

MANAGING YOUR RESOURCES

Denys Shabalin, LAMP/EPFL

[@den_sh](#)

Alternative talk title:

**TRY-FINALLY
CONSIDERED HARMFUL**

EVERYBODY GETS IT WRONG

It's hard to impossible to write exception safe code.

FEW PEOPLE GET IT RIGHT

Akka's actor cell cleanup:

```
private def finishTerminate() {  
  val a = actor  
  /* The following order is crucial for things to work properly.  
   * Only change this if you're very confident and lucky.  
   */  
  try if (a ne null) a.aroundPostStop()  
  catch handleNonFatalOrInterruptedException { ... }  
  finally try dispatcher.detach(this)  
  finally try parent.sendSystemMessage(...)  
  finally try stopFunctionRefs()  
  finally try tellWatchersWeDied()  
  finally try unwatchWatchedActors(a)  
  finally {  
    ...  
  }  
}
```

WHAT'S A RESOURCE?

An entity that you can:

- Acquire: get an instance of a resource
- Release: dispose of the used up resource

Time between the two = resource lifetime.

RESOURCES IN THE WILD

- File handles
- Network connections
- Locks
- Off-heap memory allocations
- Thread pools
- Actor systems
- OpenGL Contexts
- ...

COMMON THEME

Resources are heavy.

You don't want to lose them.

You do want to release them ASAP.

THINGS PEOPLE DO

- Manage it manually
- RAI (Resource Acquisition Is Initialization)
- Scope guards
- Using statement
- Monads

MANUAL MANAGEMENT

As seen in: every single language.

MANUAL MANAGEMENT: C

The only language where it's OK to manage things manually.


```
int main() {  
    FILE* f = fopen("file.txt", "w");  
  
    // Do some work here  
  
    fclose(f);  
    return 0;  
}
```

Λ
|
| lifetime of f
|
V

MANUAL MANAGEMENT: JAVA

Standard resource handling before Java SE 7:

```
public class MyApp {  
    public static void main(String[] args) {  
        PrintWriter f = new PrintWriter("file.txt")  
        try {  
            // Do some work here  
  
        } finally {  
            f.close()  
        }  
    }  
}
```



RESOURCE ACQUISITION IS INITIALIZATION

As seen in: C++, D, Rust, ...


RAII: C++

```
class File {  
public:  
    File(const char *path, const char *mode) {  
        _file = fopen(path, mode);  
    }  
    ~File() {  
        fclose(_file);  
    }  
private:  
    FILE* _file;  
}
```

Constructor Acquires, Destructor Releases.

RAII: C++

```
int main() {  
    File f("file.txt", "r");  
    // Do some work here  
    return 0;  
}
```



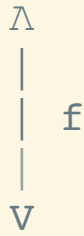
Cleanup is done for you automatically.

SCOPE GUARDS

As seen in: C++, D, Go

SCOPE GUARDS: DEFER IN GO

```
func main() {  
    f, _ := os.OpenFile("file.txt", os.O_WRONLY, 0644)  
    defer f.close()  
  
    // Do some work here  
}
```



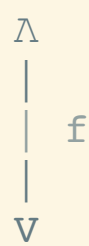
USING

As seen in: C#, Python, Java, Haskell, ...

USING: TRY WITH RESOURCES

Modern Java:

```
public class MyApp {  
    public static void main(String[] args) {  
        try (PrintWriter f = new PrintWriter("file.txt")) {  
            // Do some work here  
        }  
    }  
}
```



MONADS

As seen in: Haskell, Scala

MONADS: SCALA ARM

```
import resource._  
  
for (f <- managed(new PrintWriter("file.txt"))) {  
    // Do some work here  
}  
Λ  
|  
| f  
|  
V
```

CONTEMPORARY SCALA

WHAT HAPPENS IN PRACTICE


```
{  
  val f = new PrintWriter("file.txt"))  
  // Do some work here  
}
```

Λ
|
| f
|
?

It compiles, it runs, ship it.

Home

twitter.com


**Daniel Spiewak**
@djspiewak

Following





Random Scala anti-pattern: encoding resource lifecycles through implicits.


RETWEETS
9

LIKES
29








7:35 AM - 16 Aug 2016


 9 29







Reply to @djspiewak

**Viktor Klang** @viktorklang · Aug 16
@djspiewak wait. what.




**Daniel Spiewak** @djspiewak · Aug 16
@viktorklang


1) acquire resource; mark as implicit
2) take resource as implicit param everywhere
3) leak resource like crazy


 2

Daniel Spiewak
@djspiewak

Scala fanatic. Fun
Overly-fascinated
esoterica.

 Boulder, CO

 [codecommit.c](#)

 Joined June 2

ons. You know
fathoms deep

NOT QUITE

Passing resources as implicits is a bad idea.

Implicits are the *perfect* tool to model resource lifetimes.

UNICORN SOLUTION


```
{  
  val f = new SafeWriter("file.txt")  
  // Do some work here  
}
```

Λ
|
| f
|
 V

GETTING REAL

Encoding resource lifetime through implicits:

```
Scope { implicit in =>
  val f = new SafeWriter("file.txt")
  // Do some work here
}
```



Scope is a first-class lifetime passed around implicitly.

SCOPES = STACKS OF RESOURCES

Resources acquisition requires a scope.

Scopes gurantee clean up once finished.

RESOURCES REQUIRE SCOPES

To acquire a resource one must provide a proof of clean up.

```
class SafeWriter(path: String)(implicit in: Scope) {  
  private val writer = acquire(new PrintWriter(path))  
  def write(msg: String) = writer.write(msg)  
}
```

CAN'T GET IT WRONG

Careless resource handling won't compile any longer.

```
{  
    val f = new SafeWriter("file.txt")  
  
    // Do some work here  
}
```

WE'VE GOT YOUR BACK


MyApp.scala:2: Resource acquisition requires a scope.

```
val f = new SafeWriter("file.txt")  
      ^
```

IT GETS BETTER IN DOTTY

Implicit functions let one inject implicits automatically:

```
Scope {  
  val f = new SafeWriter("file.txt")  
  
  // Do some work here  
}
```



The diagram illustrates the implicit injection of the variable `f` into the `Scope` block. A vertical line connects the `f` in `val f = new SafeWriter("file.txt")` to the `f` in the `// Do some work here` line. The line starts with a Λ at the top and ends with a V at the bottom, representing the implicit function mechanism.

CODING IT UP

gist.github.com/densh/75d2d3063571d89ee34e161b4a61c74a

TAKEAWAY

**GET YOUR STUFF TOGETHER, GET IT
ALL TOGETHER AND PUT IT IN A
BACK PACK, ALL YOUR STUFF, SO
IT'S TOGETHER.**

**AND IF YOU GOTTA TAKE IT SOME
WHERE, TAKE IT SOMEWHERE, YOU
KNOW, TAKE IT TO THE STUFF STORE
AND SELL IT, OR PUT IT IN THE
STUFF MUSEUM.**

**I DON'T CARE WHAT YOU DO, YOU
JUST GOTTA GET IT TOGETHER.**

GET YOUR STUFF TOGETHER.

THANKS!

Follow [@den_sh](#) for more.