

# Exception Handling in mod\_perl - 45mins

**Matt Sergeant**  
**AxKit.com Ltd**

# Overview

- Intro to Exceptions
- `$SIG{__DIE__}` considered harmful
- Trapping exceptions in `mod_perl`
- Throwing objects
- `$SIG{__DIE__}` revisited
- Useful exception modules
- Exceptions in real life

# Introduction and `$SIG{__DIE__}`

# What are Exceptions?

- Better method of error handling
- Like wrapping every line of code with:

```
if (testError(<code>)) goto errorHandler
```

- Easier to manage than return codes
- Cannot forget to test for an exception
- Easier to propagate up the call stack

# Exceptions in Perl

- Exceptions thrown by die() or croak()
- Exceptions trapped with eval { <code> };
- Exceptions tested with if (\$@)

```
eval {  
    ...  
    if ($some_condition) {  
        die "Some condition failed!";  
    }  
    ...  
};  
if ($@) {  
    # process exception  
}
```

## **`$_SIG{__DIE__}` considered harmful**

- `$_SIG{__DIE__}` allows you to catch exceptions
- Action at a distance
- `$_SIG{__DIE__}` unfortunately overrides `eval {...}`
- Other modules erroneously set `$_SIG{__DIE__}` to trap exceptions
- e.g. older versions of `CGI::Carp qw(fatalsToBrowser);`

# \$SIG{\_\_DIE\_\_} evil example

```
$SIG{__DIE__} = sub {  
    my @err = @_;  
    warn("Caught error: ", @err);  
    exit(55);  
};  
  
sub main {  
    ...  
    Foo::do_something();  
    ...  
}  
  
package Foo;  
  
sub do_something {  
    eval {  
        # do something that may die  
    };  
    if ($@) {  
        # process exception  
    }  
}
```

# Documented work arounds for \$SIG{\_\_DIE\_\_} "bug"

- Localising \$SIG{\_\_DIE\_\_}

```
eval {  
    local $SIG{__DIE__};  
    die "foo happened";  
};
```

- \$EXCEPTIONS\_BEING\_CAUGHT

```
$SIG{__DIE__} = sub {  
    die @_ if $^S;  
    ...  
};
```

- The former does not scale to large projects
- Both are problematic with modules you don't own

# Exception handling in mod\_perl

# Apache::Registry or Apache::PerlRun

- For running CGI scripts without change
- Wrap code in exception handler:

```
my $r = shift;
eval {
    ...
};
if ($?) {
    ...
}
```

- Watch out for closures!
  - ◆ Especially if using Error.pm (see later)

# A mod\_perl handler

- mod\_perl calls a "handler" function (or method)

```
package MyHandler;
```

```
sub handler {  
    my $r = shift;  
    eval {  
        ...  
    };  
    if ($@) {  
        ...  
    }  
}
```

- Enabled with:

```
PerlHandler MyHandler
```

# Throwing Objects and `$SIG{__DIE__}`

# Throwing Objects

- Perl 5.5 allowed you to throw objects as well as strings
- This allows for greater information in your exception

```
eval {  
    die MyException->new();  
};  
if ($@) {  
    # here $@ is the new MyException object  
}
```

- Exception object constructor can get a stack trace using caller()
- Can also contain text and an error code, or SQL executed, or...

# \$SIG{\_\_DIE\_\_} redux

- \$SIG{\_\_DIE\_\_} really can be useful, to catch non-blessed errors and give a stack trace

```
sub handler {
  local $SIG{__DIE__} = sub {
    my $err = shift;

    if (ref($err)) {
      return $err;
    }
    else {
      return MyBase::Exception::Error->new(text => $err);
    }
  };

  eval {
    ... # main code here
  };
  process_error($@) if $@;
}
```

# Useful Exception Modules

# Fatal.pm

- Core Perl module
- Makes functions throw exceptions rather than return false

```
use Fatal qw(open close);
eval {
    open(FILE, "doesnotexist.txt");
    close FILE;
};
if ($?) {
    warn("Fatal error: $?");
}
```

## ◆ Results:

```
Fatal error: Can't open(FILE, doesnotexist.txt): No such file or directory
main:::__ANON__('FILE', 'doesnotexist.txt') called at /tmp/nofile.pl line 3
eval {...} called at /tmp/nofile.pl line 3
```

# Error.pm

- Provides rich blessed exception objects
- and a java/smalltalk - like try/catch syntax

```
try {
    open(FILE, "notexist") ||
        throw Error::IO (-text => "File did not exist: $!");
}
catch Error::Simple with {
    my $E = shift;
    ...
}
catch Error::IO with {
    my $E = shift;
    ...
}
finally {
}; # note trailing semi-colon
```

## Error.pm (cont.)

- Implemented using perl function prototype of "&"
- This turns blocks into closures
- While this is nice syntactic sugar, it can be dangerous

```
sub handler {  
    my $r = shift;  
    my $count;  
    try {  
        # use $count in here, it's now a closure. Very bad!  
    } catch Error with {  
        ...  
    };  
}
```

# Error.pm without try/catch

```
eval {
    open(FILE, ">/root/badperms") ||
        throw Error::IO (-text => "Cannot open: $!");
};
if ($@) {
    my $E = $@;
    if ($E->isa("Error::IO")) {
        Apache->request->log->error(
            "IO Error: ".$E->{-text}." at ".$E->{-file}." : ".$E->{-
        );
    }
}
```

# Real Life mod\_perl Exception Handling

# AxKit's Exceptions module

- Subclass of Error.pm
  - ◆ Apache::AxKit::Exception::Error
  - ◆ Apache::AxKit::Exception::IO
  - ◆ Apache::AxKit::Exception::OK
  - ◆ Apache::AxKit::Exception::Declined
  - ◆ Apache::AxKit::Exception::Retval
- Exceptions allow very deep code to exit safely

```
throw Apache::AxKit::Exception::Retval ( return_code => 304 );
```

# How not to use exceptions

- Don't catch them!
- At least leave catching exceptions to your outer error handler
- Caught exceptions lose their stack trace
- It's very unlikely you actually want to catch an exception
- Do throw exceptions on errors though. Don't try and handle errors manually

# Exceptions Performance

- Exceptions seem to be slower under mod\_perl?
- In AxKit, I detected 12ms between throw and if(\$@) on my PII500 (Perl 5.6.1)
- Use carefully in performance critical sections of code
- Benchmark if your system seems slow
- \$SIG{\_\_DIE\_\_} contributes to slow down
- As do exception objects

# Conclusions

- Programmer time vs execution time
- Bugs caught easier
- Impossible to ignore exceptions
- They are a good thing!