Técnicas: Tratamento de Exceções, Depuração e Logging

Tratamento de Exceções

Usar Checked Exceptions ou Unchecked Exceptions?

- Java possui Checked Exceptions que estendem java.lang.Exception
  - O compilador força o programador a capturar tais exceções
  - Elas precisam fazer parte da assinatura dos métodos
- Java também possui Unchecked Exceptions que estendem java.lang.RuntimeException
  - Essas exceções não precisam ser capturadas (mas podem sê-lo)
- A questão é: em que situação você deveria usar cada tipo de exceção?
- Há gente que advoga usar Checked Exceptions e outros que advogam o contrário
  - Você vai ter que decidir, em cada situação
- Vamos examinar os argumentos de cada lado

A favor de usar Checked Exceptions

- Deixar que o compilador verificar todos os retornos de métodos, tanto os retornos normais (return) e os erros (throw) força o programador a tratar erros e aumenta a qualidade do código
- Exceções informam o chamador sobre algo que ele *tem* que tratar
- Lançar unchecked exceptions é coisa de programador preguiçoso que não quer lidar com as situações e deixa o código frágil
- Quando faz sentido nã documentar o que você faz? Quase nunca. É a mesma coisa com o uso de checked exceptions
- Resumindo: use Checked Exceptions sempre
- Muitos especialistas acreditavam nisso mas muitos estão mudando de idéia
  - Vamos ver a argumentação do outro lado
  - Exemplo: http://www.mindview.net/Etc/Discussions/CheckedExceptions

A favor de usar Unchecked Exceptions

- Checked Exceptions geram código demais (catch em tudo que é lugar) quando, na realidade, a maioria dos eeros é fatal e não o que fazer sobre o problema
- O código fica muito mais difícil de entender e o código adicional não traz muitas vantagens (porque a maioria das exceções é do tipo "erro fatal")
- Muitos empacotamentos (wrapping) de exceções
  - Para manter as interfaces de métodos limpos, menos exceções são usadas nas assinaturas e, quando há exceções de baixo nível lançadas, tem que usar wrapping de exceções

(uma exceção dentro de outra)
    - Isso é permitido por Java (uma exceção pode ter uma causa - uma outra exceção)
    - Mas isso gera código demais, com poucos benefícios reais no fim das contas, a não ser que informação adicional útil seja adicionada na exceção
- Assinaturas de métodos ficam frágeis com Checked Exceptions
    - Se uma exceção nova for adicionada a um método, todos os chamadores terão que ser alterados
    - Lembre que cada nova exceção é essencialmente um novo valor sendo retornado pelo método
- O fato de usar Unchecked Exceptions não significa que não serão tratadas; significa que o programador (que escreve o código chamador) tem a escolha de tratar ou não num certo nível
    - Claro que o nível do topo vai capturar tudo pois nada deve ser lançado por main()

Recomendações de especialistas (Johnson) sobre Checked versus Unchecked

- Tem verdades dos dois lados
- Eis a opinião de um especialista (Rod Johnson)
- Se todos os chamadores devem obrigatoriamente tratar a exceção, usar uma Checked Exception
    - Ex. Método processaFatura() encontra um limite de crédito excedido
- Se uma minoria de chamadores terão que tratar da exceção, usar uma Unchecked Exception
    - Ex. Exceções JDBC que serão tratados apenas no nível adequado
- Se algo terrível ocorreu e não há como se recuperar, usar uma Unchecked Exception
    - Ex. Não foi possível obter uma conexão ao banco de dados
    - Deixe a exceção chegar ao topo para um erro fatal ser dado ao usuário
    - O nível de topo sempre vai logar tais casos
        - Lembre de fazer isso se o nível do topo for escrito por você
- Se você ainda não tem certeza o que fazer, usar uma Unchecked Exception
    - O chamador decide se quer tratar ou não
    - Não esqueça de documentar a exceção pois o compilador não vai verificar nada e se a programadora não souber que uma exceção pode ser lançada, ela não tem mais a escolha que queremos dar a ela
- Lembre que uma Unchecked Exception não capturada vai matar o thread
    - Isso é verdade em certas situações
    - Em J2EE, não há perigo. Motivo: não controlamos threads, o container faz isso e não há perigo pois o container captura as

exceções para proteger os threads

Práticas adicionais sobre exceções

- Divida as exceções em Business Exceptions que serão lançadas na façade principal do sistema e outras exceções que nada têm a ver com o business
    - Dessa forma, o código de tratamento de exceções sabe como melhor tratar os dois tipos de exceção
- Não use mensagens de exceções para diferenciar entre duas exceções no código
    - Use exceções diferentes herdando de uma mesma classe mãe
    - Mensagens deve ser usadas em logs ou mensagens para o usuário, não para o código diferenciar entre duas exceções
    - Use código de erros (armazenados como atributo da exceção) e arquivos de propriedade para dar mensagens ao usuário
        - Isso facilita também a localização do software para outra "locale"

Depuração e Logging

- A instrumentação de código é importante
- Logging permite acompanhar (rastrear) a execução da aplicação
- Logging tem muitos usos mas o mais importante é depurar
    - É melhor do que usando um depurador, principalmente porque é algo permanente
    - Lembre que uma sessão de depuração não entra em CVS!
    - Também, permite depurar aplicações reais que estão em produção
    - Depuradores nem sempre funcionam bem em ambientes distribuídos
- Deve-se usar um bom pacote de logging e não tentar fazer as coisas "na mão" em Java
    - Pacotes como log4j ou Java 1.4 Logging são muito melhores
    - Nunca mande log em System.err ou System.out
        - Essas coisas podem não existir ou podem ser muito pentas em servidores
        - Também, não há como ligar/desligar o log de forma conveniente
- O que um bom pacote de logging deve ter?
    - API simples de usar
    - A habilidade de configurar o que será logado e como *fora* do codigo
        - Deve ser possível alterar um arquivo de configuração do pacote de loggin para ligar ou desligar o logging por classe, por pacote, etc.
    - As mensagens de log devem ser divididas em prioridades (debug, error, info, ...) e deve ser possível escolher o limiar a partir do qual o log é feito
    - Deve ser possível configurar a formatação das mensagens de

log (com ou sem data, com ou sem nome da classe, com ou sem a linha do código onde foi feito o log, etc., etc.)
    - Deve ser possível escolher onde o log vai (console, arquivo, documento XML, servidor de erros, Windows event log, ...)
    - Deve haver bufferização da saída sendo usada de forma a minimizar o impacto de performance nas chamadas à API de logging
- Mesmo um código terminado e instalado, em produção, deve ter condição de gerar log se for assim desejado
    - Um código nào está pronto para a produção até que possa produzir boa informação de log
- Você pode usar log4j ou a API de logging package de Java 1.4
    - log4j é muito popular e um pouco mais poderoso
    - Mas a API do Java já é padrão, não precisa ser instalada

Exemplo com a API Java

```
// usar o nome da classe dá muita flexibilidade para dizer o que será logado
Logger logger = getLogger(getClass().getName());
// ...
logger.fine("Achei erro no elemento <" +
            elementNumber + ">: valor fora de faixa");
```

- A API Java oferece os seguintes níveis de prioridade:
    - SEVERE: falha séria. Frequentemente haverá uma exceção associada
    - CONFIG: mensagens geradas durante a configuração da aplicação
    - INFO: prioridade moderada: indica o que está sendo feito em vez de ser usado especificamente para depuração (ex. uma tarefa terminou)
    - FINE: informação de rastreamento: usado para debug
    - FINER: informação detalhada de rastreamento: usado para debug
    - FINEST: informação altamente detalhada de rastreamento: usado para debug
- Um exemplo típico do arquivo logging.properties

```
# Specify the handlers to create in the root logger
# (all loggers are children of the root logger)
# The following creates two handlers
handlers = java.util.logging.ConsoleHandler, java.util.logging.FileHandler

# Set the default logging level for the root logger
.level = ALL

# Set the default logging level for new ConsoleHandler instances
java.util.logging.ConsoleHandler.level = INFO

# Set the default logging level for new FileHandler instances
java.util.logging.FileHandler.level = ALL

# Set the default formatter for new ConsoleHandler instances
java.util.logging.ConsoleHandler.formatter = java.util.logging.SimpleFormatter
```

```
# Loggers
# ---------------------------------------
# Loggers are usually attached to packages.
# Here, the level for each package is specified.
# The global level is used by default, so levels
# specified here simply act as an override.
myapp.ui.level=ALL
myapp.business.level=CONFIG
myapp.data.level=SEVERE
```

Exemplo com a API log4j

- Eis uma classe típica

```
public class SageFileTopologyBuilder extends CHESFTopologyBuilder {
    static Logger logger = Logger.getLogger(SageFileTopologyBuilder.class
            .getName());
// ...
    public void buildTopology() throws IOException, TopogiggioException {
        rootRegion = new Region(Element.TYPE_REGION,
                Element.SUBTYPE_REGION_SYSTEM, "CHESF");
        logger.debug("buildTopology");
        // table processing order is important
        logger.debug("process INS");
        process_INS_Table(rootRegion);
        logger.debug("process EST");
        process_EST_Table(rootRegion);
        logger.debug("process PDS");
        process_PDS_Table(rootRegion);
        logger.debug("process LIG");
        process_LIG_Table(rootRegion);
        logger.debug("process CNC");
    }
    private void process_TR2_Table(Region rootRegion) throws IOException,
            TopogiggioException {
        List table = loadTR2Table();
        for (int i = 0; i < table.size(); i++) {
            ItemTR2 item = (ItemTR2) table.get(i);
            if (skipID(item.id)) {
                continue;
            }
            // avoid creating several transformers for parallel
            // transformers
            String id = canonicalTransformerName(item.id);
            logger.debug("adding 2-winding transformer <" + id + ">");
            if (Convert.getInstance().getElement(id) == null) {
                Substation substation = getSubstationFromStation(item.primary);
                if (substation == null) {
                    throw new TopogiggioException(
                            "TR2 com PRIM estranho (sem subestacao)(" + item.id
                                    + ")");
                }
                String nodeID1;
                String nodeID2;
                if (item.primary.equals(item.secondary)) {
                    // caso especial de transformadores ficticios
                    nodeID1 = getConnection(item.id, item.primary, 0);
                    nodeID2 = getConnection(item.id, item.secondary, 1);
                } else {
                    nodeID1 = getConnection(item.id, item.primary, 0);
                    nodeID2 = getConnection(item.id, item.secondary, 0);
```

```
                }
                if (nodeID1 == null || nodeID2 == null) {
                    throw new TopogiggioException("TR2 sem 2 conexoes("
                            + item.id + ")");
                }
                logger.debug("adding transformer, nodes " + nodeID1 + ", "
                        + nodeID2);
                Element newElement = substation.addTransformer(id, nodeID1,
                        nodeID2, "");
                aNewElement(newElement);
            }
        }
    }
}
```

- Eis o arquivo de configuração log4j

```
# Set root category priority to DEBUG and its only appender to A1.
log4j.rootCategory=DEBUG, A1

#log4j.appender.A1=org.apache.log4j.ConsoleAppender
#log4j.appender.A1.target=System.err
log4j.appender.A1=org.apache.log4j.FileAppender
log4j.appender.A1.file=topogiggio.log
log4j.appender.A1.append=false

# A1 uses PatternLayout.
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
log4j.appender.A1.layout.ConversionPattern=%-4r [%t] %-5p %c %x - %m%n

# Configure topogiggio log level here
#log4j.logger.topogiggio.Auditor=DEBUG
#log4j.logger.topogiggio.Convert=DEBUG
#log4j.logger.topogiggio.bo.AbstractElement=DEBUG
#log4j.logger.topogiggio.bo.ElementOneNode=DEBUG
#log4j.logger.topogiggio.bo.ElementTwoNodes=DEBUG
#log4j.logger.topogiggio.bo.Generator=DEBUG
#log4j.logger.topogiggio.bo.Line=DEBUG
#log4j.logger.topogiggio.bo.Node=DEBUG
#log4j.logger.topogiggio.bo.Region=DEBUG
#log4j.logger.topogiggio.bo.Substation=DEBUG
#log4j.logger.topogiggio.bo.Transformer=DEBUG
#log4j.logger.topogiggio.bo.Winding=DEBUG
#log4j.logger.topogiggio.bo=DEBUG
#log4j.logger.topogiggio.Convert=DEBUG
#log4j.logger.topogiggio.test.TestConvert=DEBUG
#log4j.logger.topogiggio.TopogiggioFacade=DEBUG
#log4j.logger.topogiggio.topology.MDRSageTopologyBuilder=DEBUG
#log4j.logger.topogiggio.topology.SageFileTopologyBuilder=DEBUG
#log4j.logger.topogiggio.visitor.AbstractVisitor=DEBUG
#log4j.logger.topogiggio.visitor.BusVisitor=DEBUG
#log4j.logger.topogiggio.visitor.classifiers.AbstractClassifierVisitor=DEBUG
#log4j.logger.topogiggio.visitor.GeneratorVisitor=DEBUG
#log4j.logger.topogiggio.visitor.HandleFictitiousLinesVisitor=DEBUG
#log4j.logger.topogiggio.visitor.HandleLinksVisitor=DEBUG
#log4j.logger.topogiggio.visitor.LineVisitor=DEBUG
#log4j.logger.topogiggio.visitor.ReactorVisitor=DEBUG
#log4j.logger.topogiggio.visitor.RemoveFictitiousLinesVisitor=DEBUG
#log4j.logger.topogiggio.visitor.ShortCircuitVisitor=DEBUG
#log4j.logger.topogiggio.visitor.ShortCircuitLinksVisitor=DEBUG
#log4j.logger.topogiggio.visitor.SubstationLinkVisitor=DEBUG
#log4j.logger.topogiggio.visitor.TransformerVisitor=DEBUG
```

```
#log4j.logger.topogiggio.visitor.VisitResult=DEBUG
#log4j.logger.topogiggio=DEBUG
log4j.logger.topogiggio=INFO
```

Logging e desempenho

- O efeito no desempenho é normalmente muito pequeno
- Uma exceção: quando gerar o string de log custa caro
    - É melhor verificar se o log realmente será feito

```
if(logger.isLoggable(Level.FINE)) {
   logger.fine("O estado do objeto eh: " + metodoQueCustaCaro());
}
```

    - Outra exceção: em código de produção, não use formatos de mensagens de log com o nome da classe, número da linha, etc. pois isso é gerado de forma muito lenta (examinando o stacktrace de exceções artificiais)

tecnicas-1 programa