LSM Tree 测试评估报告

简介

1. 目的
    - 对本次白盒测试结果进行整理和展示
    - 对测试中存在的不足进行反思和整改
2. 范围
    - 待测软件为LSM Tree
    - 源码链接https://github.com/fantasyzyg/LSM
3. 缩略语
4. 参考资料
    - 《软件测试，一个软件工艺师的方法》
5. 概述
    - 本次测试侧重于白盒测试中的单元测试
    - 主要方法是基于路径和基于数据流的覆盖
    - **对每个待测函数写出了详细的路径和数据流分析报告**，并得到了对应的测试用例

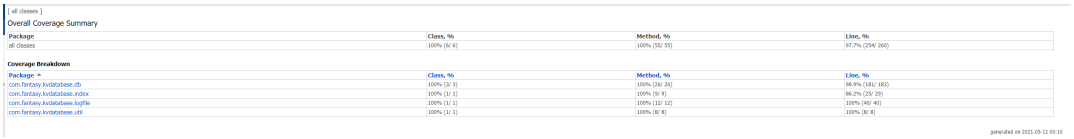
测试结果摘要

- 类覆盖率达到百分之百
- 函数覆盖率达到百分之百
- 总的行覆盖率达到97.7%
- 对每个函数生成了一份路径和数据流分析报告

建议措施

- 单元测试过程能否引入更多的自动化工具？例如自动化分析数据流和路径？

图

1. 测试报告截图
    - 
Overall Coverage Summary

Package	Class, %	Method, %	Line, %
all classes	100% (6/ 6)	100% (55/ 55)	97.7% (254/ 260)

Coverage Breakdown

Package	Class, %	Method, %	Line, %
com.fantasy.kvdatabase.db	100% (3/ 3)	100% (26/ 26)	98.9% (181/ 183)
com.fantasy.kvdatabase.index	100% (1/ 1)	100% (9/ 9)	86.2% (25/ 29)
com.fantasy.kvdatabase.logfile	100% (1/ 1)	100% (12/ 12)	100% (40/ 40)
com.fantasy.kvdatabase.util	100% (1/ 1)	100% (8/ 8)	100% (8/ 8)
2. 测试代码节选

```java
package com.fantasy.kvdatabase.db;

import org.junit.After;
import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;

import java.io.IOException;

public class CompactionHashIdxKvDBTest {
    private CompactionHashIdxKvDB compactionHashIdxKvDB;

    @Before
    public void createBD() {
        compactionHashIdxKvDB = new CompactionHashIdxKvDB();
    }

    @Test
    public void testCurrent() throws IOException {
        compactionHashIdxKvDB.set(String.valueOf(10), String.valueOf(11));
        String result = compactionHashIdxKvDB.get(String.valueOf(10));
        Assert.assertEquals(result,String.valueOf(11));

        for(int i=11;i<15;++i){
            compactionHashIdxKvDB.set(String.valueOf(i), String.valueOf(i+1));
        }
        result = compactionHashIdxKvDB.get(String.valueOf(10));
        Assert.assertEquals(result,String.valueOf(11));
    }

    @After
    public void closeDB() {
        compactionHashIdxKvDB.close();
    }
}
```

```java
package com.fantasy.KVDataBase.DB;

import org.junit.After;
import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;

import java.io.IOException;
import java.util.Map;

public class HashIdxKvDBDBTest {
    private HashIdxKvDB hashIdxKvDB;

    @Before
    public void createBD() throws IOException {
        String path = "D:\\IdeaProjects\\KVDatabase\\src\\database.log";
        String idxPath = "D:\\IdeaProjects\\KVDatabase\\src\\database.idx";
        hashIdxKvDB = new HashIdxKvDB(path, idxPath);
    }

    @Test
    public void testGetIdx() throws IOException {
        Map<String, Long> result = hashIdxKvDB.getIdx();
        Assert.assertTrue(result.size()==10);
        hashIdxKvDB.set("10","10");
        Assert.assertTrue(result.size()==11);
    }

    @Test
    public void testSet() throws IOException {

        for (int i = 0; i < 10; ++i) {
            hashIdxKvDB.set(String.valueOf(i), String.valueOf(i + 1));
        }
        Assert.assertEquals(10, hashIdxKvDB.getIdx().size());
        Assert.assertEquals("1", hashIdxKvDB.get("0"));
        Assert.assertEquals("10", hashIdxKvDB.get("9"));

    }

    @Test
    public void testGet() throws IOException {
        for (int i = 0; i < 10; ++i) {
            hashIdxKvDB.set(String.valueOf(i), String.valueOf(i + 1));
        }
        String result;
        result=hashIdxKvDB.get(String.valueOf(1));
        Assert.assertEquals("2", result);

//        System.out.println(hashIdxKvDB.get(String.valueOf(1)));
        result=hashIdxKvDB.get(String.valueOf(8));
        Assert.assertEquals("9", result);

//        System.out.println(hashIdxKvDB.get(String.valueOf(8)));
        result=hashIdxKvDB.get(String.valueOf(20));
        Assert.assertEquals("", result);

//        System.out.println(hashIdxKvDB.get(String.valueOf(20)));
    }


    @After
    public void closeDB() throws IOException {
        hashIdxKvDB.close();
    }
}
```

3. 路径和数据流分析报告节选

一、简介

对于 CompactionHashIdxKvDB 函数，我们分别采取基于数据流的测试用例生成方法和基于路径的测试用例生成方法。

需要注意的是，为了方便测试和展示，我们对所有的目标测试函数进行行号重排，即行号自函数头从 1 开始重新计数，并忽略代码中的空白行、注释等。

二、程序图

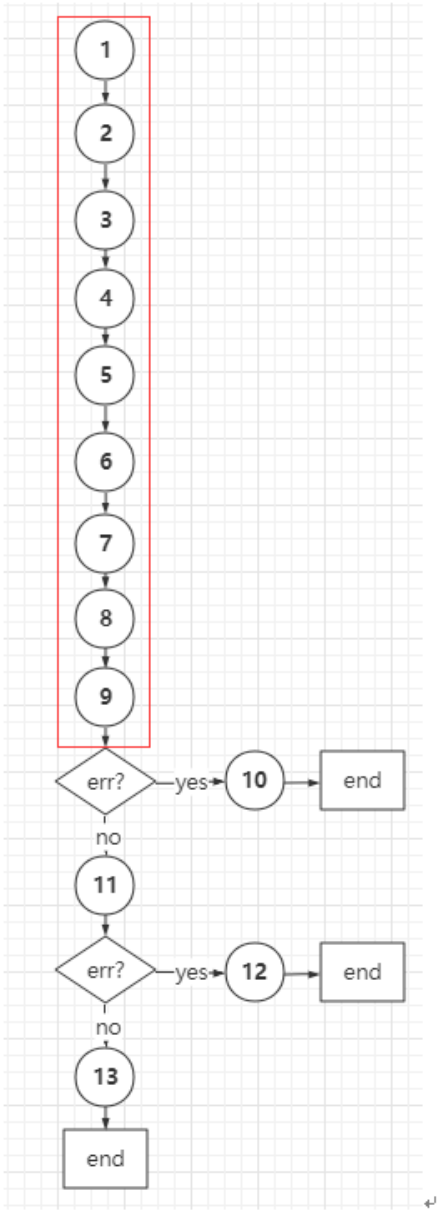


图 1　原始程序图

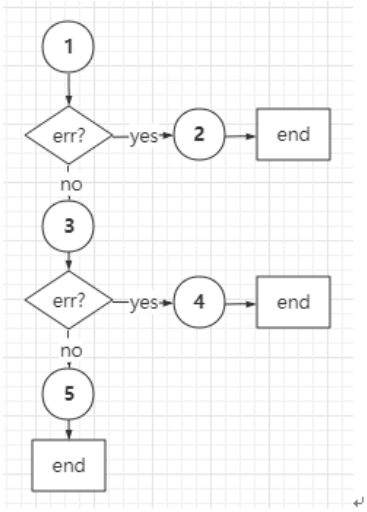


图 2　DD 路径图

三、基于数据流的测试

1、统计变量的定义/使用节点

在对程序图进行简化和绘制后，我们可以对 CompactionHashIdxKvDB 中的所有变量的定义/使用节点进行统计，具体可见表 1。

变量	定义节点 原始程序图	定义节点 简化程序图	使用节点 原始程序图	使用节点 简化程序图
toCompact	1	1	9，11，13	1，3，5
compactedLevel1	2	1	9，11，13	1，3，5
compactedLevel2	3	1	9，11，13	1，3，5
Idx	4	1	9，11，13	1，3，5
toCompactNum	5	1	9，11，13	1，3，5
level1Num	6	1	9，11，13	1，3，5
level2Num	7	1	9，11，13	1，3，5

表 1　CompactionHashIdxKvDB 的变量的定义/使用节点

2、列出变量的定义-使用路径

在列出所有变量的定义/使用节点后，我们可以依据简化程序图的节点情况，依次列出所有变量的定义-使用路径，并且注明其是否为定义-清除路径，具体可见表 2。同时，因为部分路径存在重合部分，因为我们对路径进行了整理，比如用<p1, 4>来代表 p2 的路径。

变量	路径	完整路径	简化路径	是否为定义-清除路径
toCompact	P1	1, 2, 3, 4, 5, 6, 7, 8, 9	<P4, 1>	Y
	P2	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11	<P1, 10, 11>	Y
	P3	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13	<P2, 12, 13>	Y
compactedLevel1	P4	2, 3, 4, 5, 6, 7, 8, 9	<P7, 2>	Y
	P5	2, 3, 4, 5, 6, 7, 8, 9, 10, 11	<P4, 10, 11>	Y
	P6	2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13	<P5, 12, 13>	Y
compactedLevel2	P7	3, 4, 5, 6, 7, 8, 9	<P10, 3>	Y
	P8	3, 4, 5, 6, 7, 8, 9, 10, 11	<P7, 10, 11>	Y
	P9	3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13	<P8, 12, 13>	Y
Idx	P10	4, 5, 6, 7, 8, 9	<P13, 4>	Y
	P11	4, 5, 6, 7, 8, 9, 10, 11	<P10, 10, 11>	Y
	P12	4, 5, 6, 7, 8, 9, 10, 11, 12, 13	<P11, 12, 13>	Y
toCompactNum	P13	5, 6, 7, 8, 9	<P16, 5>	Y
	P14	5, 6, 7, 8, 9, 10, 11	<P13, 10, 11>	Y
	P15	5, 6, 7, 8, 9, 10, 11, 12, 13	<P14, 12, 13>	Y
level1Num	P16	6, 7, 8, 9	<P19, 6>	Y
	P17	6, 7, 8, 9, 10 ,11	<P16, 10, 11>	Y
	P18	6, 7, 8, 9, 10, 11, 12, 13	<P17, 12, 13>	Y
Level2Num	P19	7, 8, 9	7, 8, 9	Y
	P20	7, 8, 9, 10, 11	<P19, 10, 11>	Y
	P21	7, 8, 9, 10, 11, 12, 13	<P20, 12, 13>	Y

表 2　CompactionHashIdxKvDB 的变量的定义-使用路径

3、选取测试覆盖指标，生成测试用例

在这一步，我们选用全定义准则作为测试覆盖指标，即选择每一个定义节点到一个使用节点的定义-清除路径。而对于所有的定义-使用路径，依照表 2 中的简化路径可知，我们可以用尽可能少的测试用例去覆盖尽可能多的目标路径，比如对于路径 P1、P2、P3，我们只需要生成一个可以覆盖 P3 的测试用例即可。而具体的生成的测试用例，我们放在附件中予以展示，此处略去。

4、编写代码，进行测试

上述步骤就是基于数据流的测试用例生成过程，接下来我们就能够编写代码，进行测试。由于按照全使用准则，我们的覆盖率必然是 100%。

四、基于路径的测试

1、绘制程序图、DD 路径图

我们根据源代码，绘制程序图和 DD 路径图。具体图像如文档第二部分所示。

2、选取测试覆盖指标，生成测试用例

这里我们采用分支/条件覆盖的路径测试方法。分支/条件覆盖是执行足够的测试用例，使得分支中每个条件取到各种可能的值，并使每个分支取到各种可能的结果。其缺点是覆盖指标为 C1p，可能会出现覆盖率不足 100% 的情况，有点是测试用例较少，设计较为简单方便。

如果我们采取覆盖率更高的 C∞ 路径测试，虽然覆盖率能够达到 100%，但是测试用例将会呈指数级增加，实际不可行。

所以我们综合数据流与路径测试两种方法，最终使得覆盖率达到 100% 。而具体的生成的测试用例，我们放在附件中予以展示，此处略去。

3、编写代码，进行测试

上述步骤就是基于路径的测试用例生成过程，接下来我们就能够编写代码，进行测试。具体的测试代码，我们放在附件中予以展示，此处略去。