2014

Sonar 代码质量管理

修订历史

版本号	作者	修订章节	修订原因	修订日期	审阅人
V1.0	王善勇		创建文档	2014/03/31	

目录

一、 问题简介

1. 问题 严重

Exception handlers should provide some context and preserve the original exception

Throwable.printStackTrace(...) should never be called

描述：

Java 捕获异常后，应当记录两个重要信息。

- 一些上下文，方便问题重现。
- 原始的异常，为了方便获取原始异常的 message 和栈跟踪。

不符合规范的代码

```
// Noncompliant - exception is lost
try { /* ... */ } catch (Exception e) { LOGGER.info("context"); }

// Noncompliant - context is required
try { /* ... */ } catch (Exception e) { LOGGER.info(e); }

// Noncompliant - exception is lost (only message is preserved)
try { /* ... */ } catch (Exception e) { LOGGER.info(e.getMessage()); }

// Noncompliant - exception is lost
try { /* ... */ } catch (Exception e) { throw new
RuntimeException("context"); }
```

符合规范的代码

```
try { /* ... */ } catch (Exception e) { LOGGER.info("context", e); }

try { /* ... */ } catch (Exception e) { throw new
RuntimeException("context", e); }

try {
  /* ... */
} catch (RuntimeException e) {
  doSomething();
  throw e;
} catch (Exception e) {
  // Conversion into unchecked exception is also allowed
  throw new RuntimeException(e);
}
```

2. 问题 严重

Security - Array is stored directly

描述：

构造函数或者方法接收了数组参数，应该克隆该参数并使用和保存它的副本。

- 直接引用或者赋值。

解决：

使用 System.arraycopy。

3. 问题 严重 -> 主要

Switch cases should end with an unconditional break statement

描述：

switch 中必须有 default 标签，default 中最好也使用 break 或者返回型语句（throw，return）结尾。

Case 语句中除非是 empty，否则必须使用 break 或者返回型语句（throw，return）结尾。

解决：

在非空的 case、default 语句中添加 break 或者返回型语句。

4. 问题 严重

Broken Null Check

描述：

调用方法的对象可能为 null，会抛出 NullPointerException。对象的 null 检查可能出现了逻辑错误，在应该使用 && 的地方，使用了 || 。

com.greenline.hrs.biz.manager.external.impl.wxzyy.WXZYYShiftCaseManagerImpl.

解决：

查看代码逻辑，排除调用 null 对象方法的可能。

5. 问题 主要 -> 次要

Insufficient branch coverage by unit tests

描述：

单元测试应该到达测试一个类方法分支的阈值，否则会出现这个问题。

- 阈值是可以更改的，现在是 65%。

解决：
单元测试应该到达测试一个类方法分支的阈值，否则会出现这个问题。

6. 问题 主要

Constant names should comply with a naming convention
描述：
常量名应当符合命名规范，命名的正则"^[A-Z][A-Z0-9]*(_[A-Z0-9]+)*$"。

解决：
查看静态成员变量和枚举类型变量是否为大写字母、数字和下划线的组合。

7. 问题 主要

Strings literals should be placed on the left side when checking for equality。包括 equals()和 equalsIgnoreCase()。
描述：
字符串字面量在与字符串变量检查是否相等时，应当放到判断的左边。

符合规范的代码

```
// Compliant - properly deals with the null case
System.out.println("Equal?" + "foo".equals(myString));
```

不符合规范的代码

```
String myString = null;

// Non-Compliant - will raise a NPE
System.out.println("Equal? "+myString.equals("foo"));
// Non-Compliant - null check could be removed
System.out.println("Equal? "+(myString != null && myString.equals("foo")));
```

8. 问题 主要

Methods should not be too complex
描述：
方法不要太复杂。方法的圈复杂度是通过统计构造函数、方法（除了 getter 和 setter）、

静态初始化、实例初始化中（&&、||）操作和（if、while、do、for、?:、catch、switch、case、return、throw）语句的数量计算的，最后的返回语句不算（throw 和 return）。如果圈复杂度超过了设定的阈值就会报告出错。

- 阈值是可以更改的，现在是 15。

解决：

检查类是否违反了单一原则，是否需要被重构。

9. 问题 主要 –> 次要

Avoid commented-out lines of code

描述：

避免代码被临时的注释掉。

- 注释的代码很难被再重新利用。

解决：

不要将不必要的代码 patch 到主分支上。删除主分支中已经遗忘用途或者没有用途的注释代码。

10. 问题 主要

Loggers should be "private static final" and should share a naming convention

描述：

Logger 应当被“private static final”修饰，并且符合命名规范。

解决：

private static final Logger LOG(?:GER)? ……

11. 问题 主要

Duplicated blocks

描述：

重复代码块。

解决：

检查类重复的代码是否可以提取重构。

12. 问题 主要

Generic exceptions Error, RuntimeException, Throwable and Exception should never be thrown。

描述：

通用的 Throwable、Error、Excetpion、RuntimeExcetpion 不应当被抛出。

解决：

抛出自己定义相应的 Exception。

```
public void foo(String bar) throws Throwable { // 不符合
  throw new RuntimeException("My Message");    // 不符合
}

public void foo(String bar) {
  throw new MyRuntimeException("My Message");  // 符合
}
```

13. 问题 主要

Collection.isEmpty() should be used to test for emptiness

描述：

不要使用 Collection.size()方法来判断 collection 是否为空，使用 isEmpty 方法可读性更高。

解决：

```
if (myCollection.isEmpty())
```

14. 问题 主要

Field names should comply with a naming convention

Constants should be declared "final static" rather than merely "final"

Local variable and method parameter names should comply with a naming convention

描述：

需要注意成员变量的命名。本地成员变量和方法参数命名需要使用驼峰命名规则，符合正则表达式"^[a-z][a-zA-Z0-9]*$"。

常量需要使用 static final 修饰，而不仅仅是 final。

解决：

符合规范的代码

```
public class Myclass {
```

```
  public static final THRESHOLD = 3;    // Compliant
}
```

不符合规范的代码

```
public class Myclass {
  public final THRESHOLD = 3;          // Non-Compliant
}
```

15. 问题 主要

Merging collapsible if statements increases the code's readability.

描述：

合并可折叠的 if 语句，提高代码可读性。

解决：

符合规范代码

```
if (condition1 && condition2) { // Compliant
  /* ... */
}
```

不符合规范代码

```
if (condition1) {
  if (condition2) {            // Non-Compliant
    /* ... */
  }
}
```

16. 问题 主要

Method parameters, caught exceptions and foreach variables should not be reassigned

描述：

方法参数、捕获的异常、foreach 迭代的变量不应该被重新赋值。

```
public int add(int a, int b) {
    a = a + b;                    // 不符合规范
    /* additional logic */
    return a;                     // Seems like the parameter is returned as
is, what is the point?
  }
```

解决：

不要给这些变量重新赋值，使用方法临时变量代替。

17. 问题 主要

Utility classes should not have a public constructor

描述：

工具类不应该拥有公共构造函数。

解决：

为工具类创建私有构造函数。

```
private XXXUtils {
……
}
```

18. 问题 主要

Use Index Of Char

描述：

当调用 String.indexOf 或者 String.lastIndexOf 方法时，参数是一个字符的时候，使用 char 型参数比 String 型参数更加高效。

解决：

String.indexOf(‘y’)、String.lastIndexOf(‘\n’)。

19. 问题 主要

Class variable fields should not have public accessibility

描述：

成员变量不应当被 public 修饰符修饰。Public 的成员变量不符合封装原则，有两个缺点

- 增量式的动作不可控，例如检验动作的合法性。
- 内部展现被暴露后，后续不能更改。

解决：

将内部成员变量使用非 public 修饰，提供可行的访问方法。

20. 问题 主要

if/else/for/while/do statements should always use curly

描述：

没有使用大括号包括代码块。

解决：

使用大括号把/else/for/while/do 的代码块括起来。

21. 问题 主要

Right curly brace and next "else", "catch" and "finally" keywords should be located on the same line

描述：

大括号的右半部分应当和紧挨着的"else", "catch" and "finally"关键字在一行。

解决：

```
public void myMethod() {
  if(something) {
    executeTask();
  } else if (somethingElse) {         // 符合
    doSomethingElse();
  }
  else {                             // 不符合
     generateError();
  }

  try {
    generateOrder();
  } catch (Exception e) {             // 符合
    log(e);
  }
  finally {                          // 不符合
    closeConnection();
  }
}
```

22. 问题 主要

System.out and System.err should not be used as loggers

描述：

System.out and System.err 不应当用于日志记录。

解决：

使用 Log 记录。

23. 问题 主要

Empty arrays and collections should be returned instead of null

描述：

返回 null 而不是数组或者集合会让调用者检查为 null 的情况，是代码更复杂、可读性更差。在大多数情况下，null 经常被用来表示空的同义词。

解决：

返回 new Result[0] 或者 Collections.EMPTY_LIST。

代码示例

```
public static Result[] getResults() {
  return null;                          // Non-Compliant
}
public static void main(String[] args) {
  Result[] results = getResults();

  if (results != null) {                // Nullity test required to prevent
NPE
    for (Result result: results) {
      /* ... */
    }
  }
}
```

```
public static Result[] getResults() {
  return new Result[0];                 // Compliant
}

public static void main(String[] args) {
  for (Result result: getResults()) {
    /* ... */
  }
}
```

24. 问题 主要

Synchronized classes Vector, Hashtable, Stack and StringBuffer should not be used

描述：

这些早期的 java 类内部带有同步机制，影响单线程使用性能。如果多线程，可以自己控制锁使用。

解决：

使用非同步类代替。

- `ArrayList` or `LinkedList` instead of `Vector`
- `Deque` instead of `Stack`
- `HashMap` instead of `Hashtable`
- `StringBuilder` instead of `StringBuffer`

25. 问题 主要

Unnecessary Local Before Return

描述：

没必要的内部变量。

示例：

//不符合

```
JaxbBinder rspBinder = new JaxbBinder(clazz);
Object obj = rspBinder.fromXml(resultxml);
return obj;
```

//符合

```
JaxbBinder rspBinder = new JaxbBinder(clazz);
return rspBinder.fromXml(resultxml);
```

26. 问题 主要

Unused formal parameter

Unused Private Field

Unused private method

描述：

未使用传递给方法、构造函数的参数。

解决：

删除这些参数，或者使用方法重载。

27. 问题 主要

FIXME tags should be handled

描述：

注释中含有 FIXME 标签，被认为是一个需要修复的主要问题。

解决：

28. 问题 主要

Switch cases should not have too many lines

描述：

Switch cases 语句中含有太多行代码。

解决：

代码重构。

29. 问题 主要

Try-catch blocks should not be neste

描述：

避免嵌套 try-catch 语句，代码可读性差。

解决：

将两个 try-catch 拆分成平行 catch。

```
try {
  try {                                // 不符合
    doSomething();
  } catch (RuntimeException e) {
    /* Ignore */
  }

  doSomethingElse();
} catch (Exception e) {
  /* ... */
}
```

```
try {
  dedicatedMethod();                     // 符合
  doSomethingElse();
} catch (Exception e) {
  /* ... */
}

/* ... */

private void dedicatedMethod() {
  try {                                // 符合
    doSomething();
  } catch (RuntimeException e) {
```

```
    /* Ignore */
  }
}
```

30. 问题 主要

Nested blocks of code should not be left empty

描述：

嵌套代码块不能有空代码块。

示例：

```
void doSomething() {
  for (int i = 0; i < 42; i++)       // Non-Compliant
  {
  }
  for (int i = 0; i < 42; i++);      // Compliant

  if (myVar == 4)                  // Compliant - contains a comment
  {
    // Do nothing because of X and Y
  }
  else                             // Compliant
  {
    doSomething();
  }

  try                              // Non-Compliant
  {
  }
  catch (Exception e)               // Compliant
  {
    // Ignore
  }
}
```

解决：

理清代码逻辑，重构代码。

31. 问题 主要

Deprecated elements should have both the annotation and the Javadoc tag

描述：

废弃的元素应当同时拥有注解和 Javadoc 标签。

示例：

```
class MyClass {

  @Deprecated           // Non-Compliant
  public void foo1() {
  }

  /**
    * @deprecated
    */
  public void foo2() {    // Non-Compliant
  }

  /**
    * @deprecated
    */
  @Deprecated
  public void foo3() {    // Compliant
  }

}
```

32. 问题 主要

Declarations should use Java collection interfaces such as "List" rather than specific implementation classes such as "LinkedList"

描述：

应当使用 Java 集合接口声名变量或者参数，而不是具体实现。

解决：

HashMap -> Map

HashSet -> Set

ArrayList - > List

33. 问题 主要

Boolean expressions should be as compact as possible

描述：

Boolean 表达式可以直接用来作为判定条件，不需要在使用‘==false’、‘!=true’等操作。

示例：

```
if (booleanVariable) { /* ... */ }
if (!booleanVariable) { /* ... */ }
if (booleanVariable) { /* ... */ }
doSomething(true);
```

34. 问题 主要

Non-static class initializers should not be used

描述：

不可使用非静态类初始化。

示例：

//不符合

```
class MyClass {
  private static final Map MY_MAP = new HashMap() {

    // Non-Compliant - HashMap should be extended only to add behavior, not
for initialization
    {
      put("a", "b");
    }

  };
}
```

//符合

```
class MyClass {
  // Compliant
  private static final Map MY_MAP = ImmutableMap.of("a", "b");
}
```

35. 问题 主要

Exception types should not be tested using "instanceof" in catch blocks

描述：

不要在 catch 代码块中使用 instanceof 关键字来判断 Exception 的类型。

解决：

使用多层 catch 代码块捕获不同的 Excetpion。

示例：

不符合

```
try {
  /* ... */
} catch (Exception e) {
  if(e instanceof IOException) { /* ... */ }        // Non-Compliant
  if(e instanceof NullPointerException{ /* ... */ }  // Non-Compliant
}
```

符合

```
try {
  /* ... */
} catch (IOException e) { /* ... */ }              // Compliant
} catch (NullPointerException e) { /* ... */ }      // Compliant
```

36. 问题 主要

Return of boolean expressions should not be wrapped into an if-then-else statement

描述：

Boolean 表达式不需要经过 if-then-else 语句转化返回。

不符合：

```
if (someBooleanMethod()) {    // Non-Compliant
  return true;
} else {
  return false;
}
```

符合：

```
return someBooleanMethod();
```

37. 问题 主要

Compare Objects With Equals

Strings should be compared using equals()

不符合

this.title = “title”

符合

“title”.equals(this.title);

38. 问题 主要

Avoid Throwing Null Pointer Exception

描述：

避免抛出 NullPointerExcetion。

解决：

使用其他 Exception 替代，如 IllegalArgumentExcetpion

二、 Java 语言规范中建议的修饰符顺序:

1. Annotations
2. public
3. protected
4. private
5. abstract
6. static
7. final
8. transient
9. volatile
10. synchronized
11. native
12. strictfp

三、 主要问题

1. 异常捕获、记录、抛出。
2. Format，命名格式，代码格式。
3. 修饰符，使用、修饰顺序。
4. 逻辑判断。