

Obsługa sytuacji wyjątkowych w technologii JEE.

Tomasz.Skutnik@e-point.pl



Java Developers Day, 16 października 2009

Wprowadzenie

- **Sytuacje wyjątkowe w cyklu życia systemu**
- **Teoria i praktyka obsługi wyjątków w Javie**
- **Modele obsługi wyjątków w JEE**
- **Praktyka obsługi wyjątków w JEE**

Sytuacje wyjątkowe w cyklu życia systemu

- **Sytuacje wyjątkowe nie są wyjątkowe**
- **Zaakceptuj, że przyszłości nie da się przewidzieć**
- **Obsługa awarii przekłada się na satysfakcję klienta**
- **Zorganizuj proces obsługi awarii**

Sytuacje wyjątkowe w cyklu życia systemu

```
1 Caused by: pl.epoint.jdbc.RuntimeSQLException: org.jboss.util.NestedSQLException: Transaction is not active:
2   tx=TransactionImple < ac, BasicAction: -3f57f071:e7f1:4a41ff7c:4780f status: ActionStatus.ABORTED >;
3   - nested throwable: (javax.resource.ResourceException: Transaction is not active: tx=TransactionImple
4   < ac, BasicAction: -3f57f071:e7f1:4a41ff7c:4780f status: ActionStatus.ABORTED >)
5     at pl.epoint.jdbc.JdbcUtil.withDataSourceConnection(JdbcUtil.java)
6     at pl.epoint.jdbc.JdbcUtil.query(JdbcUtil.java)
7     ... 80 more
8 Caused by: org.jboss.util.NestedSQLException: Transaction is not active: tx=TransactionImple < ac, BasicAction:
9   -3f57f071:e7f1:4a41ff7c:4780f status: ActionStatus.ABORTED >; - nested throwable: (javax.resource.
10  ResourceException: Transaction is not active: tx=TransactionImple < ac, BasicAction: -3f57f071:e7f1
11  :4a41ff7c:4780f status: ActionStatus.ABORTED >)
12    at org.jboss.resource.adapter.jdbc WrapperDataSource.getConnection(WrapperDataSource.java:94)
13    ... 100 more
14 Caused by: javax.resource.ResourceException: Transaction is not active: tx=TransactionImple < ac, BasicAction:
15   -3f57f071:e7f1:4a41ff7c:4780f status: ActionStatus.ABORTED >
16    at org.jboss.resource.connectionmanager.TxConnectionManager.getManagedConnection(TxConnectionManager...
17    at org.jboss.resource.connectionmanager.BaseConnectionManager2.allocateConnection(BaseConnectionMana...
18    ... 100 more
```

```
Transaction is not active: tx=TransactionImple < ac,
BasicAction: -3f57f071:e7f1:4a41ff7c:4780f status:
ActionStatus.ABORTED >
```

Sytuacje wyjątkowe w cyklu życia systemu

- **Sytuacje wyjątkowe nie są wyjątkowe**
- **Zaakceptuj, że przyszłości nie da się przewidzieć**
- **Obsługa awarii przekłada się na satysfakcję klienta**
- **Zorganizuj proces obsługi awarii**

Elementy polityki obsługi awarii

- **Monitoring**
- **Strony awaryjne i strony z błędami**
- **Narzędzia dla operatorów**
- **Narzędzia do migawek z logów i monitoringu**
- **Analizy „post mortem”**

Checked Exceptions Are Evil

Checked Exceptions Are Evil

Checked Exceptions Are Evil

- **Nie sprawdzają się w praktyce**
- **W 95% sytuacji nie jest możliwe kontynuowanie działania**
- **Decyzja kiedy można kontynuować działanie nie należy do kodu rzucającego wyjątek**
- **Zwiększają stres i uczą złych nawyków**
- **Niekompatybilne z niektórymi technikami programowania**

Antywzorce — część 1

- **Programowanie za pomocą wyjątków**
- `break`, `continue` i `return` w `finally`
- **Łapanie `Error` i `Throwable`**
- **Zjadanie wyjątków**
- **Wielokrotne logowanie wyjątków**

Antywzorce — część 1

```
1 public static void antipattern1() throws Exception {
2     String[] s = new String[] { "ala", "ma", "kota" };
3     try {
4         int i = 0;
5         while (true) {
6             someMethod(s[i]);
7             i = i + 1;
8         }
9     } catch (IndexOutOfBoundsException e) {
10         // end of loop
11     }
```

Antywzorce — część 1

- **Programowanie za pomocą wyjątków**
- `break`, `continue` i `return` w `finally`
- **Łapanie** `Error` i `Throwable`
- **Zjadanie wyjątków**
- **Wielokrotne logowanie wyjątków**

Antywzorce — część 1

```
1   String result = null;
2   while (true) {
3       try {
4           result = "RESULT";
5           throw new NullPointerException("Throw attempt");
6       } finally {
7           break; // oops!
8       }
9   }
10  return result;
```

Antywzorce — część 1

- **Programowanie za pomocą wyjątków**
- `break`, `continue` i `return` w `finally`
- **Łapanie** `Error` i `Throwable`
- **Zjadanie wyjątków**
- **Wielokrotne logowanie wyjątków**

Antywzorce — część 1

```
1  try {  
2      // ... some code  
3  } catch (Error e) {  
4      // ... some error handling  
5  }
```

```
1  try {  
2      // ... some code  
3  } catch (Throwable t) {  
4      // ... some error handling  
5  }
```

Antywzorce — część 1

- **Programowanie za pomocą wyjątków**
- `break`, `continue` i `return` w `finally`
- **Łapanie `Error` i `Throwable`**
- **Zjadanie wyjątków**
- **Wielokrotne logowanie wyjątków**

Antywzorce — część 1

```
1     try {  
2         // ... some code  
3     } catch (Exception e) {  
4     }
```

```
1     try {  
2         // ... some code  
3     } catch (Exception e) {  
4         e.printStackTrace();  
5     }
```

```
1     try {  
2         // ... some code  
3     } catch (Exception e) {  
4         log.log(Level.SEVERE, "Error!", e);  
5     }
```

Antywzorce — część 1

- **Programowanie za pomocą wyjątków**
- `break`, `continue` i `return` w `finally`
- **Łapanie `Error` i `Throwable`**
- **Zjadanie wyjątków**
- **Wielokrotne logowanie wyjątków**

Antywzorce — część 1

```
1    try {  
2        // ... some code  
3    } catch (Exception e) {  
4        e.printStackTrace();  
5        throw e;  
6    }
```

```
1    try {  
2        // ... some code  
3    } catch (Exception e) {  
4        log.log(Level.SEVERE, "Error!", e);  
5        throw new RuntimeException(e);  
6    }
```

Antywzorce — część 2

- **Zwracanie „standardowej” wartości**
- **Niszczanie stosu wykonania**
- **Wieloliniowy zapis do logów**
- **`throws` w metodzie która nie rzuca wyjątków**
- **Rzucanie wyjątków przy interaktywnym wczytywaniu danych wejściowych**

Antywzorce — część 2

```
1  try {  
2      // ... some code  
3  } catch (Exception e) {  
4      return "default";  
5      // return "";  
6      // return null;  
7      // return 0;  
8      // return -1;  
9  }
```

Antywzorce — część 2

- **Zwracanie „standardowej” wartości**
- **Niszczanie stosu wykonania**
- **Wieloliniowy zapis do logów**
- **`throws` w metodzie która nie rzuca wyjątków**
- **Rzucanie wyjątków przy interaktywnym wczytywaniu danych wejściowych**

Antywzorce — część 2

```
1  try {  
2      // ... some code  
3  } catch (IllegalArgumentException e) {  
4      throw new Exception(e.getMessage());  
5  }
```

```
1  try {  
2      // ... some code  
3  } catch (Exception e) {  
4      // ... some error handling  
5      log.log(Level.SEVERE, "Message part 1...");  
6      log.log(Level.SEVERE, "... message part 2.", e);  
7  }
```

Antywzorce — część 2

- **Zwracanie „standardowej” wartości**
- **Niszczanie stosu wykonania**
- **Wieloliniowy zapis do logów**
- **`throws` w metodzie która nie rzuca wyjątków**
- **Rzucanie wyjątków przy interaktywnym wczytywaniu danych wejściowych**

Antywzorce — część 2

```
1 public static void someMethod(String name)
2     throws Exception
3 {
4     System.out.println("Hi, " + name);
5     // ... some more code that
6     // doesn't throw exception
7 }
```

```
1 String userName = null;
2 // ... some code parsing user input
3 // from HTTP basic authentication header
4 if (userName == null) {
5     throw new IllegalArgumentException("Bad HTTP " +
6         "basic authentication header");
7 }
```

Dobre praktyki — część 1

- **Najpierw przywróć system do działania**
- **Szybko przerywaj działanie — „fail fast”**
- **Nie przewiduj przyszłości — szybko reaguj**
- **Obsługa wyjątków maksymalnie uproszczona**
- **Zamontuj bezpieczniki w punktach integracji**
- **Zatrudnij koronera: przeglądy logów, analizy „post-mortem”, itp.**

Dobre praktyki — część 2

- Anatomia wyjątku: co, gdzie i dłaczego
- Loguj wszystkie istotne dane: nagłówki HTTP, ciasteczka, zawartość sesji, itp.
- Log powinien się łatwo parsować
- Logowanie to nie wszystko — zainstaluj zewnętrzny monitoring
- Zmień szablon klauzuli catch w IDE

Zalecany sposób postępowania z wyjątkami

- 1 Centralizuj obsługę błędów
- 2 Naostrz nóż — zaprogramuj narzędzia wykonujące standardowe operacje
- 3 Opakowuj wyjątki w `RuntimeException`
- 4 Jeśli musisz obsłużyć sytuację wyjątkową, wprowadź specyficzny wyjątek „unchecked”
- 5 Podczas obsługi rozpakuj wyjątek
- 6 Osiągnięty efekt:
 - Spójny sposób korzystania z bibliotek i zasobów
 - Prostszy, łatwiejszy w czytaniu i utrzymaniu kod
 - `catch` oznacza: „Uwaga! Coś ważnego!”

Serwlety

- Zaimplementuj „transakcyjną” sesję (filtr)
- Obsługa wyjątku odbywa się przez „forward”
żądania HTTP bez filtrów
- Transakcje i pamięć podręczna
- ServletException nie wypisuje stosu
wykonania przyczyny wyjątku
- Strony z komunikatami o błędach
 - Estetycznie i spójne z szatą graficzną systemu
 - Identyfikator sytuacji wyjątkowej
 - Powiedz użytkownikowi co ma teraz zrobić
 - I dodatkowo zrób to za niego

JDBC

- **Uważaj na podwójne błędy podczas zwalniania zasobów**
- `SQLException` **nie wypisuje w komunikacie** `SQLState` **ani** `errorCode`
- **Zrób sobie nóż — zakoduj standardową interakcję w postaci narzędzia**
- **Ponawianie transakcji po zakleszczeniu (deadlock)**
- **Naruszenie więzów integralności jako funkcja systemu (duplikacja loginu użytkownika)**

JDBC — narzędzia 1

```
1 public <T> T withDataSourceConnection(ConnectionOp<T> operation) {
2     Connection connection = null;
3     try {
4         connection = dataSource.getConnection();
5     } catch (SQLException e) {
6         throw new RuntimeException(e);
7     }
8     RuntimeException exception = null;
9     T result = null;
10    try {
11        result = operation.invoke(connection);
12    } catch (SQLException e) {
13        exception = new RuntimeException(e);
14    } catch (RuntimeException e) {
15        exception = e;
16    } finally {
17        if (connection != null) {
18            try {
19                connection.close();
20            } catch (SQLException e) {
21                if (exception != null) {
22                    exception = new RuntimeException(exception, e);
23                } else {
24                    exception = new RuntimeException(e);
25                }
26            }
27        }
28    }
29    if (exception != null) { throw exception; }
30    return result;
31 }
```

JDBC — narzędzia 2

```
1 public List<RH> invoke(Connection con) throws SQLException {
2     List<RH> result = new ArrayList<RH>();
3     PreparedStatement preparedStatement = null;
4     ResultSet resultSet = null;
5     RuntimeException exception = null;
6     try {
7         preparedStatement = con.prepareStatement(query);
8         parameterHandler.bindParameters(preparedStatement, parameters);
9         resultSet = preparedStatement.executeQuery();
10        while(resultSet.next()) {
11            result.add(rowHandler.getObject(resultSet));
12        }
13    } catch (SQLException e) {
14        exception = new RuntimeException(e);
15    } catch (RuntimeException e) {
16        exception = e;
17    } finally {
18        if (resultSet != null) {
19            try {
20                resultSet.close();
21            } catch (SQLException e) {
22                if (exception != null) {
23                    exception = new RuntimeException(exception, e);
24                } else {
25                    exception = new RuntimeException(e);
26                }
27            }
28        }
29        // continued on next slide...
```

JDBC — narzędzia 3

```
1      // ... continued from previous slide
2      if (preparedStatement != null) {
3          try {
4              preparedStatement.close();
5          } catch (SQLException e) {
6              if (exception != null) {
7                  exception = new RuntimeException(exception, e);
8              } else {
9                  exception = new RuntimeException(e);
10             }
11         }
12     }
13 }
14 if (exception != null) {
15     throw exception;
16 }
17 return result;
18 }
```

JDBC — narzędzia 4

```
1 package pl.epoint.util.jdbc;
2
3 import java.sql.SQLException;
4
5 public class RuntimeException extends RuntimeException {
6
7     private static final long serialVersionUID = -4702776719908322845L;
8
9     public RuntimeException(SQLException cause) {
10         super(cause);
11     }
12
13     public SQLException getSQLException() {
14         return (SQLException) getCause();
15     }
16
17     @Override
18     public String getMessage() {
19         return "SQLSTATE: " + getSQLException().getSQLState()
20             + "; ERROR CODE: " + getSQLException().getErrorCode()
21             + "; " + super.getMessage();
22     }
23 }
```

MDB

- **Poznaj swój serwer aplikacji**
- **Używaj transakcji zarządzanych przez kontener**
- **Zaprojektuj kolejkę na komunikaty, których obsługa zakończyła się błędem**
- **Zaprojektuj system odporny na duplikację komunikatów**
- **Programując nie popełnij żadnego błędu ;-)**

EJB

- **Skomplikowany i trudny w użyciu model obsługi wyjątków**
- **Trudna w implementacji i utrzymaniu korelacja wywołań z żądaniami HTTP**
- **Pamiętaj o wycofaniu transakcji (rollback) przy wyjątkach aplikacyjnych**
- **Umieść klasy wyjątków po stronie klienta komponentów EJB**

Zapraszam do dyskusji

Czy są jakieś pytania?

`Tomasz.Skutnik@e-point.pl`