What is an exception?

This thread has been locked. Questions are automatically locked after two months of inactivity, or sooner if deemed necessary by a moderator.

20

The word *exception* is highly overloaded, not just on Apple platforms but across the industry as a whole. From an Apple perspective there are 3 things that are commonly conflated under that term:

- Machine exceptions — These are raised by the hardware in response to problems detected by the hardware, for example, accessing invalid memory, executing an illegal instruction, and executing a trap instruction.
- Language exceptions — This includes Objective-C (`@try`, `@catch`, `@throw`) and C++ exceptions (`try`, `throw`, `catch`)
- Cocoa errors

Catching machine exceptions is *super* hard and it only makes sense in very specific circumstances, for example, when you're working on a language runtime.

IMPORTANT Folks commonly try to catch machine exceptions as part of a custom crash reporter. I *strongly* recommend against doing that, for the reasons I outline in Implementing Your Own Crash Reporter.

The Exception Handling framework, which is the tag that I applied to this post (-:, lets you convert machine exceptions to language exceptions. This is dangerous nonsense and should never be used.

The situation with language exceptions varies by language:

- In C++ it's common to use language exceptions as part of your program.
- In Objective-C language exceptions are reserved for serious programming errors. Do not throw a language exception unless you want your program to crash. Do not attempt to catch a language exception and then recover from it. Doing so will not work reliably if you're using ARC or if the language exception originated in the OS.
- Swift has no facilities for dealing with language exceptions. The exception-like mechanisms you see in Swift are actually syntactic sugar on the Cocoa error facilities (more on that below).

In no situation is it safe to throw or catch a language exception across an ABI boundary.

Note Historically some Cocoa APIs expected you to catch language exceptions. These APIs are now either deprecated (for example, Distributed Objects) or have been replaced by APIs that use the Cocoa error mechanism (for example, `NSFileManager`).

The Cocoa error mechanism involves a function that returns a status result and can optionally return an `NSError` via an 'out' parameter. In Objective-C this is done using an `NSError **` parameter. For example, to read an `NSData` from a file you use this `NSData` method:

```
+ (nullable instancetype)dataWithContentsOfURL:(NSURL *)url options:(NSDataReadingOptions)readOptionsMask error:
(NSError **)errorPtr;
```

To see the error in Objective-C, call it like so:

```
1 NSURL * url = … something …;
2 NSError * error;
3 NSData * data = [NSData dataWithContentsOfURL:url options:0 error:&error];
4 if (data == nil) {
5     … look at `error` …
6 }
```

IMPORTANT Only look at `error` if the function result indicates that an error occurred.

Swift has syntactic sugar to make this look like an exception mechanism. For example, in Swift you'd call it like so:

```
1 let url: URL = … something …
2 let data = try Data(contentsOf: url: options:[])
```

While this looks like an exception, it's not. Rather, it's a convenient way to handle the existing Cocoa error convention.

Share and Enjoy
—
Quinn "The Eskimo!" @ Developer Technical Support @ Apple
`let myEmail = "eskimo" + "1" + "@" + "apple.com"`

ExceptionHandling

Asked 1 day ago by eskimo

Reply to this question