Sistemas en Tiempo Real

Grado en Ingeniería Electrónica

UNIDAD 3: Excepciones y manejo de excepciones

Capítulo 3 de la edición Inglés (2009)

Capítulo 6 de la edición en Español (2003)

Tutoría

1. Introducción
2. Gestión de las excepciones en lenguajes de tiempo real antiguos
3. Gestión de excepciones en la actualidad
4. Excepciones en Ada, Java y C
5. Bloques de recuperación y excepciones

Introducción

En este tema se verá cómo se puede realizar el manejo de excepciones (Excepción: la ocurrencia de un error).

Requisitos generales para cualquier mecanismo de manejo de excepciones:

- R1: Debe ser fácil de comprender y utilizar.
- R2: No debe dificultar la comprensión del funcionamiento normal del programa (sin errores).
- R3: Sólo supondrá sobrecarga en la ejecución cuando se maneja una excepción. Solamente será aceptable cierta pequeña sobrecarga cuando sea primordial la rapidez de recuperación al ocurrir una excepción.
- R4: Deberá permitir un tratamiento uniforme de las excepciones detectadas tanto por el entorno como por la propia aplicación.
- R5: Deberá permitir la programación de acciones de recuperación.

Gestión de las excepciones en lenguajes de tiempo real antiguos

Retorno de un valor inusual y banderas de estado (C, POSIX)

Una función puede devolver un valor inusual. Desde la función que la invoca, se analiza el valor devuelto y se actúa en consecuencia.

Existen métodos de control de excepciones en lenguajes tipo C implementados con valores de retorno inusuales, normalmente distintos de cero.

No cumplen R2, R3 ni R4: mecanismo complejo, sobrecarga la ejecución y depende del origen.

Bifurcación forzada (En lenguajes ensambladores)

Consiste en forzar saltos a diferentes puntos del programa cuando se detecta la excepción.

No cumplen R1, R2 ni R4: mecanismo complejo, complica el programa y el manejo depende de su origen.

Goto no local (RTL/2, C)

Una bifurcación forzada en lenguajes de alto nivel. Aunque es muy flexible, genera programas oscuros.

Además, como rompe el flujo del programa, es adecuado solo para errores irrecuperables.

En errores recuperables se pueden usar variables de procedimiento de error, para que el control del programa vuelva al punto donde se originó la excepción, aunque siguen complicando los programas.

No cumplen R1 y R2: mecanismo complejo, complica el programa.

Gestión de las excepciones en la actualidad

Actualmente se tiende a incluir funcionalidades de manejo de excepciones directamente en el lenguaje, lo que provee un mecanismo más estructurado.

- Se puede detectar el error desde el entorno o desde la propia aplicación.
- Las excepciones se pueden provocar de forma **síncrona** (respuesta inmediata) o **asíncrona** (un tiempo después de que se haya detectado el error).

Existen cuatro tipos de excepciones:

1. Detectada por el entorno y generada síncronamente, por ejemplo, una división por cero.
2. Detectada por la aplicación y generada síncronamente, por ejemplo, un fallo en un aserto.
3. Detectada por el entorno y generada asíncronamente, por ejemplo, un fallo de alimentación o un mecanismo de monitorización vital.
4. Detectada por la aplicación y generada asíncronamente, por ejemplo, cuando un proceso verifica una condición que ocasionará que un proceso no cumpla los plazos o no termine correctamente.

Las excepciones asíncronas también se denominan **señales**. Serán estudiadas más adelante.

Gestión de las excepciones en la actualidad

Existen diferentes formas de declarar las excepciones síncronas:

1. Mediante un nombre constante que necesita ser declarado explícitamente.
2. Mediante un objeto de cierto tipo que podrá o no ser declarado explícitamente.

Ada precisa que las excepciones se declaren como constantes, por ejemplo, las excepciones que puede generar el entorno de ejecución se declaran en el paquete **Standard**.

Java y C++ dan una visión de las excepciones más orientada al objeto.

En java las excepciones son instancias de una subclase de la clase **Throwable** y podrán ser lanzadas tanto por el sistema de ejecución como por la aplicación.

En C++ se pueden lanzar excepciones de cualquier tipo de objeto sin declararlas previamente.

Gestión de las excepciones en la actualidad

Dominio del manejador

- El dominio de un manejador de excepción es la zona de código en que se activa el manejador si se produce en ella la excepción que controla.
- En los lenguajes estructurados, el dominio coincide con un bloque de programación (bloque, función, procedimiento…).
- Una excepción puede tener varios manejadores.
- Un manejador se puede definir para una excepción o para varias.

El paso de parámetros junto a las excepciones facilita al manejador saber cuál fue la operación que provocó la excepción.

Java lo hace automáticamente, dado que una excepción es un objeto que puede contener tanta información como se desee.

Ada tiene un procedimiento predefinido *Exception-Information* que devuelve detalles (definidos en la implementación) sobre la aparición de la excepción.

Gestión de las excepciones en la actualidad

Propagación de excepciones

Cuando no hay un manejador de excepciones en un bloque de código y se produce una excepción hay dos opciones:

- Entender la ausencia de manejador como un error de programación y notificarlo en tiempo de compilación.

  El compilador no siempre puede detectarlo, así que los procedimientos deben especificar las excepciones que no tratan localmente. (Java, C++)

- Buscar manejadores en el pasado de la cadena de invocación en tiempo de ejecución, esto se conoce como **propagación de excepciones**. (Ada, Java, C++, Modula 2/3)

Gestión de las excepciones en la actualidad

Modelos

Existen varios modelos de excepción, dependiendo de cómo continúa la ejecución después del manejo de la excepción:

- Modelo de reanudación (**resumption**), en el caso de que el invocador sea capaz de manejar la excepción propagada por el proceso invocado, este aplicará las acciones necesarias y continuará con su ejecución. (Perl, Mesa, Eiffel)
- Modelo de finalización (**termination**), cuando el control de la excepción no se devuelve al invocador, sino que se pasa el control al procedimiento de llamada. (Ada, Java, C++, Modiula 2/3, CHILL)
- Modelos **híbridos**, el manejador es el que decide si continuar con la ejecución del proceso o terminar la ejecución (Mesa, Real Time Basic, Eiffel)

Manejo de excepciones en Ada

Ada soporta la declaración explícita de excepciones, el modelo de terminación de manejo de excepciones con propagación de excepciones no manejadas, y una forma limitada de parametrización de excepciones.

Declaración de excepciones

- El tipo de constante definido por la palabra clave **exception**.
- El tipo privado llamado Exception-Id del paquete predefinido Ada.Exceptions.

Excepciones estándar: Constraint-Error y Storage-Error.

Lanzar una excepción

Un programa puede generar las excepciones con la sentencia **raise.**

Cada generación de excepción se denomina **ocurrencia** de una excepción, y se representa mediante un valor del tipo Ada.Exceptions.Exception-Ocurrence que aporta información sobre la causa de la excepción.

Manejo de excepciones en Ada

Manejo de excepciones

Cada bloque Ada puede contener un conjunto de manejadores de excepciones, declarados al final del bloque.

Cada manejador comienza por **when**, un parámetro opcional para la identidad de la ocurrencia, las excepciones que serán gestionadas y el símbolo **=>**.

Como último manejador se puede poner uno que trate cualquiera de las excepciones no recogidas en los anteriores, usando **when others.**

Una excepción generada en un manejador de excepción no puede ser manejada en el mismo bloque. Se termina el bloque y se busca un manejador en el bloque exterior o en el punto de llamada del subprograma.

```
declare
   Sensor_High, Sensor_Low, Sensor_Dead : exception;
   -- other declarations
begin
   -- statements that may cause the above exceptions
   -- to be raised
exception
   when E: Sensor_High | Sensor_Low =>
      -- Take some corrective action
      -- if either Sensor_High or Sensor_Low is raised.
      -- E contains the exception occurrence
   when Sensor_Dead =>
      -- sound an alarm if the exception
      -- Sensor_Dead is raised
end;
```

Manejo de excepciones en Ada

Propagación de excepciones

En los bloques sin manejador Ada propaga las excepciones hacia el bloque superior o el punto de invocación del subprograma.

Últimas voluntades

Un manejador local puede relanzar excepciones:

La sentencia **raise** (o el procedimiento **Ada.Exceptions.Reraise-Ocurrence**) regenera la última excepción (o una excepción específica).

También se pueden utilizar **tipos controlados**: consiste en definir subprogramas que se ejecuten automáticamente cuando algún objeto de un tipo controlado se cree, se elimine o se modifique

Supresión de excepciones

Algunas excepciones estándar son manejadas de manera automática por el compilador si el programador no lo hace. Esto se puede suprimir con la directiva **Suppress**

Manejo de excepciones en Java

Java es similar a Ada en cuanto a que permite el modelo de terminación de manejo de excepciones.

Sin embargo, las excepciones de Java se integran en el modelo orientado al objeto.

Declaración de excepciones

En Java, todas las excepciones son subclases de la clase predefinida **java.lang.Throwable**.

Se incluyen algunas clases ya definidas: **Error**, **Exception**, y **RuntimeException**.

Cuando hablamos de excepciones nos referimos a cualquier clase derivada de **Throwable**, y no sólo a las derivadas de **Exception**.

Cada función debe especificar una lista de excepciones lanzables comprobadas, con la palabra **throws**

Lanzar una excepción

Al hecho de generar una excepción en Java se le denomina **lanzar** o arrojar la excepción.

Se utiliza la palabra **throw.**

Manejo de excepciones en Java

Manejo de excepciones

En Java, una excepción sólo puede ser manejada desde el interior de un bloque **try**.

Cada manejador se especifica mediante una sentencia **catch**.

Propagación de excepciones

Como en Ada, si no se encuentra ningún manejador de excepción en el contexto de llamada de una función, el contexto de llamada se da por terminado y se busca un manejador en su contexto de llamada superior.

Últimas voluntades

Se puede usar la cláusula **finally** en un bloque try-catch, para tratar las excepciones no gestionadas (o incluso aunque no se haya producido ninguna excepción).

Manejo de excepciones en C

En el lenguaje C no existe un mecanismo para el manejo de excepciones, lo que limita la utilidad del lenguaje para la programación estructurada de sistemas fiables.

Aun así, es posible proporcionar algún mecanismo de manejo de excepciones mediante el entorno de macros del lenguaje.

Para implementar en C un modelo de **terminación**, hay que guardar el estado del programa a la entrada del dominio de una excepción, para restaurarlo en el caso de excepción.

Se pueden utilizar las primitivas de POSIX ***setjmp*** y ***longjmp***.

- La rutina ***setjmp*** almacena el estado del programa y devuelve un O;
- la rutina ***longjmp*** restaura el estado del programa y provoca que el programa abandone su ejecución actual reiniciándose desde la posición donde se invocó el salto.

Bloques de recuperación y excepciones

En este tema se ha visto cómo tratar los errores anticipables (hacia adelante).

Los bloques de recuperación pueden emplearse para recuperarse de errores imprevistos, particularmente de errores en el diseño de componentes software.

- Son bloques en el sentido usual de los lenguajes:
  - La entrada corresponde a un punto de recuperación.
  - En la salida se realiza un test de aceptación.
- Funcionamiento:
  - Primero se guarda el estado y ejecuta un módulo primario.
  - Si falla el test se restaura el estado y se ejecuta un módulo alternativo.


UNED
Página Principal
Área personal
Mis cursos
Soporte técnico
Foro de Tutoría
Foro de coordinación tut...
MÓDULO 1: CONCEPTO...
Foro de dudas del mód...
Auto-evaluación de la ...
Autoevaluación de la U...
Autoevaluación de la U...
MÓDULO 2: ASPECTOS ...
Foro de dudas del mód...
Dudas PED1
Dudas PED2
Autoevaluación de la U...
Autoevaluación de la U...
Autoevaluación de la U...
Autoevaluación de la U...
Autoevaluación de la U...
MÓDULO 1: CONCEPTOS BÁSICOS DE LOS SISTEMAS EN TIEMPO REAL
Este módulo explica los fundamentos de los sistemas de tiempo real y las características a tener en cuenta en su desarrollo. Se define el modelo de fallo y el concepto de tolerancia a fallos, indicando algunas técnicas para su resolución. Se termina el módulo explicando cómo se gestionan las excepciones (posibles errores) desde el punto de vista del lenguaje de programación y del propio sistema operativo (en tiempo real o no)
Foro de dudas del módulo 1
Marcar como hecha
Autoevaluaciones de las unidades
Auto-evaluación de la Unidad 1: Introducción a los sistemas en tiempo real
Autoevaluación de la Unidad 2: Fiabilidad y tolerancia a fallos
Autoevaluación de la Unidad 3: Excepciones y manejo de excepciones

Sistemas en Tiempo Real

Grado en Ingeniería Electrónica

UNIDAD 3: Excepciones y manejo de excepciones

Nieves Pedreira Souto