

Apache Commons Lang Testing (P3)

Group Name: HelloWorld

Group Members: Zicheng Shan, Chenxu Wang

Repo URL: <https://github.com/chenxu-wang/commons-lang>

Date: Feb 16, 2022

Introduce of Structural Testing

Structural testing is also known as “white box” testing. It focuses on the internal state of the object under test and needs to track the operation of the source code.

Structural testing design techniques include the following code coverage criteria:

([Wikipedia](#))

1. Statement coverage
2. Branch testing
3. Method-level testing
4. Path testing
5. Control flow testing
6. Data flow testing
7. Decision coverage

Advantages

1. Force testers to think carefully about the implementation process and principles of the software.
2. Every branch and path in the code can be detected.
3. Reveal bugs hidden in the code.
4. Test the code thoroughly.

Description of Existing Test Suite

Tool

The code coverage tool that comes with IntelliJ IDEA is used in this project. It can efficiently calculate class coverage rate, method coverage rate, and line coverage rate.

Existing Test Suite Coverage

Before adding any further test cases. The coverage rate of the existing test suite is as follows:

Overall Coverage Rate

	Class coverage	Method coverage	Line Coverage	Branch Coverage
coverage number	290/308	3583/3782	14463/15170	4423/5312
coverage rate	94%	94%	95%	83%

Table 1: Overall coverage rate

Specific Class Coverage Rate

Take StringUtils.java as an example. We subsequently add test cases to it to increase its coverage.

The existing test suite for StringUtils.java is saved in StringUtilsTest.java (Path: org/apache/commons/lang3/StringUtilsTest.java).

	Class coverage	Method coverage	Line Coverage	Branch Coverage
coverage number	1/1	173/251	1217/1705	446/516
coverage rate	100%	66%	71%	86%

Table2: StringUtils.java coverage rate

Uncovered Code Example

Uncovered methods and lines are marked in red by using the built-in coverage tool. Those that have already been covered will be marked in green. Examples are shown in Figure1 and Figure2.

```

8899  @
8900  public static String substringBetween(final String str, final String open, final String close) {
8901  if (!ObjectUtils.allNotNull(str, open, close)) {
8902      return null;
8903  }
8904  final int start = str.indexOf(open);
8905  if (start != INDEX_NOT_FOUND) {
8906      final int end = str.indexOf(close, fromIndex: start + open.length());
8907      if (end != INDEX_NOT_FOUND) {
8908          return str.substring(start + open.length(), end);
8909      }
8910  }
8911  return null;

```

Figure1: Uncovered code example

```

9560  public static String wrap(final String str, final String wrapWith) {
9561  if (isEmpty(str) || isEmpty(wrapWith)) {
9562      return str;
9563  }
9564  return wrapWith.concat(str).concat(wrapWith);
9565  }
9566
9567

```

Figure 2: Covered code example

New Test Cases

StringUtils.java is chosen as the feature to add coverage testing cases. After adding new test cases its method coverage number increased from 174 to 183, and its line coverage number increased from 1224 to 1299. 75 extra lines of code are covered after modification. We cover all branches as much as possible in the newly added test cases.

	Class coverage	Method coverage	Line Coverage	Branch Coverage
coverage number	1/1	183/251	1299/1705	486/565
coverage rate	100%	72%	76%	86%

Table3: StringUtils.java coverage rate after modification

Following are the test cases added in the StringUtilsTest.java:

1. `compare()` method compares two Strings lexicographically, return `int = 0` if `str1` is equal to `str2` (or both null); return `int < 0`, if `str1` is less than `str2`, return `int > 0`, if `str1` is greater than `str2`

```
/**
 * SWE261P3 White box testing for StringUtil
 */

/**
 * Coverage test for int compare(final String str1, final String str2, final boolean nullIsLess) at line [867]
 */
@Test
public void testCompare(){
    assertEquals(0, StringUtil.compare("abc", "abc", false));
    assertEquals(-1, StringUtil.compare(null, "xyz", true));
    assertEquals(1, StringUtil.compare(null, "xyz", false));
    assertEquals(1, StringUtil.compare("xyz", null, true));
    assertEquals(-1, StringUtil.compare("xyz", null, false));
    assertEquals(-23, StringUtil.compare("abc", "xyz", false));
    assertEquals(-23, StringUtil.compare("abc", "xyz", true));
}
```

2. `contains()` method checks if `CharSequence` contains a search `CharSequence`.

```
/**
 * Coverage test for boolean contains(final CharSequence seq, final CharSequence searchSeq) at line [996]
 */
@Test
public void testContains(){
    assertFalse(StringUtil.contains("abc", null));
    assertFalse(StringUtil.contains(null, "abc"));
    assertTrue(StringUtil.contains("abc", "a"));
}
```

3. `containsAny()` method checks if the `CharSequence` contains any character in the given set of characters.

```

/**
 * Coverage test for containsAny(final CharSequence cs, final char... searchChar
s) at line [1054]
 * need modify
 */

@Test
public void testContainsAny(){
    assertFalse(StringUtils.containsAny("xyz", new char[]{}));
    assertFalse(StringUtils.containsAny("", new char[]{}));
    assertFalse(StringUtils.containsAny("xyz", new char[]{'a','b','c'}));
    assertTrue(StringUtils.containsAny("zzabyycdxx", new char[]{'a','z','c'}));
    assertTrue(StringUtils.containsAny("zzabyycdxx", new char[]{'b','y'}));
}

```

4. `containsWhitespace()` method checks whether the given `CharSequence` contains any whitespace characters.

```

/**
 * Coverage test for boolean containsWhitespace(final CharSequence seq) at line
[1411]
 */
@Test
public void testContainsWhitespace(){
    assertFalse(StringUtils.containsWhitespace(""));
    assertFalse(StringUtils.containsWhitespace("abc"));
    assertTrue(StringUtils.containsWhitespace("ab c"));
}

```

5. `equals()` method checks whether two `charSequences` are equal.

```

/**
 * Coverage test for boolean equals(final CharSequence cs1, final CharSequence c
s2) at line [1801]
 */
@Test
public void testEquals(){
    assertTrue(StringUtils.equals(null, null));
    assertFalse(StringUtils.equals(null, "abc"));
    assertFalse(StringUtils.equals("abc", null));
    assertFalse(StringUtils.equals("ab", "abc"));
    assertTrue(StringUtils.equals("abc", "abc"));
    assertFalse(StringUtils.equals("abc", "ABC"));
    assertTrue(StringUtils.equals(" ", " "));
}

```

6. `countMatch()` method counts how many times the char appears in the given string.

```

/**
 * Coverage test for int countMatches(final CharSequence str, final char ch) and
 * int countMatches(final CharSequence str, final CharSequence sub) at line 1453 and
 * 1489
 */
@Test
public void testCountMatch(){
    assertEquals(0,StringUtils.countMatches("",'a'));
    assertEquals(2,StringUtils.countMatches("aa",'a'));
    assertEquals(0,StringUtils.countMatches("", "ab"));
    assertEquals(1,StringUtils.countMatches("abc","ab"));
}

```

7. `endsWithAny()` method checks if a `CharSequence` ends with any of the provided case-sensitive suffixes.

```

/**
 * Coverage test for boolean endsWithAny(final CharSequence sequence, final Char
 * Sequence... searchStrings) at line 1739
 */
@Test
public void testEndsWithAny(){
    assertEquals(false, StringUtils.endsWithAny("", "aa", "zzz"));
    assertEquals(true, StringUtils.endsWithAny("aaa", "a", "aa"));
    assertEquals(false, StringUtils.endsWithAny("aaa", "b", "bb"));
}

```

8. `indexOfAny()` method searches a `CharSequence` to find the first index of any character in the given set of characters.

```

/**
 * Coverage test for int indexOfAny(final CharSequence cs, final char... searchC
 * hars) and int indexOfAny(final CharSequence str, final CharSequence... searchStr
 * s)
 * at line 2759 and 2811
 */
@Test
public void testIndexOfAny(){
    assertEquals(-1,StringUtils.indexOfAny("", 'a', 'b'));
    assertEquals(0,StringUtils.indexOfAny("abc", 'a'));
    assertEquals(-1,StringUtils.indexOfAny("abc", 'd'));
    assertEquals(-1,StringUtils.indexOfAny("", "aa", "bb"));
    assertEquals(-1,StringUtils.indexOfAny("ccc", "aa", "bb"));
    assertEquals(0,StringUtils.indexOfAny("ccc", "cc", "bb"));
}

```

Reference

Wikipedia, Structural Testing, https://en.wikipedia.org/wiki/Structural_testing