Unit Testing for Multi-Threaded Java Programs

Marcelo Fabri (117903) e Thiago de Oliveira Pires (123153)

Agenda

- Motivação
- Solução Proposta
- Resultados
- Conclusão

Motivação

- Programação concorrente
- Testes unitários
- Bugs nem sempre acontecem na execução dos testes
- Rodar os testes várias vezes?
  - Não é confiável

Solução Proposta

Ideia

- Gerar as possíveis execuções (*schedulings*) para cada teste unitário e executá-las
- Comparar os resultados das diferentes execuções
- Java Path Finder (**JPF**)

Solução Proposta

JPF

- JPF cria diferentes *schedulings* para o teste unitário e as executa
- JPF implementa uma JVM
  - Como comunicar a JVM do JUnit com a do JPF?
    - Mecanismo de escutas e eventos
- Criação do **cJUnit**, encapsulando o JPF + helpers
- Annotation criada: `@ConcurrentTest`

Solução Proposta

- Comparação dos resultados para diferentes *schedulings*:
  - **Sucesso**: nenhuma falha em nenhum teste para todas as *schedulings*
  - **Falha normal**: falhas consistentes em todas as *schedulings*
  - **Falha de concorrência**: deadlocks e falhas em algumas *schedulings*

Exemplo

```
public class ConcurrentCounter {
    int count = 0;

    @Override
    public int getNext() {
        int c;
        synchronized (this) { c = ++count; }
        return c;
    }
    public synchronized int getCurrent() {
        return count;
    }
}
```

```
public class ConcurrentCounterTest {
    int count = 0;
    ConcurrentCounter counter;

    @Before
    public void setup() {
        counter = new ConcurrentCounter();
    }

    @ConcurrentTest
    public void testConcurrentGetNext() throws Throwable {
        Thread t = new Thread() {
            public void run() {
                counter.getNext();
            }
        };
        t.start();
        counter.getNext();
        t.join();
        assertThat(counter.getCurrent(), equalTo(2));
    }
}
```

```
@ConcurrentTest(threadCount=2)
public void testConcurrentGetNext() throws Throwable {
    counter.getNext();
    TestBarrier.await();
    assertThat(counter.getCurrent(), equalTo(2));
}
```

Outro exemplo

```
@ConcurrentTest(threadGroup=1)
public void testConcurrentGetNextA() throws Throwable {
    counter.getNext();
    TestBarrier.await();
    assertThat(counter.getCurrent(), equalTo(2));
}

@ConcurrentTest(threadGroup=1)
public void testConcurrentGetNextB() throws Throwable {
    counter.getNext();
    TestBarrier.await();
}
```

Resultados

- Suíte de testes do Helgrind adaptada de C++ para Java (50 casos de teste)
- 14 casos de teste criados para testar deadlocks, atomicidade e violação de ordem de execução
- **Todos** problemas de concorrência foram detectados e reportados adequadamente
- **Nenhum** falso positivo

Conclusão

Pontos Positivos

- Artigo bem focado
- Deadlocks são sempre detectados e reportados como problema de concorrência
- Extende o JUnit → Familiaridade
- API simples
- Ferramenta open source (*https://github.com/szeder/cJUnit/*)

Conclusão

Pontos Negativos

- Report incorreto quando exceções equivalentes acontecem por motivos de concorrência
    - Resolvido com uso mais detalhado do JPF, a nível de bytecode

```
objeto.metodoComRetorno().metodo()
```

Conclusão

Pontos Negativos

- Overhead do JPF
- Pouco detalhes de como o JPF funciona
- Não funciona com código nativo

Conclusão

- JUnit e TesteNG possuem mecanismos de rodar testes em paralelo, mas o motivo é otimizar o tempo de execução da suíte de testes (e não encontrar problemas de concorrência)
- cJUnit altera o scheduling de threads, "garimpando" o código buscando por bugs vindos da execução multi-threaded
- **Usaríamos a solução!**

Conclusão

Sugestões de Melhorias

- Realizar testes em projetos reais para medir o quão fácil seria a adoção, além de medir o desempenho
- Tratar caso de exceções equivalentes que acontecem por motivos de concorrência
- Mais detalhes sobre o JPF

Referências

- G. Szeder. 2009. **Unit testing for multi-threaded Java programs**. In *Proceedings of the 7th Workshop on Parallel and Distributed Systems: Testing, Analysis, and Debugging* (PADTAD '09). ACM, New York, NY, USA, Article 4, 8 pages. http://doi.acm.org/10.1145/1639622.1639626

Perguntas?