Three Unlikely Successful Features of D

Andrei Alexandrescu

Research Scientist

Facebook

© 2012 Andrei Alexandrescu

Predictably Successful Features

© 2012 Andrei Alexandrescu

1. The `scope` Statement: Casual Correct Code

© 2012 Andrei Alexandrescu

⟨action⟩

© 2012 Andrei Alexandrescu

⟨action⟩

⟨cleanup⟩

© 2012 Andrei Alexandrescu

⟨action⟩

⟨cleanup⟩

⟨next⟩

© 2012 Andrei Alexandrescu

⟨action⟩

⟨cleanup⟩

⟨next⟩

⟨rollback⟩

© 2012 Andrei Alexandrescu

C

```
if (⟨action⟩) {
    if (!⟨next⟩) {
        ⟨rollback⟩
    }
    ⟨cleanup⟩
}
```

© 2012 Andrei Alexandrescu

C++

```
class RAII {
    RAII() { ⟨action⟩ }
    ~RAII() { ⟨cleanup⟩ }
};
...
RAII raii;
try {
    ⟨next⟩
} catch (...) {
    ⟨rollback⟩
    throw;
}
```

© 2012 Andrei Alexandrescu

Java, C#

```
⟨action⟩
try {
    ⟨next⟩
} catch (Exception e) {
    ⟨rollback⟩
    throw e;
} finally {
    ⟨cleanup⟩
}
```

© 2012 Andrei Alexandrescu

Go

```
result, error := ⟨action⟩
if error != nil {
    defer ⟨cleanup⟩
    if !⟨next⟩
        ⟨rollback⟩
}
```

© 2012 Andrei Alexandrescu

Composition

C

```
if (⟨action1⟩) {
    if (⟨action2⟩) {
        if (!⟨next2⟩) {
            ⟨rollback2⟩
            ⟨rollback1⟩
        }
        ⟨cleanup2⟩
    } else {
        ⟨rollback1⟩
    }
    ⟨cleanup1⟩
}
```

© 2012 Andrei Alexandrescu

C++

```
class RAII1 {
    RAII1() { ⟨action1⟩ }
    ~RAII1() { ⟨cleanup1⟩ }
};
class RAII2 {
    RAII2() { ⟨action2⟩ }
    ~RAII2() { ⟨cleanup2⟩ }
};
...
```

© 2012 Andrei Alexandrescu

C++

```
RAII1 raii1;
try {
    RAII2 raii2;
    try {
        ⟨next_2⟩
    } catch (...) {
        ⟨rollback_2⟩
        throw;
    }
} catch (...) {
    ⟨rollback_1⟩
    throw;
}
```

© 2012 Andrei Alexandrescu

Java, C#

```
⟨action1⟩
try {
    ⟨action2⟩
    try {
        ⟨next2⟩
    } catch (Exception e) {
        ⟨rollback2⟩
        throw e;
    } finally {
        ⟨cleanup2⟩
    }
} catch (Exception e) {
    ⟨rollback1⟩
    throw e;
} finally {
    ⟨cleanup1⟩
}
```

© 2012 Andrei Alexandrescu

Go

```
result1, error := ⟨action1⟩
if error != nil {
    defer ⟨cleanup1⟩
    result2, error := ⟨action2⟩
    if error != nil {
        defer ⟨cleanup2⟩
        if !⟨next2⟩
            ⟨rollback2⟩
    } else {
        ⟨rollback2⟩
    }
}
```

© 2012 Andrei Alexandrescu

Dislocation + Nesting = Fail

© 2012 Andrei Alexandrescu

Windows Runtime in JavaScript

```
function peerFinder_AcceptRequest() {
        // Accept the connection if the user clicks okay.
        ProximityHelpers.displayStatus("Connecting to " + requestingPeer.displayName + " ...");
        ProximityHelpers.id("peerFinder_AcceptRequest").style.display = "none";

        ProxNS.PeerFinder.connectAsync(requestingPeer).then(
            function (proximitySocket) {
                ProximityHelpers.displayStatus("Connect to " + requestingPeer.displayName + "
succeeded");
                startSendReceive(proximitySocket);
            },
            function (err) {
                ProximityHelpers.displayError("Connect to " + requestingPeer.displayName + " failed
with " + err);
                ProximityHelpers.id("peerFinder_Connect").style.display = "none";
            });
    }
```

“Programs must be written for people to read, and only incidentally for machines to execute.”

– Abelson/Sussman, SICP

© 2012 Andrei Alexandrescu

“Error handling is about maintaining program invariants, and only incidentally about dealing with the error itself.”

– I. Meade Etop

© 2012 Andrei Alexandrescu

Enter D

```
⟨action⟩
scope(failure) ⟨rollback⟩
scope(exit) ⟨cleanup⟩
```

© 2012 Andrei Alexandrescu

But wait, there's more (of the same)

```
⟨action1⟩
scope(failure) ⟨rollback1⟩
scope(exit) ⟨cleanup1⟩
⟨action2⟩
scope(failure) ⟨rollback2⟩
scope(exit) ⟨cleanup2⟩
```

© 2012 Andrei Alexandrescu

Three's a charm

```
⟨action1⟩
scope(failure) ⟨rollback1⟩
scope(exit) ⟨cleanup1⟩
⟨action2⟩
scope(failure) ⟨rollback2⟩
scope(exit) ⟨cleanup2⟩
⟨action3⟩
scope(failure) ⟨rollback3⟩
scope(exit) ⟨cleanup3⟩
... moar please ...
```

© 2012 Andrei Alexandrescu

Example

```
void[] read(string name)
{
    invariant fd = std.c.linux.linux.open(toStringz(name), O_RDONLY);
    cenforce(fd != -1, name);
    scope(exit) std.c.linux.linux.close(fd);

    struct_stat statbuf = void;
    cenforce(std.c.linux.linux.fstat(fd, &statbuf) == 0, name);

    void[] buf;
    auto size = statbuf.st_size;
    if (size == 0)
    {  /* The size could be 0 if the file is a device or a procFS file,
        * so we just have to try reading it.
        */
        int readsize = 1024;
        while (1)
        {
            buf = GC.realloc(buf.ptr, size + readsize, GC.BlkAttr.NO_SCAN)
                [0 .. cast(int)size + readsize];
            enforce(buf, "Out of memory");
            scope(failure) delete buf;

            auto toread = readsize;
            while (toread)
            {
                auto numread = std.c.linux.linux.read(fd, buf.ptr + size, toread);
                cenforce(numread != -1, name);
                size += numread;
                if (numread == 0)
                {   if (size == 0)                      // it really was 0 size
                        delete buf;                     // don't need the buffer
                    return buf[0 .. size];              // end of file
                }
                toread -= numread;
            }
        }
    }
    else
    {
        buf = GC.malloc(size, GC.BlkAttr.NO_SCAN)[0 .. size];
        enforce(buf, "Out of memory");
        scope(failure) delete buf;

        cenforce(std.c.linux.linux.read(fd, buf.ptr, size) == size, name);

        return buf[0 .. size];
    }
}
```

```
void[] read(string name)
{
    immutable fd = std.c.linux.linux.open(toStringz(name), O_RDONLY);
    cenforce(fd != -1, name);
    scope(exit) std.c.linux.linux.close(fd);

    struct_stat statbuf = void;
    cenforce(std.c.linux.linux.fstat(fd, &statbuf) == 0, name);

    immutable initialAlloc = statbuf.st_size ? statbuf.st_size + 1 : 1024;
    void[] result = GC.malloc(initialAlloc, GC.BlkAttr.NO_SCAN)
        [0 .. initialAlloc];
    scope(failure) delete result;
    size_t size = 0;

    for (;;)
    {
        immutable actual = std.c.linux.linux.read(fd, result.ptr + size,
                result.length - size);
        cenforce(actual != actual.max, name);
        size += actual;
        if (size < result.length) break;
        auto newAlloc = size + 1024 * 4;
        result = GC.realloc(result.ptr, newAlloc, GC.BlkAttr.NO_SCAN)
            [0 .. newAlloc];
    }

    return result[0 .. size];
}
```

© 2012 Andrei Alexandrescu

2-5x improvement on relevant metrics

© 2012 Andrei Alexandrescu

. . . on code *you* write

© 2012 Andrei Alexandrescu

Straight line + Implicit flow = Win

© 2012 Andrei Alexandrescu

2. Built-in Arrays

© 2012 Andrei Alexandrescu

Systems-Level Language

© 2012 Andrei Alexandrescu

Pointers?

© 2012 Andrei Alexandrescu

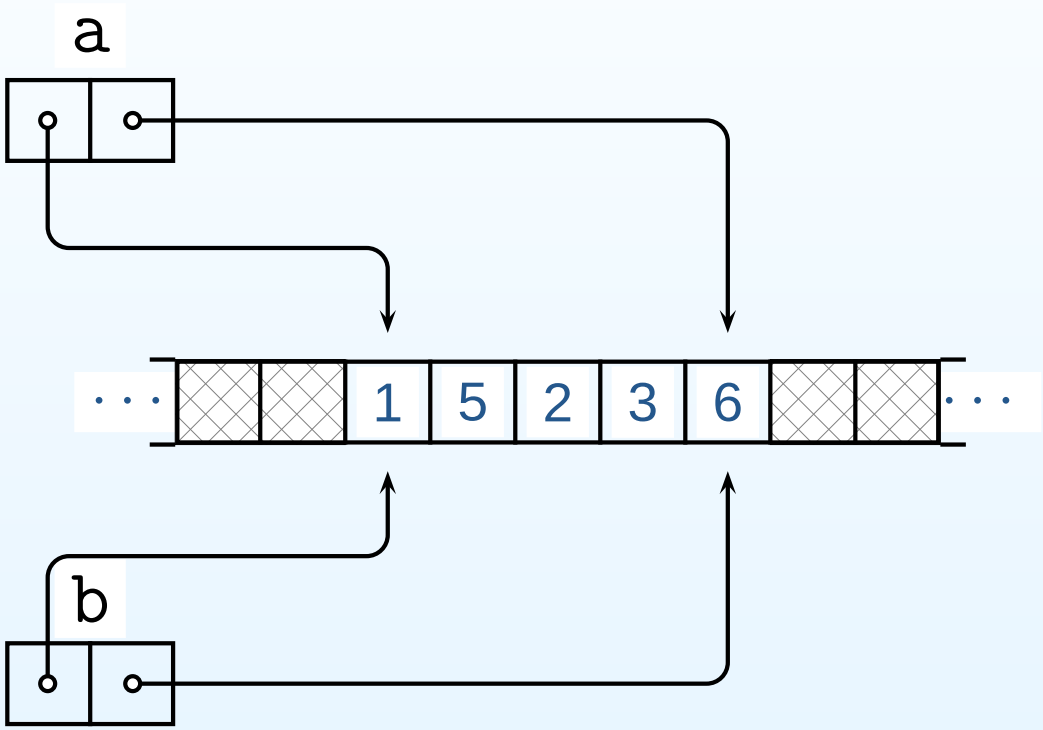
Unsafe iteration

Unsafe arithmetic

Efficient

© 2012 Andrei Alexandrescu

D array = pointer + length



© 2012 Andrei Alexandrescu

Safe iteration

Safe indexing

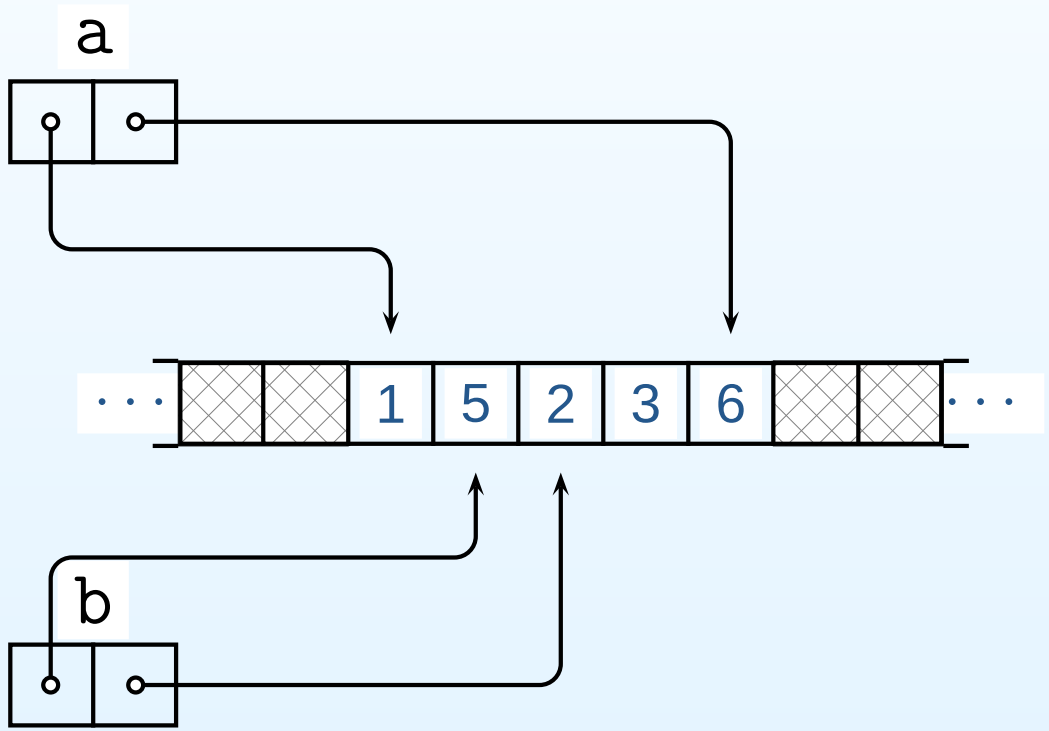
Efficient

Enabling

© 2012 Andrei Alexandrescu

Cheap, type-preserving slicing

```
b = b[1 .. $ - 1];
```



© 2012 Andrei Alexandrescu

Convenient

```
bool palindrome(T)(T[] a) {
  for (; a.length > 1; a = a[1 .. $ - 1]) {
    if (a[0] != a[$ - 1])
      return false;
  }
  return true;
}
```

© 2012 Andrei Alexandrescu

Generalization

Pointers $\rightarrow$ Iterators

Arrays $\rightarrow$ Ranges

© 2012 Andrei Alexandrescu

Palindrome generalized

```
bool palindrome(Range)(Range r) {
  for (; !r.empty; r.popFront(), r.popBack()) {
    if (a.front != a.back)
      return false;
  }
  return true;
}
```

© 2012 Andrei Alexandrescu

3. Compile-time evaluation and `mixin`

© 2012 Andrei Alexandrescu

Embedded DSLs

© 2012 Andrei Alexandrescu

Embedded DSLs
Introspection

© 2012 Andrei Alexandrescu

Embedded DSLs
Introspection
Macro systems

© 2012 Andrei Alexandrescu

Embedded DSLs

Force into host language's syntax

© 2012 Andrei Alexandrescu

Embedded DSLs

- Formatted printing?

© 2012 Andrei Alexandrescu

Embedded DSLs

- Formatted printing?
- Regular expressions?

© 2012 Andrei Alexandrescu

Embedded DSLs

- Formatted printing?
- Regular expressions?
- EBNF?

© 2012 Andrei Alexandrescu

Embedded DSLs

- Formatted printing?
- Regular expressions?
- EBNF?
- PEG?

© 2012 Andrei Alexandrescu

Embedded DSLs

- Formatted printing?
- Regular expressions?
- EBNF?
- PEG?
- SQL?

© 2012 Andrei Alexandrescu

Embedded DSLs

- Formatted printing?
- Regular expressions?
- EBNF?
- PEG?
- SQL?

- . . . Pasta for everyone!

© 2012 Andrei Alexandrescu

Embedded DSLs

Here: use with native grammar

Process during compilation

Generate D code accordingly

© 2012 Andrei Alexandrescu

Expression parser

```
import pegged.grammar; // by Philippe Sigaud
mixin(grammar("
  Expr      <   Factor AddExpr*
  AddExpr   <   ('+'/'-') Factor
  Factor    <   Primary MulExpr*
  MulExpr   <   ('*'/'/') Primary
  Primary   <   Parens / Number / Variable
                / '-' Primary
  Parens    <   '(' Expr ')'
  Number    <~ [0-9]+
  Variable <- Identifier
"));
```

How does it work?

```
enum s = ⟨stringExpression⟩;
```

Evaluate expression during compilation

Most of safe D available

How does it work?

```
mixin(stringExpression);
```

Evaluate expression

Feed the string back to the compiler

The circle closes!

© 2012 Andrei Alexandrescu

Static use of expression parser

```
// Parsing at compile-time:
enum parseTree = Expr.parse(
  "1 + 2 - (3*x-5)*6");
pragma(msg, parseTree1.capture);
```

© 2012 Andrei Alexandrescu

Dynamic use of expression parser

```
// Parsing at run-time:
auto parseTree = Expr.parse(readln());
writeln(parseTree.capture);
```

© 2012 Andrei Alexandrescu

Scaling up

1000 lines of D grammar $\rightarrow$
3000 lines D parser

© 2012 Andrei Alexandrescu

Highly integrated lex+yacc

© 2012 Andrei Alexandrescu

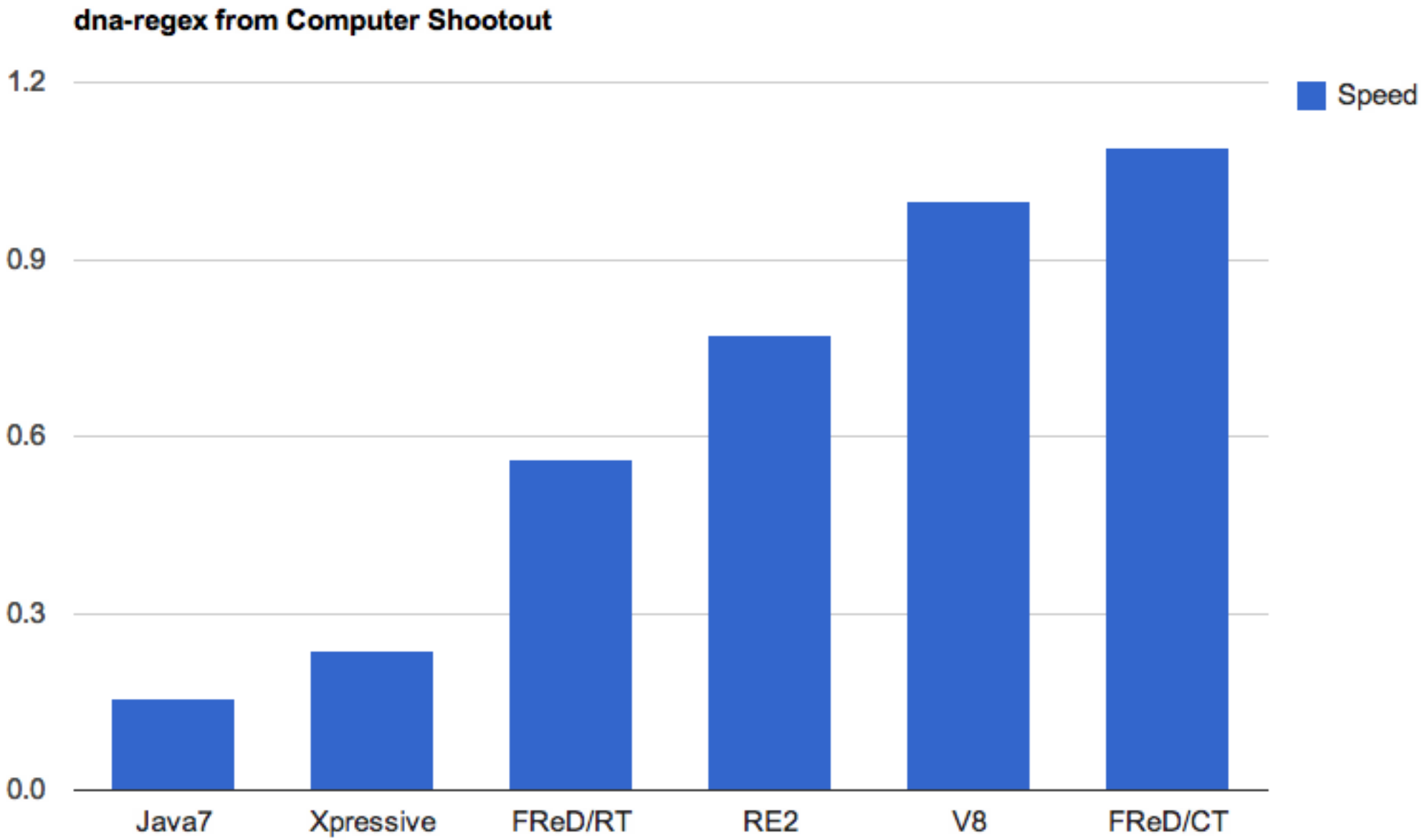
What about regexen?

© 2012 Andrei Alexandrescu

Integrated static/dynamic engine

```
import std.regex;
auto r1 = regex("^.*/([^/]+)/?$");
enum r2 = ctRegex!("^.*/([^/]+)/?$");
```

© 2012 Andrei Alexandrescu


dna-regex from Computer Shootout
1.2
0.9
0.6
0.3
0.0
Speed
Java7
Xpressive
FReD/RT
RE2
V8
FReD/CT

Summary

© 2012 Andrei Alexandrescu

Summary

1. The `scope` statement

© 2012 Andrei Alexandrescu

Summary

1. The `scope` statement

2. Built-in arrays

© 2012 Andrei Alexandrescu

Summary

1. The `scope` statement

2. Built-in arrays

© 2012 Andrei Alexandrescu

Summary

1. The `scope` statement
2. Built-in arrays
3. Compile-time evaluation/`mixin`

© 2012 Andrei Alexandrescu