Code ▾ Jump to bottom

Fixing bug report #39 [extract local] Extract to local variable may result in NullPointerException. #66

Merged

jjohnstn merged 9 commits into `eclipse-jdt:master` from `chixiaye:NullPointerExecptionInExtractVariable` on Nov 5, 2022

chixiaye commented on May 25, 2022

This is a patch based on the solution explained in the bug report.

OverView

In total, this patch changes four files: Three are modified and a new file is added.

We list all involved files and explain the changes made on them:

1. Changes on `RefactoringCoreMessages.java` and `refactoring.properties`

Add the warning message about NullPointerException.

2. `ExtractTempRefactoring.java`

First, we add three members in the class `ExtractTempRefactoring` and initialize accordingly in the constructors. `startPoint` and `endPoint` are used to indicate the position of the extracted sequence selected from a to-be-extracted sequence. And `nullWarning` is the NullPointerException flag. Correspondingly, we decide whether to prompt according to the value of the member in `checkFinalConditions` method. It's worth noting that `nullWarning` is true if extracting the selected expression must result in an exception.

Second, since we will select subsequences with extracted sequences, We modify the parameter list of method `findDeepestCommonSuperNodePathForReplacedNodes` to match.

Third, because we're not necessarily extracting the first potential location, we modify the `getFirstReplacedExpression()` function's name and parameter list.

Next, the function `reSortRetainOnlyReplacableMatches()` is added to make sure, the replaceable expressions are sorted in ascending order by the offset value.

Last, function `addReplaceExpressionWithTemp()` is the main part of our modification. As with our proposed solution in the issue, we use a greedy strategy to obtain replaceable sequences.

3. `NullChecker.java`

The Class `NullChecker` is added to detect potential NullPointerException between the two given offsets.

In the constructor, we initialize the required members and calculate the invocationSet. For example, if expression is `a.b().c`, the invocationSet will contain `a`, `a.b()`. We created the class `InvocationVisitor` to finish the task.

As for method `isExistNull`, it is used to determine whether there is a potential NullPointerException. It uses class `NullMiddleCodeVisitor` to find `InfixExpression` such as `a==null`. The `nullFlag` member indicates if a potential NullPointerException was detected.

chixiaye commented on Jun 15, 2022

@jjohnstn Would it be convenient for you to review this PR? Thanks a lot!

jjohnstn commented on Jun 18, 2022

Looking at it now. I added a comment in the initial issue about what I think the proper fix should be.

chixiaye commented on Sep 14, 2022

Based on the suggestions in the issue, We optimized the process of the refactoring when we perform NullPointerException checking.

Besides, three test cases have been appended to verify the changes, which are available in `ExtractTempTests.java`. Test-118 is a simple case for selecting the second `arr.length` to conduct extracting variable. Test-119 is a special case that only one expression `arr.length` would be extracted, such refactoring would be warned. Test-120 is a slightly more complex case, aiming to extract the expression `s.charAt(s.length() - 1)` which may cause NullPointerException if extracting all the expressions together.

jjohnstn commented on Sep 17, 2022

/rebase

github-actions bot force-pushed the `NullPointerExecptionInExtractVariable` branch from `cd12c64` to `cb8359c` 8 months ago

Compare

chixiaye commented on Sep 20, 2022

Thank you very much. If you have any requirements, we are willing to help.

jjohnstn commented on Sep 21, 2022

/rebase

github-actions bot force-pushed the `NullPointerExecptionInExtractVariable` branch from `3147b0d` to `8fca7bc` 8 months ago

Compare

jjohnstn self-requested a review 8 months ago

jjohnstn requested changes on Sep 28, 2022

View reviewed changes

jjohnstn left a comment

Hi, I have a chance to look at the code in more detail now that it is building properly. There are logic flaws in using the toString() method for an ASTNode (see comments in NullChecker.java) and I think the code should avoid adding the local var in the case where there is only one use and adding the variable will precede the null check. Never adding the invalid statement would negate the need for a warning message. Most users will not preview and will simply hit Ok so would never get to see the warning. I don't think it is an issue to omit as most users would never select a single use method call to be extracted to a local variable and your code already does the right thing when there are multiple uses.

If you need help in making the logic correct in NullChecker, let me know. As mentioned, the right way to verify a match is to find a Name and then resolve it's binding at which point you can compare the bindings. Note that resolving a binding may end up returning null in which case, you should probably bail (this can happen if the code has errors). You can make it more robust by using the Name.getFullyQualifiedName() method and don't bother asking for bindings until you get a potential match.

...se.jdt.ui/core refactoring/org/eclipse/jdt/internal/corext/refactoring/util/NullChecker.java

Outdated

```
58 +                }
59 +                @Override
60 +                public void preVisit(ASTNode node) {
61 +                        if(node instanceof MethodInvocation) {
```

jjohnstn on Sep 28, 2022

This logic will not work reliably. The toString() method is not a valid way to determine equality. For example, you could have casting or parenthesized expressions. In addition, variables/fields can have the same names (e.g. a local variable in a block overriding another local variable outside the block). You need to find a Name and then resolve its binding. Then you can verify that the binding of A.isEqualTo(B).

Reply...

...se.jdt.ui/core refactoring/org/eclipse/jdt/internal/corext/refactoring/util/NullChecker.java

Outdated

```
99  +            }
100 +
101 +            @Override
102 +            public boolean preVisit2(ASTNode node) {
```

jjohnstn on Sep 28, 2022

You also need to check for instanceof expressions as these will weed out nulls as well. e.g. if (!(a instanceof String)) { return null; } else {((String)a).length();}

Reply...

...se.jdt.ui/core refactoring/org/eclipse/jdt/internal/corext/refactoring/util/NullChecker.java

Outdated

```
94 +                this.nullFlag=false;
95 +            }
```

```
96  +
97  +           public boolean isNull() {
```

jjohnstn on Sep 28, 2022

I think a better name could be helpful. For example, hasNullCheck()

Reply...

...se.jdt.ui/core refactoring/org/eclipse/jdt/internal/corext/refactoring/util/NullChecker.java — Outdated

```
45  +               this.invocationSet=iv.invocationSet;
46  +       }
47  +
48  +       public boolean isExistNull( ) {
```

jjohnstn on Sep 28, 2022

I think a better name for this could be helpful. hasNullCheck() for example.

Reply...

...se.jdt.ui/core refactoring/org/eclipse/jdt/internal/corext/refactoring/util/NullChecker.java — Outdated

```
112 +                                   Expression leftExpression = infixExpression.get
113 +                                   Expression rightExpression = infixExpression.ge
114 +                                   Expression target=null;
115 +                                   if( rightExpression.getNodeType()==ASTNode.NULL
```

jjohnstn on Sep 28, 2022

If you do a CTRL+Shift+F you can eliminate some formatting issues (like spaces between if and open brackets, etc..

Reply...

org.eclipse.jdt.ui.tests.refactoring/resources/ExtractTemp/canExtract/A_test119_out.java — Outdated

```
2  +
3  + class A {
4  +       public int m(int[] arr) {
5  +               int length= arr.length;
```

jjohnstn on Sep 28, 2022

I think the logic should avoid doing anything for this particular test. This makes it consistent with the case where there are two references and it moves it beyond the if statement. Most people will not do a preview and won't see the warning.

I also ran into another scenario:

```
public void foo(String s) {
        if (s == null || s.length() == 0) {
                System.out.println("here");
        } else {
                int k = s.length();
                System.out.println(k);
        }
        int z = s.length();
        System.out.println(z);
}
```

If I select the first s.length(), the scope of that should be enough to add a local declaration outside the if statement and then set z = length; Instead, it only finds the s.length() calls in the else statement and adds a local variable there, but not for the int z case.

liuhuigmail on Oct 9, 2022

Suppose we add a local declaration 'length=s.length()' before the if statement, it will result in nullpointer exception. However, if we add the declaration after the else statement (just before the declaration of z, we cannot replace the occurence of 's.length()' within the else statement with the new variable.

Although it is possible to declare the variable (but without initialization) before the if statement, it could be clumsy. There is no guarantee whether the variable has been initialized when it is assessed: Initializing it on the else branch only may not guarantee that it has been initialized after the whole if statement. To this end, we should initialized the same variable multiple times (first on the else branch, and then after the whole if statement). I don't think it is better than employing different variables to represent the occurrences within else statements and those outside the statement, respectively.

liuhuigmail on Oct 9, 2022 • edited

A possible solution here is: replace the expressions within the else statement only （as what the commit does）.

A better solution: Don't do anything here. It is odd to delcare (and initialize) a variable and the only use of the variable is to intialize another variable in the form of VariableA=VarialbeB.

`

```
            int length=s.length();

             int k = length;

        System.out.println(k);`
```

Reply...

chixiaye commented on Sep 28, 2022

Thanks for your patient reply. I will read the details and fix the flaws as soon as possible.

chixiaye added a commit to chixiaye/eclipse.jdt.ui that referenced this pull request on Oct 13, 2022

`modify based on pr eclipse-jdt#66` `86e8cce`

chixiaye commented on Oct 14, 2022

Hi, we have carefully read your comments and made the following revisions to the code.

1. Remove Warning Message

We remove the warning message when only one expression is extracted and such extracting would result in NullPointerExecption. Correspondingly, we modified test-119 so that the refactored code is the same as the source code. Notably, since `ExtractTempRefactoring` class requires that the source code must be modified, we additionally specify that no modification of the source code is allowed when no safe extracting can be found.

2. Replacing String Matching with Combination of Name Binding and AST Matching

As you suggested, we match different expressions based on name bindings. Notably, for MethodInvocation node, we can only access its method declaration binding. So we reused API `ASTFragmentFactory.createFragmentForFullSubtree(ASTNode node).getSubFragmentsMatching(IASTFragment fragment)` to match the same MethodInvocation nodes in the enclosing body node. Test-121 is a new test case for overriding the member variable with the local variable to validate the matching.

3. Handling of CastExpression and ParenthesizedExpression

To deal with CastExpression and ParenthesizedExpression, before we compare each node, we remove the brackets outside the variables/expressions, which is implemented in the method `getOriginalExpression`. We add test-122 to validate the CastExpression case and test-123 to validate ParenthesizedExpression case.

4. Roughly Ignore InstanceofExpression

As for InstanceofExpression, you provided a case as follows:

```
'''
if (!(a instanceof String)){
    return null;
}
else {
    ((String)a).length();
}
((String)a).length();
'''
```

I think if expressions `((String)a).length()` are extracted and declaration is outside the if-statement, the ClassCastException would be thrown probably in its initializer. Surely, we can also take such extracting as unsafe from the point of view of accessing null object. In my opinion, it's better to analyze such a case from the perspective of ClassCastException rather than from the perspective of NullPointerException which requires more complex conditions analysis. Our previous work proposed an effective approach that is similar to handling NullPointerException. We highly expect to commit as a new PR in the future.

jjohnstn approved these changes on Oct 15, 2022

View reviewed changes

jjohnstn commented on Oct 15, 2022

Thanks for the changes. One test issue I ran into:

```
public class TestLocalVar {

        public void foo(String s) {
                if (s == null || s.length() == 0) {
                        return;
                } else {
                        System.out.println(s.length());
                }
                int z = s.length();
                System.out.println(z);
```

```
        }
}
```

When the test above goes to replace s.length() from the if statement and I have chosen to replace all references, it adds an int length= s.length() statement in the else statement which is fine, but it does not add anything to replace the int z = s.length() statement. It will have to add a 2nd statement (e.g. int length2 = s.length()) or abandon changes in the else statement and move the int length = s.length() statement just before the int z statement.

jjohnstn requested changes on Oct 15, 2022

View reviewed changes

jjohnstn left a comment

See my comment about test case.

chixiaye commented on Oct 27, 2022

Hi, the latest changes have been submitted. According to your suggestion, we try to extract as many as possible expressions if `replace all expressions` option is selected. The variable name will be named `NAME_NO`. For example, test-124 is a test case shown as follows:

```
    public void foo(String s) {
            if (s == null || s.length() == 0) {
                    return;
            } else {
                    int length= s.length();
                    System.out.println(length);
            }
            int length_1= s.length();
            int z = length_1;
            System.out.println(z);

    }
```

Assuming that the user selected the second `s.length()` to conduct extracting local variable, we introduce the name which is recommended by Eclipse or provided by the user to replace it. Considering the third `s.length()` also can be extracted, we use `length_1` to conduct another extraction. However, we do not multiply call `ExtractTempRefactoring` to implement it. We apply an iterative strategy and use HashSet to avoid redundant extractions.

Another thing is that we improve the code, which handles the situation that no expression can be extracted safely. We use TextEditVisitor `to traverse the` TextEdit `tree, and remove the invalid` CopySourceEdit` node. Such change can be more robust since retaining the original condition check.

chixiaye commented on Nov 3, 2022

@jjohnstn Could you please review the latest code? Thank you so much!

chixiaye and others added 8 commits 6 months ago

Fix [extract local] Extract to local variable may result in NullPoint…	···	363bf46
[extract local] Extract to local variable may result in NullPointerEx…	···	6f3d8d0
delete rebundant files		6bbacf5
modify based on pr eclipse-jdt#66		70c4e80
Minor field name edits		b3d079a
fix one test issue		ae5d213
fix a ArrayIndexOutOfBounds exception		3a760ef
update ExtractTempRefactoring.java		✓ dbe0bd6

jjohnstn force-pushed the `NullPointerExecptionInExtractVariable` branch from **072d24f** to **dbe0bd6** 6 months ago

Compare

jjohnstn commented on Nov 5, 2022

@chixiaye Just reviewing now. The patch needed a refresh to sync with master and I had to force push a rebase.

Make some additional fixes	···	✓ 42d21ec

jjohnstn approved these changes on Nov 5, 2022

View reviewed changes

jjohnstn merged commit `2987007` into `eclipse-jdt:master` on Nov 5, 2022

View details

3 checks passed

---

chixiaye commented on Nov 6, 2022

Thanks for your cooperation to fulfill this pr. I look forward to continuing to contribute to Eclipse in the future.

jjohnstn added a commit to jjohnstn/www.eclipse.org-eclipse-news that referenced this pull request on Nov 17, 2022

`Smarter extract to local variable refactoring` ... `c3b8779`

jjohnstn mentioned this pull request on Nov 17, 2022

Smarter extract to local variable refactoring eclipse-platform/www.eclipse.org-eclipse-news#66

Merged

jjohnstn added the noteworthy label on Nov 17, 2022

akurtakov pushed a commit to eclipse-platform/www.eclipse.org-eclipse-news that referenced this pull request on Nov 17, 2022

`Smarter extract to local variable refactoring` ... `7672a9f`

Reviewers

liuhuigmail

jjohnstn

Assignees

No one assigned

Labels

noteworthy

Projects

None yet

Milestone

No milestone

Development

Successfully merging this pull request may close these issues.

None yet

3 participants