Throw/Catch Control Flow Pattern

`rs-try` version 2017-10-29

Ralph Schleicher

This is the reference manual for the `rs-try` library version 2017-10-29.

Table of Contents

1 Introduction **1**

2 API Reference **2**

2.1 Catch Tags 2
2.2 Control Flow 2
2.3 Meta Data 3
2.4 Error Handling 3
2.5 Customization 3

3 Examples **4**

3.1 Catch All Handler 4
3.2 Cleanup with Finally 5

Concept Index **6**

Symbol Index **7**

1 Introduction

The `rs-try` library implements the throw/catch control flow pattern in C. It's general form is depicted below.

```
try
  {
    /* ... */
    throw (tag);
    /* ... */
  }
catch (tag)
  {
    /* ... */
  }
finally
  {
    /* ... */
  }
```

Each try block establishes a throw/catch control flow environment with zero or more catch blocks and an optional finally block. If a throw occurs during the execution of the try block, control is transferred immediately to the most recently established throw/catch control flow environment. If there is a catch block matching the tag argument of the throw, it is executed. Then the optional finally block is executed and the throw/catch control flow environment is disestablished. If there is no matching catch block, control is transferred to the then most recently established throw/catch control flow environment.

A catch block matches if the thrown tag is a sub-type of any of the tags specified by the catch block.

The code of a try, catch, and finally block must not be compound statement, it can also be an expression statement.

2 API Reference

All symbols described in this chapter are defined in the header file `rs-try.h`.

2.1 Catch Tags

`rs_try_tag_t` [Data Type]
The data type of a catch tag object.

This is an opaque data type. You only deal with pointers to catch tag objects.

`rs_try_define_tag` (*name*, *rs_try_tag_t const *super*) [Macro]
Define a catch tag.

- First argument *name* is the symbolic name of the catch tag.
- Second argument *super* is the super-type of the catch tag. This can be used to build a hierarchical tree of catch tags. If there is no super-type, use `NULL`.

`rs_try_declare_tag` (*name*) [Macro]
Declare a catch tag.

- Argument *name* is the symbol name of the catch tag.

`int rs_try_subtypep` (*rs_try_tag_t const *tag1*, *rs_try_tag_t const *tag2*) [Function]
Return true if *tag1* is a sub-type of *tag2*.

- Arguments *tag1* and *tag2* are catch tags.

`rs_try_tag_t const * rs_try_super_tag` (*rs_try_tag_t const *tag*) [Function]
Return the super-type of a catch tag.

- Argument *tag* is a catch tag. If *tag* is a null pointer, the return value is `NULL`.

`char const * rs_try_tag_name` (*rs_try_tag_t const *tag*) [Function]
Return the symbol name of a catch tag as a string constant.

- Argument *tag* is a catch tag. If *tag* is a null pointer, the return value is `"NULL"`.

2.2 Control Flow

`rs_try` [Macro]
Begin a throw/catch control flow environment.

`rs_catch` (*tag*, ...) [Macro]
Define a catch block.

Arguments are one or more catch tags. The catch block matches if the tag argument of the throw is a sub-type of any of the specified catch tags.

`rs_finally` [Macro]
Define a finally block.

The finally block is always executed if the corresponding try and catch blocks return.

`rs_throw` (*tag*) [Macro]
Throw a catch tag.

- Argument *tag* is a catch tag.

2.3 Meta Data

Throwing

`rs_try_tag_t const * rs_try_throw_tag ()` [Function]
Return the tag argument of the throw.

`char const * rs_try_file_name ()` [Function]
Return the source file name where the throw occurs.

`int rs_try_line_number ()` [Function]
Return the source line number where the throw occurs.

`int rs_try_error_number ()` [Function]
Return the value of `errno` when the throw occurred.

Catching

`rs_try_tag_t const * rs_try_catch_tag ()` [Function]
Return the super-type matching the thrown tag.

2.4 Error Handling

A control error terminates the program by calling `abort`.

`void (*) (void) rs_try_control_error_hook` [Variable]
Function to be called if a control error occurs.

In case of an error, `errno` is set to describe the error. The following error conditions are defined for this function.

`EFAULT` A throw occurs but there is no corresponding catch block.

`ENOMEM` There it not enough memory available to establish a new throw/catch control flow environment.

`EINVAL` The most recently established throw/catch control flow environment got lost. This can not happen in a correct program.

2.5 Customization

These macros have to be defined before the header file `rs-try.h` is included.

`int RS_TRY_USE_SIGJMP` [Macro]
If true, use `sigsetjmp` and friends instead of `setjmp`. Disabled by default.

`int RS_TRY_USE_THREADS` [Macro]
If true, allocate throw/catch control flow environments in thread-local storage. Enabled by default.

`int RS_TRY_USE_KEYWORDS` [Macro]
If true, define keywords `try`, `catch`, `finally`, and `throw` as aliases for `rs_try`, `rs_catch`, `rs_finally`, and `rs_throw` respectively. Disabled by default.

`int RS_TRY_STACK_SIZE` [Macro]
If the value is a positive number, allocate that many throw/catch control flow environments on the stack, i.e. the number of nested try blocks is fixed. Otherwise, allocate throw/catch control flow environments dynamically on the heap. The later is the default.

3 Examples

3.1 Catch All Handler

Since NULL is the super-type of all tags, it can be used to establish a catch all handler.

```
#define RS_TRY_USE_KEYWORDS 1

#include <stdlib.h>
#include <stdio.h>
#include "rs-try.h"

rs_try_define_tag (BALL, NULL);
rs_try_define_tag (MUD, NULL);

static void
sub (rs_try_declare_tag (thing))
{
  try
    {
      fprintf (stderr, "Throwing ‘%s’\n",
               rs_try_tag_name (thing));

      throw (thing);
    }
  catch (BALL)
    {
      fprintf (stderr, "Caught ‘%s’\n",
               rs_try_tag_name (rs_try_throw_tag ()));
    }
}

int
main (void)
{
  try
    {
      sub (BALL);
      sub (MUD);
    }
  catch (NULL)
    {
      fprintf (stderr, "Unhandled tag ‘%s’ thrown at ‘%s’ line %d\n",
               rs_try_tag_name (rs_try_throw_tag ()),
               rs_try_file_name (),
               rs_try_line_number ());
    }

  return 0;
}
```

The output of this example is as follows:

```
Throwing ‘BALL’
Caught ‘BALL’
Throwing ‘MUD’
Unhandled tag ‘MUD’ thrown at ‘example.c’ line 18
```

3.2 Cleanup with Finally

Whether or not a throw occurs, a finally block is always executed.

```
#define RS_TRY_USE_KEYWORDS 1

#include <stdlib.h>
#include <stdio.h>
#include "rs-try.h"

int
main (void)
{
  try
    {
      try
        {
          fprintf (stderr, "Setup\n");
          fprintf (stderr, "Working\n");
        }
      finally
        fprintf (stderr, "Cleanup\n");

      try
        {
          fprintf (stderr, "Before throwing\n");
          throw (RS_TRY_ERROR);
          fprintf (stderr, "Not reached\n");
        }
      finally
        fprintf (stderr, "After throwing\n");
    }
  catch (RS_TRY_CONDITION)
    {
      fprintf (stderr, "‘%s’ caught by ‘%s’\n",
               rs_try_tag_name (rs_try_throw_tag ()),
               rs_try_tag_name (rs_try_catch_tag ()));
    }
  catch (NULL)
    {
      fprintf (stderr, "Unhandled tag ‘%s’ thrown at ‘%s’ line %d\n",
               rs_try_tag_name (rs_try_throw_tag ()),
               rs_try_file_name (),
               rs_try_line_number ());

      abort ();
    }

  return 0;
}
```

The output of this example is as follows:

```
Setup
Working
Cleanup
Before throwing
After throwing
‘RS_TRY_ERROR’ caught by ‘RS_TRY_CONDITION’
```

Concept Index

control error 3

Symbol Index

rs_catch 2
rs_finally 2
rs_throw 2
rs_try 2
rs_try_catch_tag 3
rs_try_control_error_hook 3
rs_try_declare_tag 2
rs_try_define_tag 2
rs_try_error_number 3
rs_try_file_name 3
rs_try_line_number 3
rs_try_subtypep 2
rs_try_super_tag 2
rs_try_tag_name 2
rs_try_tag_t 2
rs_try_throw_tag 3
RS_TRY_STACK_SIZE 3
RS_TRY_USE_KEYWORDS 3
RS_TRY_USE_SIGJMP 3
RS_TRY_USE_THREADS 3