Universidade Federal de Pernambuco

Projeto de Monografia

Tratamento de Exceções Concorrentes na Linguagem Java

Autor: Wellington de Oliveira Júnior

Orientador: Fernando José Castor de Lima Filho

Recife

Julho - 2013

Sumário

Sumário 2

Capítulo 1 – Apresentação do projeto 3

1.1: Introdução 3

1.2: Objeto de Estudo 4

1.3: Problema 4

1.4: Objetivo 5

1.4.1: Geral 5

1.4.2: Específicos 5

Capítulo 2 – Justificativa 6

Capítulo 3 – Referencial Teórico 7

Capítulo 4 – Metodologia 8

Capítulo 5 – Cronograma 9

Capítulo 6 – Bibliografia 10

Capítulo 1 – Apresentação do projeto

1.1: Introdução

Threads são fluxos de execução de um código derivados de um fluxo principal, tem como objetivo paralelizar a execução de determinado código e com isso dividir tarefas para fazer melhor uso dos recursos, tanto no caso onde haja mais de um núcleo de processamento quanto no caso em que há uma determinada ação que apresenta um grande tempo de resposta. Ao chavear a execução de threads se obtém um uso ótimo dos recursos o que resulta numa melhor percepção de velocidade e desempenho.

Java fornece suporte extensivo à construção de programas concorrentes e paralelos. A linguagem permite a criação de threads e o gerenciamento de threads, além de vários mecanismos para controlar o acesso de threads a recursos compartilhados, como de variáveis atómicas, monitores e locks. A linguagem também implementa, através de bibliotecas, construções de mais alto nível, como executors, barreiras de sincronização e pools de threads. Outra característica importante de Java é o seu mecanismo de tratamento de exceções. Este permite que erros que aconteçam durante a execução de um método possam ser tratados dentro do código do próprio método ou lançados para qualquer método que venha a chamá-lo.

Embora Java forneça suporte de alto nível tanto à programação concorrente quanto ao tratamento de erros, a linguagem não integra essas funcionalidades de maneira satisfatória. Em particular, quando uma thread executa um método e este produz uma exceção esta pode fazer com que a thread seja finalizada sem notificar nenhuma outra thread que possa, porventura, depender da que falhou. Essa situação pode levar a diversos problemas, como deadlocks, corrupção do estado de variáveis e não-liberação de recursos previamente alocados. Esta situação está em conflito com os objetivos dos projetistas de Java , que ambicionavam desenvolver

uma linguagem ao mesmo tempo concorrente e altamente robusta. Apesar disso, não é surpreendente. Ainda há muito poucos trabalhos apresentando propostas para integrar tratamento de erros e construções para programação concorrente/paralela de forma prática. Algumas das propostas existentes focam em modelos puramente teóricos enquanto outras baseiam-se principalmente em aplicações de tempo real, muito diferentes das aplicações que são e serão executadas em máquinas multi-núcleo.

1.2: Objeto de Estudo

O tratamento de exceções da linguagem de programação Java. Para validar todo o estudo desenvolvido, experimentos devem ser realizados com desenvolvedores que utilizem esta linguagem para comprovar, ou refutar, a praticidade da solução proposta para a problemática das exceções levantadas em uma aplicação concorrente.

1.3: Problema

Exceções levantadas durante o desenvolvimento de aplicações concorrentes na linguagem Java não recebem nenhum tipo de tratamento nativo na linguagem, isto resulta em diversos problemas entre eles: o desinteresse dos programados em utilizar threads para o desenvolvimento de suas aplicações; a dificuldade de rastrear erros decorrentes da utilização de threads; a falta de isometria quanto as exceções que são levantadas num programa sequencial e um programa concorrente.

1.4: Objetivo

1.4.1: Geral

Propor uma solução para os problemas que resultam da criação de threads e sua utilização. Através da ferramenta desenvolvida é possível abstrair uma parte da problemática de se lidar com as exceções que são lançadas na execução das threads isso sem impor ao desenvolvedor dificuldades e ou grandes diferenças na sua maneira de programar habitual.

1.4.2: Específicos

Viabilizar a utilização de threads no desenvolvimento de aplicações na linguagem Java.

Capítulo 2 – Justificativa

Nota-se uma tendencia acentuada da utilização de processadores multinúcleo nos últimos anos para diversos tipos de aparelhos eletrônicos, como computadores de mesa, notebooks ou smartphones. Para fazer melhor uso da capacidade de vários processadores a uma necessidade da utilização de multiplas Threads e é notado que a manipulação destas é notavelmente complicada para os desenvolvedores.

A linguagem de programação Java é uma fortemente usada para aplicações em todos os meio dispositivos computacionais atuais, destacando-se a sua presença na plataforma móvel Android da Google que tem hoje vasta maioria de utilização em smartphones e é uma linguagem que apresenta um desenvolvimento continuo e com uma base gigantesca de desenvolvedores, contribuindo para sua escolha como linguagem de estudo.

Apesar das várias abordagens já presentes na linguagem Java, quando os desenvolvedores utilização threads eles o fazem da maneira mais tradicional e não utilizando recursos mais avançados como Thread Pools.

Uma vez aceita a necessidade da implementação de múltiplas threads, a forte presença da linguagem Java no desenvolvimento de aplicações e a falta de aceitação dos desenvolvedores em utilizar soluções presentes na linguagem para o problema apresentado, encontra-se a necessidade de apresentar uma nova solução, o mais semelhante possível a como atualmente funciona a linguagem para evitar resistencia por parte dos desenvolvedores.

Capítulo 3 – Referencial Teórico

O mecanismo a ser desenvolvido dará ênfase aos vários problemas inerentes à construção de aplicações paralelas confiáveis e levará em consideração também as particularidades de Java. Por exemplo, threads em Java trabalham com objetos compartilhados, o que aumenta a probabilidade do estado do sistema ser corrompido devido a uma exceção. Já em Scala é mais fácil evitar esse problema porque os atores (elementos de execução similares a threads) comunicam-se apenas através de passagem de objetos imutáveis, sem estado compartilhado. Além disso, considerará questões recentes que surgiram tanto na literatura acadêmica [11, 12] quanto na indústria [13] sobre o projeto de mecanismos de tratamento de exceções, como o uso ou não de exceções verificadas e o uso de especificações locais ou globais para exceções e seus tratadores.

Há exemplos de novos mecanismos de tratamento de exceções que buscam sanar os problemas das abordagens clássicas, como o o AtomicBox [4] ou o FailBox [5] ou até mesmo a abordagem utilizada pela Microsoft na linguagem C# [9] mas em Java ainda há uma ausência de qualquer tipo de solução apresentada buscando solucionar este problema.

Algumas questões importantes precisam ser respondidas durante o projeto do mecanismo proposto. Exemplos de tais perguntas incluem como as threads podem ser agrupadas em contextos de tratamento de exceções, para permitir que uma mesma exceção, em um mesmo contexto, possa ser tratada sempre da mesma forma? Adicionalmente, tais deve ser possível propagar exceções entre esses contextos e associar às exceções propagadas informações de seus stack traces. Que tipo de garantia o mecanismo de tratamento de exceções pode dar sobre a consistência do estado do sistema, dado que um erro ocorreu? Um princípio que guiará todo o projeto deste novo mecanismo é a imposição da menor sobrecarga possível a programadores habituados à linguagem Java

Capítulo 4 – Metodologia

A princípio, o mecanismo proposto será implementado como uma extensão de um dos compiladores existentes para a linguagem Java. Há várias opções de código aberto que podem ser usadas para este fim, como o javac, o GCJ e o compilador utilizado pela plataforma Eclipse de desenvolvimento. Uma abordagem alternativa é construir um pré-processador que transforme programas escritos em Java estendida com construções para tratamento concorrente de exceções em Java "pura". Esta última abordagem é mais simples, do ponto de vista de compilação, mas pode criar complicações em tempo de execução, tanto em termos de implementação quanto de eficiência. As duas abordagens serão avaliadas com cuidado antes que seja definido qual será empregada. Nesta etapa, é possível que o mecanismo precise ser adaptado para considerar as particularidades da abordagem de implementação escolhida.

Depois de implementado e testado, o mecanismo deverá ser então validado por programados, inicialmente no desenvolvimento de aplicações extremamente simples e evoluindo para o desenvolvimento de aplicações mais complexas, para verificar a sua real utilidade na prática, que é um dos objetivos do desenvolvimento do projeto.

Com os dados coletados com a utilização no desenvolvimento de aplicações por programadores, revisitaria-se o mecanismo, fazendo adaptações no seu sistema de acordo com as necessidades apresentadas pelos desenvolvedores na sua utilização, buscando assim um sistema que resolva os problemas apresentados e seja de fácil utilização.

Capítulo 5 – Cronograma

O cronograma a seguir busca sintetisar as etapas do desenvolvimento do projeto, buscando-se antecipar o seu progresso e tendo em mente a elasticidade das etapas caso seja necessário.

A1 – Levantamento Bibliográfico.
A2 – Reuniões com o Orientador.
A3 – Projeto do Novo Mecanismo de Tratamento de Exceções.
A4 – Implementação do Mecanismo Proposto.
A5 – Escrita do Relatório Final

Ano	2012				
Mês	5	6	7	8	9
A1	**X**	**X**			
A2		**X**	**X**	**X**	**X**
A3			**X**		
A4			**X**	**X**	
A5				**X**	**X**

Tabela 1 : Cronograma de Atividades

Capítulo 6 – Bibliografia

[1] James Gosling. The java language environment. Section 1.2: Design Goals of Java. Technical report, Sun Microsystems, 1996. Adress: http://java.sun.com/docs/white/langenv/Intro.doc2.html. Last visit: July 1st 2012

[2] Avelino Zorzo, Brian Randell, and Alexander Romanovsky. Tla specification of a mechanism for concurrent exception handling. In Concurrency in Dependable Computing, chapter 3, pages 41–59. Kluwer Academic Plublishers, 2002.

[3] Fernando Castor, Alexander Romanovsky, and Cecília M. F. Rubira. Improving reliability of cooperative concurrent systems with exception flow analysis. Journal of Systems and Software, 82(5):874–890, 2009.

[4] Derin Harmanci, Vincent Gramoli and Pascal Felber. Atomic Boxes: Coordinated Exception Handling with Transactional Memory. Adress: http://lpd.epfl.ch/gramoli/doc/pubs/ECOOP2011-preprint.pdf. Last visit: July 1st 2012

[5] Jie Xu, Brian Randell, Alexander Romanovsky. Fault Tolerance in Concurrent Object-Oriented Software through Coordinated Error Recovery. Adress: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.66.3758&rep=rep1&type=pdf Last visit: July 1st 2012

[5] Bart Jacobs and Frank Piessens. Failboxes: Provably Safe Exception Handling. Adress: https://lirias.kuleuven.be/bitstream/123456789/221909/2/failboxes.pdf Last visit: July 1st 2012

[6] Eclipse Java development tools (JDT). Adress: http://www.eclipse.org/jdt/ Last visit: July 1st 2012

[7] OpenJDK. Address: http://openjdk.java.net/. Last visit: July 1st 2012

[8] JastAddJ: The JastAdd Extensible Java Compiler. Adress: http://jastadd.org/web/jastaddj/ Last visit: July 1st 2012

[9] Parallel Tasks. Adress: http://msdn.microsoft.com/en-us/library/FF963549.aspx Last visit: July 1st 2012

[10] Nélio Cacho, Fernando Castor Filho, Alessandro Garcia, and Eduardo Figueiredo. Ejflow: taming exceptional control flows in aspect-oriented programming. In Proceedings of the 7th ACM Conference on Aspect-Oriented Software Development, pages 72–83, Brussels, Belgium, March 2008.

[11] Bart Jacobs and Frank Piessens. Failboxes: Provably safe exception handling. In Proceedings of the 23rd European Conference on Object-Oriented Programming, volume 5653 of LNCS, pages 470–494, Genoa, Italy, 2009. Springer.

[12] Brian Goetz. Java theory and practice: The exceptions debate, May 2004. Address: http://www.ibm.com/developerworks/java/library/j-jtp05254.html. Last visit: July 1st 2012.