Description** |
| disable | *(default)* Allatori will not mark fields as synthetic. （默认值）Allatori 不会标记字段作为合成。 |
| all | All fields will be marked as synthetic. 所有字段将被都标记为合成。 |
| private | All private fields will be marked as synthetic. 所有的private字段将被标记为合成。 |
| package | All package visible fields will be marked as synthetic.  所有包可见字段将被都标记为合成。 |
| protected | All protected fields will be marked as synthetic. 所有protected字段将被标记为合成。 |
| public | All public fields will be marked as synthetic. 所有public字段将被标记为合成。 |

Some decompilers do not output synthetic fields.   
  
Example:

<property name="synthetize-fields" value="all"/>

The property can be used more than once: 可以不止一次使用该属性︰

<property name="synthetize-fields" value="private"/>

<property name="synthetize-fields" value="package"/>

<property name="synthetize-fields" value="protected"/>

The property can be applied to the specified classes using apply2class attribute. The apply2class attribute has the same format as [template](http://www.allatori.com/doc.html" \l "keep-names-class-template) attribute of the class tag. Here is an example:

可以使用apply2class属性为指定的类设置此属性。Apply2class 属性和class标记下的[template](http://www.allatori.com/doc.html" \l "keep-names-class-template)属性具有相同的格式。下面是一个示例︰

<property name="synthetize-fields" value="all" apply2class="class com.abc.\*"/>

<property name="synthetize-fields" value="private"/>

#### 1.6.5.9 remove-toString删除toString方法

|  |  |
| --- | --- |
| **Value** | **Description** |
| disable | *(default)* toString methods would not be removed.  （默认值） toString 方法不会被删除。 |
| enable | Allatori will remove toString() method in obfuscated classes. toString method can reveal some information about the class and is often used for debug only, therefore it can be removed.  Allatori 将从经过混淆处理的类中删除 tostring()方法。  toString 方法可以揭示出一些关于类的信息，经常仅用于调试，因此它可以被删除。 |

Example:

<property name="remove-toString" value="enable"/>

The property can be applied to the specified classes using apply2class attribute. The apply2class attribute has the same format as [template](http://www.allatori.com/doc.html" \l "keep-names-class-template) attribute of the class tag. Here is an example:

可以使用apply2class属性为指定的类设置此属性。Apply2class 属性和class标记下的[template](http://www.allatori.com/doc.html" \l "keep-names-class-template)属性具有相同的格式。下面是一个示例︰

<property name="remove-toString" value="enable" apply2class="class com.abc.\*"/>

<property name="remove-toString" value="enable" apply2class="class com.xyz.\*"/>

#### 1.6.5.10 remove-calls删除调用

|  |  |
| --- | --- |
| **Value** | **Description** |
| ClassName.methodName | Allatori will remove calls to the specified method. Useful to remove debug logging calls. ClassName and methodName can contain \* to match several classes/methods.  Allatori 将删除对指定方法的调用。这对移除调试日志记录调用有用。类名和方法名称可以包含“\*”来匹配几个类或者方法。 |

Examples:

<property name="remove-calls" value="android.util.Log.d"/>

<property name="remove-calls" value="android.util.Log.\*"/>

The property can be applied to the specified classes using apply2class attribute. The apply2class attribute has the same format as [template](http://www.allatori.com/doc.html" \l "keep-names-class-template) attribute of the class tag. Here is an example:

可以使用apply2class属性为指定的类设置此属性。Apply2class 属性和class标记下的[template](http://www.allatori.com/doc.html" \l "keep-names-class-template)属性具有相同的格式。下面是一个示例︰

<!-- Removing Logger.debug calls from classes in com.abc package -->

<!--从 com.abc 包中的类中删除 Logger.debug 调用-->

<property name="remove-calls" value="com.package.Logger.debug" apply2class="class com.abc.\*"/>

### 1.6.6 Incremental obfuscation properties增量混淆处理属性

#### 1.6.6.1 incremental-obfuscation增量混淆处理

|  |  |
| --- | --- |
| Value | Description |
| name of the previously created log file  之前创建的日志文件的名称 | Sets the [log file](http://www.allatori.com/doc.html" \l "property-log-file) of the previous Allatori run. Relative paths are resolved against configuration file location.  设置Allatori 之前运行的日志文件。相对路径是相针对于配置文件的路径位置。  Incremental obfuscation is used when you need to create a patch or add-on to your application. In such case you need guarantee, that new names of classes, methods and fields are consistent with the previously obfuscated version. Using log file generated during the previous run of Allatori as input when obfuscating the next release, makes these two releases fully compatible. So a patch or add-on can be seamlessly integrated into the previously deployed application.  当您需要创建一个修补程序或附加程序到您的应用程序中去时，可以使用增量混淆处理。在这种情况下你需要保证，新的类、方法和字段的名称与以前混淆处理的版本一致。在混淆处理下一个版本时，利用上一次Allatori运行期间生成的日志文件作为输入，可以使这两个版本完全兼容。所以修补程序或加载项可以无缝地集成到以前部署的应用程序。  When using incremental obfuscation you should include in the obfuscation process all your classes even if only some of them are to be distributed.  使用增量模糊处理时，你应在模糊处理过程中包括所有类，即使只有一部分类将被分发。 |

Example:

<property name="incremental-obfuscation" value="input-renaming-log.xml"/>

#### 1.6.6.2 unique-renaming唯一重命名

|  |  |
| --- | --- |
| Value | Description |
|  |  |
| disable | *(default)* Unique renaming is disabled. （默认值）唯一重命名将被禁用。 |
| enable | If any two methods have the same name and signature, then these methods will be renamed to the same new name. If any two methods have different name/signature, then these methods will have different names after renaming. It ensures consistency during subsequent incremental obfuscation runs.  如果任何两种方法具有相同名称和签名，那么这些方法会被重命名为相同的新名称。如果任何两种方法具有不同的名称签名，那么这些方法会被重命名为不同的名称。在随后的增量模糊处理运行期间，它可以确保一致性。 |

Example:

<property name="unique-renaming" value="enable"/>

### 1.7 Ignore-classes tag忽略类标记

The ignore-classes tag is used to completely exclude some classes from the obfuscation process. These classes are copied "as is" with no changes to the output jar file. Note that ignored classes will reference other classes by their original names. You cannot rename classes/methods referenced by ignored classes.   
忽略类标记用于从混淆处理过程中完全排除某些类。这些类会被原样复制到输出 jar 文件中。请注意，被忽略的类引用其他类时还会沿用原来的名称。你不能重命名被忽略类引用的类或者方法的名称。

The tag contains nested class tags which are the same as [class tags in the keep-names section](http://www.allatori.com/doc.html" \l "keep-names-class).

该标记包含了嵌套的class标记，该标记和keep-names一节中class标记相同。

Example:

<ignore-classes>

<class template="class com.company.abc.\*"/>

<class template="class com.company.xyz.SomeClass"/>

</ignore-classes>

# 2 Annotations注释

Annotation classes are located in the allatori-annotations.jar. All Allatori annotations are removed during the obfuscation process, so you don't need this jar file in the runtime. Annotations are used for easier and more accurate configuration of the obfuscator.

注释类位于 allatori-annotations.jar。所有的 Allatori 注释都会在混淆处理过程中被删除，所以在运行过冲中你根本用不到此 jar 文件。注释用于对混淆处理程序更容易和更精确的进行配置。  
Here all are available annotations located in the com.allatori.annotations package:

这些事是 com.allatori.annotations 包中可用的注释︰

|  |  |  |
| --- | --- | --- |
| Annotation | Applicable to  适用于 | Description |
| Rename | classes, methods and fields | Instructs Allatori to rename the annotated element. Overrides configuration file settings.  指示Allatori命名该注释元素。重写配置文件设置。 |
| DoNotRename | classes, methods and fields | Instructs Allatori not to rename the annotated element. Overrides configuration file settings.  指示Allatori不去命名该注释元素。重写配置文件设置。 |
| StringEncryption | classes | Possible values are: 可能的值有︰     StringEncryption.ENABLE      StringEncryption.DISABLE      StringEncryption.MAXIMUM      StringEncryption.MAXIMUM\_WITH\_WARNINGS  Overrides [string-encryption](http://www.allatori.com/doc.html" \l "property-string-encryption) property. Example:  重写[string-encryption](http://www.allatori.com/doc.html" \l "property-string-encryption)属性。示例︰ @StringEncryption(StringEncryption.ENABLE) |
| StringEncryptionType | classes | Possible values are: 可能的值有︰     StringEncryptionType.FAST      StringEncryptionType.STRONG  Overrides [string-encryption-type](http://www.allatori.com/doc.html" \l "property-string-encryption-type) property. Example:  重写[string-encryption-type](http://www.allatori.com/doc.html" \l "property-string-encryption-type)属性。示例︰ @StringEncryptionType(StringEncryptionType.STRONG) |
| ControlFlowObfuscation | classes and methods | Possible values are: 可能的值有︰     ControlFlowObfuscation.ENABLE      ControlFlowObfuscation.DISABLE  Overrides [control-flow-obfuscation](http://www.allatori.com/doc.html" \l "property-control-flow-obfuscation) property. Example:  重写控制[control-flow-obfuscation](http://www.allatori.com/doc.html" \l "property-control-flow-obfuscation)属性，示例︰ @ControlFlowObfuscation(ControlFlowObfuscation.ENABLE) |
| ExtensiveFlowObfuscation广泛流模糊处理 | classes and methods | Possible values are: 可能的值有︰     ExtensiveFlowObfuscation.DISABLE      ExtensiveFlowObfuscation.NORMAL      ExtensiveFlowObfuscation.MAXIMUM  Overrides [extensive-flow-obfuscation](http://www.allatori.com/doc.html" \l "property-extensive-flow-obfuscation) property. Example:  重写[extensive-flow-obfuscation](http://www.allatori.com/doc.html" \l "property-extensive-flow-obfuscation)属性。示例︰ @ExtensiveFlowObfuscation(ExtensiveFlowObfuscation.MAXIMUM) |

# 3 Android obfuscation Android 的模糊处理

Allatori performs full-featured obfuscation of Android applications and can be easily integrated into the build process. We've created a typical configuration file for Android projects, making it very easy to obfuscate Android application. You can find the configuration file (allatori.xml) and sample build files (build.gradle and build.xml) in the tutorial that comes with Allatori distribution.

Allatori 针对Android 应用程序进行全功能混淆，并可以方便地集成到构建过程。我们已经创建 Android 项目的典型的配置文件，使Android 应用程序的混淆很容易。您可以在Allatori发行版附带的教程中找到配置文件（allatori.xml）和示例构建文件（build.gradle和build.xml）。

## 3.1 Using Android Studio使用 Android Studio

Here are three steps to set up Allatori for your Android Studio project:

这里是为你的Android Studio项目设置 Allatori 的三个步骤︰

1.Create allatori folder in your rootDir folder. Copy allatori.jar to the created folder;   
2. Copy allatori.xml to your projectDir folder (where project's build.gradle is) from our tutorial;   
3. Edit build.gradle:

1.在 rootDir 文件夹中创建 allatori 文件夹。将 allatori.jar 复制到创建的文件夹中;

2. 从教程中复制 allatori.xml 到目录文件夹 （其中项目的 build.gradle 在该文件夹）;

3.编辑 build.gradle

android {

...

applicationVariants.all { variant ->

variant.javaCompile.doLast {

runAllatori(variant)

}

}

}

def runAllatori(variant) {

copy {

from "$projectDir/allatori.xml"

into "$buildDir/intermediates/classes/"

expand(classesRoot: variant.javaCompile.destinationDir,

androidJar: "${android.sdkDirectory}/platforms/${android.compileSdkVersion}/android.jar",

classpathJars: variant.javaCompile.classpath.getAsPath(),

logFile: "allatori-log-${variant.name}.xml")

rename('allatori.xml', "allatori-${variant.name}.xml")

}

new File("${variant.javaCompile.destinationDir}-obfuscated").deleteDir()

javaexec {

main = 'com.allatori.Obfuscate'

classpath = files("$rootDir/allatori/allatori.jar")

args "$buildDir/intermediates/classes/allatori-${variant.name}.xml"

}

new File("${variant.javaCompile.destinationDir}").deleteDir()

new File("${variant.javaCompile.destinationDir}-obfuscated").renameTo(new File("${variant.javaCompile.destinationDir}"))

}

## 3.2 Using Ant使用 Ant

Here are three steps to set up Allatori for your Android Ant project:

这里是为你安卓 Ant 项目设置 Allatori 的三个步骤︰   
1. Create allatori folder in your project's folder. Copy allatori.jar to the created folder;   
2. Copy allatori.xml to your project's folder from our tutorial;   
3. Add the following target to your build.xml:

1.在您的项目文件夹中创建 allatori 文件夹。将 allatori.jar 复制到创建的文件夹中;

2. 从我们的教程复制 allatori.xml 到您的项目文件夹;

3.将下列目标添加到您的 build.xml文件中︰

<target name="-obfuscate" unless="do.not.compile">

<taskdef name="allatori" classname="com.allatori.ant.ObfuscatorTask"

classpath="allatori/allatori.jar"/>

<delete dir="${out.classes.absolute.dir}-obfuscated"/>

<allatori config="allatori.xml"/>

<property name="out.dex.input.absolute.dir"

value="${out.classes.absolute.dir}-obfuscated"/>

</target>

You may need to perform further configuration for your project. Properties defined in the Ant build file can be referenced from the Allatori configuration file using standard Ant syntax: ${PropertyName}.

您可能需要为您的项目执行进一步的配置。在 Ant 构建文件中定义的属性可以从使用 Ant 的标准语法的 Allatori 配置文件引用: ${属性名}。

# 4 Eclipse IDE plugin (Eclipse IDE 插件)

Allatori can be easily used with Eclipse IDE:   
  1. Copy allatori.jar to the eclipse/dropins folder;   
  2. Right-click your project's name in Eclipse, choose Configure -> Add Allatori Builder in the popup menu ([see screenshot](http://www.allatori.com/img/eclipse.png));   
  3. Clean rebuild the project, obfuscation process runs only on clean builds.   
通过 Eclipse IDE可以很容易的使用Allatori:

1.将 allatori.jar 复制到 eclipse/dropins 文件夹中;

2.右键单击您的项目名称在 Eclipse 中，在弹出菜单中选择配置-添加 Allatori Builder > （见截图）;

3.清除重新生成该项目，混淆处理过程仅在clean builds运行。

The default allatori.xml configuration file will be created in the project's root folder during the first run.

在第一次运行期间将项目的根文件夹中创建默认的 allatori.xml 配置文件。

You can use ${eclipse-input} and ${eclipse-classpath} properties in the configuration file:

<input>

${eclipse-input}

</input>

<classpath>

${eclipse-classpath}

</classpath>

Feel free to [contact us](http://www.allatori.com/contact.html) with any questions.

[Buy now](http://www.allatori.com/price.html) | [Download](http://www.allatori.com/downloads/Allatori-6.0-Demo.zip) | [Overview](http://www.allatori.com/overview.html) | [Features](http://www.allatori.com/features.html) | [Documentation](http://www.allatori.com/doc.html) | [Changelog](http://www.allatori.com/changelog.html) | [Contact](http://www.allatori.com/contact.html)

Copyright © 2005-2016 Smardec Inc.  
All rights reserved