# 一种 Java 异常处理框架的设计与实现①

## Design and Implementation of Exception – handling Framework in Java

杨龙飞　李 华
（光电技术与智能控制教育部重点实验室（兰州交通大学）　甘肃兰州 730070）

**摘要**：对目前 Java 异常处理中所存在的问题进行了分析，提出了一种具有可扩充性和重用性的异常处理框架，该框架实现了异常处理和系统的商业逻辑分离，从而当异常处理的方法改变时，系统的业务逻辑不受影响，增强了系统的可维护性和可靠性。

**关键词**：异常处理框架　可扩充性　重用性

## 1　引言

Java 提供了强制性的异常处理机制，程序员必须撰写可适当处理异常的程序代码，从而错误处理也更加容易。但是，从另一方面看，程序员这种撰写异常的自由性，正是异常处理中的最大隐患。在一个实际 Java 软件项目中，如果每个程序员都可以任意定义，分类和处理异常，那么后果是显而易见的。首先，会造成程序代码的臃肿，难以维护。其次，代码中可能会有多余的实现，即同一类型的错误有不同的表示，这导致了处理的复杂化。

为了解决上述问题，应制定一套有效的异常处理策略，规范异常处理。而通过面向框架的编程思想，把异常处理封装在一个框架中，不仅可以达到统一协调程序员对异常处理的目的，又能实现针对异常处理领域的软件复用。

基于以上异常处理中的问题和使用框架的益处，建立异常处理框架。

## 2　异常处理框架的设计与实现

### 2.1　异常处理框架的结构

框架的结构如图 1 所示。该框架分成 5 大块。

（1）异常类封装了一个特定的错误。

（2）异常处理器封装了异常处理方法，其中含有一个缺省异常处理器，它的存在，是为了保证程序能从未经定义的异常中回复过来，提醒用户新增异常类别和相应的异常处理器。

（3）XML 配置文件描述了异常类与异常处理器的映射关系。

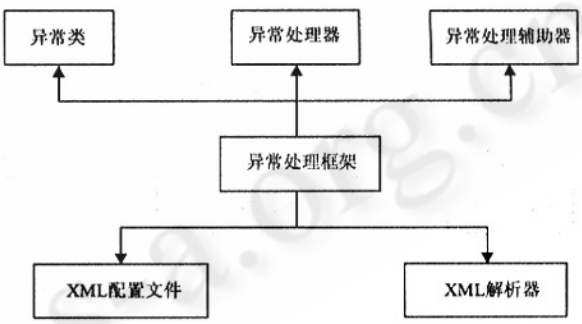


图 1　框架结构

（4）XML 解析器用来对 XML 配置文件的进行解析。

（5）异常处理辅助器用来获取实际的异常处理器。

### 2.2　异常处理框架的处理流程

在异常处理辅助器中，先产生一个 XML 解析器实例，通过该实例把 XML 配置文件中的异常类和对应的异常处理器读出存储在一个 HashMap 中，异常类为 keys（键值），异常处理器为 values（实值）。当捕获异常时，以异常型别为参数，遍历前述 HashMap，找到与

① 甘肃省自然科学基金项目（3ZS042 – B25 – 039）

该异常型别相对应的异常处理器，然后通过 Java 的 Reflection(映像)机制，调用异常处理器的处理方法。

### 2.3 核心接口和类的设计

在 Java 编程中，Java 的接口为构建可重用的系统提供了很大的支持，因而为了使框架得到最大程度的复用，利用接口的"定义和实现分离"特性，在该框架中定义了两个接口，一个是每个用户自定义异常类所必须实现的接口 BaseExInterface，而封装了异常详细信息的类 ExDetails 与接口 BaseExInterface 是一种依赖关系，即接口 BaseExInterface 的一个方法的参量是类 ExDetails 的实例。同理，异常处理辅助器 ExHandlerHelper 与接口 BaseExInterface 也是依赖关系。另一个是每个用户自定义异常处理器所实现的接口 ExHandlerInterface，该接口封装了处理异常的方法，该方法的参量是实现接口 BaseExInterface 的类的实例。XML 解析器 XMLConfigReader 与 ExHandlerHelper 是一种关联关系，即在 ExHandlerHelper 中，含有一个 XMLConfigReader 的实例变量。这些接口和类的关系如图 2 所示。

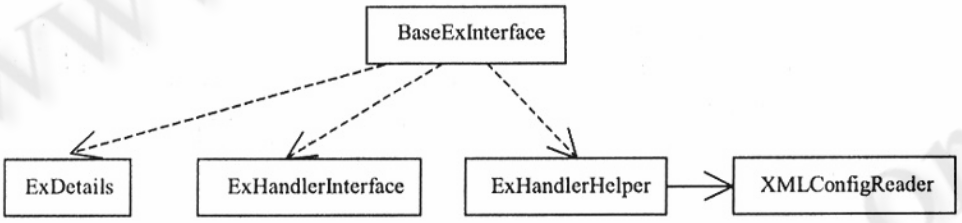


图 2 类与接口关系图

由于框架充分利用了面向对象的编程思想，提供了接口 BaseExInterface 和 ExHandlerInterface，所以接口的实现方式，即异常类或异常处理器的方式改变时，系统照样可以运行，这显然增强了系统的可扩充性。

### 2.4 XML 配置文件的制定

利用 XML 非常简易的数据描述方式，定义好异常发生的地方，将每个异常类与相应的异常处理器进行映射。XML 配置文件的内容如下所示，xmlconfig 是该 XML 文件的根元素，item 标记的是 exception/handler 对的个数，exception 元素标记异常的类别，handler 标记对应于 exception 的异常处理器。

```
<? xml version = "1.0" ? >
<xmlconfig >
    <item id = "0" >
        < exception > com. exception. DefaultEx </exception >
        < handler > com. exceptionhandling. DefaultExHandler </handler >
    </item >
    <item id = "1" >
        < exception > com. exception. MyEx </ exception >
        < handler > com. exceptionhandling. MyExHandler </ handler >
    </item >
    ……
</xmlconfig >
```

将异常处理定义在配置文件中的好处是显而易见的。

首先，由于异常处理的全部信息都记录在配置文件中，整个文件就相当于一个完整的异常结构图。通过该文件，开发人员可以比较容易地了解到整个异常的结构和处理策略，在开发的过程中可以提高效率，在系统出现问题的时候也可以迅速找到问题的所在，降低维护的代价。

其次，当某一种异常的处理方法改变时，只需要修改 handler 标记的异常处理器中的处理方法，重新编译异常处理器即可。而在以前的系统中则需要将这个异常经过的类全部修改。这样做不仅给开发人员带来了额外的工作量，而且易于出错，增加了调试的难度，而且破坏了代码的一致性。

另外，把处理异常方法和捕获异常分离可以减少很多冗余代码，提高代码的可读性。在以前的系统中，虽然同一种异常的处理方法基本上一样，但却不得不在多个调用的地方写同样的代码。

## 3 异常处理框架实现的关键技术

### 3.1 基于 DOM 的 XML 解析器

有很多基于 Java 的开源的 XML 解析器，如 Apache

组织的 Xerces，Sun 公司的 JDom，以及近来流行的 dom4j，虽然它们功能强大，但是实现这些强大的功能的同时也要消耗等量的资源。DOM（Document Object Model，文档对象模型）API 是 W3C 组织定义的标准，它是读取、操纵 XML 文件的一种标准方法，它提供了一个具有完整文档并且易于理解的 XML 对象模型。虽然 DOM 在处理复杂的 XML 文件时，会引起系统资源不足，但由于本框架中的 XML 配置文件结构并不复杂，权衡存储量和 XML 文档的复杂性，所以只需编写一个基于 DOM 的 XML 解析器。

为了根据异常型别来定位异常处理器，在 XMLConfigReader 中，定义了一个 HashMap，以 XML 配置文件中的 item 元素为 keys，以另一个 HashMap 为 values，而在这个子 HashMap 中，它的 keys 为 item 的子元素 exception，它的 values 为 item 的子元素 handler。通过 getString( ) 方法，实现异常处理器的定位。关键代码如下：

```
public String getString(String key, String subkey) {
    String param_value = "";
    String defaultvalue = "com.exceptionhandling.DefaultExHandler";
    try {
      // 根据 item，得到存储 exception/handler 对的子 Hashtable
      HashMap submap = (HashMap) this.getObject(key);
      // 取得异常处理器的路径信息
      param_value = (String) submap.get(subkey);
      if ((param_value != null) && (param_value.length( ) > 0)) {
        // 返回异常处理器路径信息
        return param_value;
      }
    } catch (Exception e) {
      return defaultvalue;
    }
    return defaultvalue;
  }
```

### 3.2 基于 Reflection 机制的异常处理辅助器

所谓 Reflection，便是提供了"找出这些可用函数并传回函数名称"的一个机制。Java 提供了一个 java.lang.reflect 程序库，其中含有 Field，Method，Constructor 等 classes。这些型别的对象由 JVM 在执行期产生，用来代表未知 class 内的相应成员。然后，使用 invoke( ) 来调用相应于 Method 对象的函数。因此即使完全不认识某个对象，编译期无法获得其任何信息，任然可以在执行期找出完整的 class 信息。异常处理辅助器正是利用这种机制，先将从 XML 解析器中得到的异常处理器类加载，然后通过 invoke( ) 来调用该异常处理器中的处理异常的方法。异常处理辅助器的关键代码如下：

```
import java.lang.reflect.*;
// 在此导入其它相关的包，如异常类和异常处理器所在的包等
public class ExHandlerHelper {
    // 在此实例化 XMLConfigReader，调用 getString( ) 方法获得 path，定位异常处理器，
    // 如果捕获的异常在 XML 配置文件中没有描述，则先调用缺省的异常处理器，然
    // 后用户可对该异常在 XML 配置文件中重新配置。
    public static void handleException (BaseExInterface be) {
        Class c = null;
        Object obj = null;
      try {
        c = Class.forName (path); // path 为从 XML 配置文件中解析出的异常处理器// 路径，加载异常处理器
        obj = c.newInstance ( ); // 产生一个异常处理器的实例
      } catch (……) { …… } // 在此声明 Class.forName( ) 方法可能抛出的异常
      Method handleMethod; // 定义 Method 对象
      Object[ ] arguments = new Object[ ] {be}; // 定义 invoke 方法的参量
      try {
       // 取得 ExHandlerInterface 的 handleEx 方法句柄
        handleMethod = c.getMethod ("handleEx", BaseExInterface.class);
```

（下转第 70 页）

(上接第 67 页)

```
        // 调用异常处理器的 handleEx 方法
        handleMethod. invoke( obj, arguments);
        } catch (……) { ……} // 在此声明 invoke(
)方法可能抛出的异常
        }
}
```

系统捕获异常时,只需调用 ExHandlerHelper 的静态方法 handleException,由于用户自定义异常都实现 BaseExInterface 接口,所以将 handleException 的参量定义为 BaseExInterface 型别,通过多态和 Reflection 机制,实现处理异常方法的调用。由此,用户只需调用 ExHandlerHelper 类的 handleException 方法即可处理异常,从而简化了处理异常的复杂度。

## 4 结束语

框架强调的是软件设计的重用性和系统的可扩充性。本文提出的异常处理框架规范了 Java 软件开发中的异常处理,把异常处理封装在框架中,将异常处理与系统商业逻辑分离,用户更能集中力量于系统商业逻辑的实现上,同时用户可根据实际需要,对框架进行完善和扩充,从而缩短了软件系统的开发周期,提高了开发质量。

## 参考文献


1 Bruce Eckel,侯捷,Java 编程思想[M],北京 机械工业出版社,2002。

2 刘晓华等, J2EE 企业级开发[M],北京 电子工业出版社,2003。

3 Brett McLaughlin,孙兆林、汪东、王鹏, Java 与 XML [M],北京 中国电力出版社,2001。

4 Sun Microsystems, Inc. JavaTM 2 SDK, Standard Edition Documentation [EB/OL]. http:// java. sun. com/j2se/1.4.2/docs/index. html.

5 李非一, J2EE 应用异常模式的研究[J], 微机发展, 2004, 14(3): 41-43。