

最近のD言語の話題など

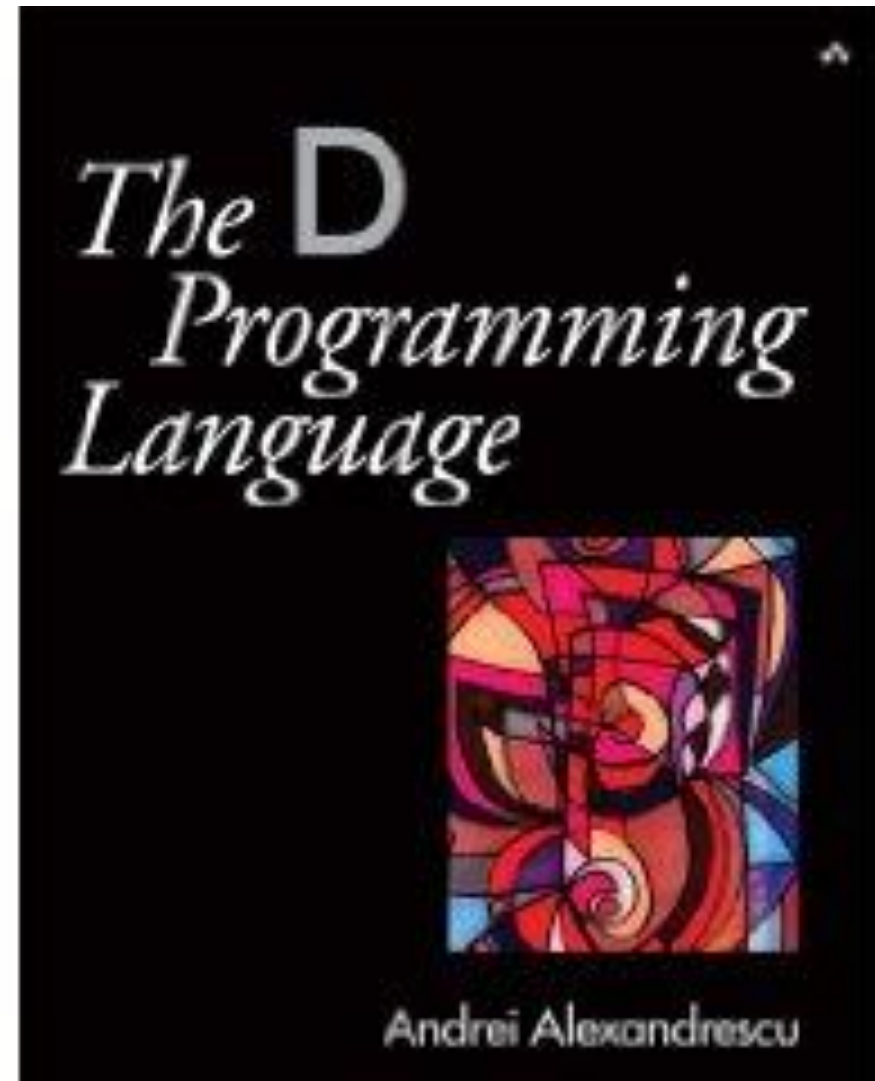
言語雑談会 Jan. 2010

k.inaba

<http://www.kmonos.net/>

初の公式なD言語本が出る！

- ・ 2009年10月に発売
- ・ …されず、(延期の)+ 結果
- ・ 2010年5月発売予定



最新ニュース！

宛先: kiki@kmonos.net <kiki@kmonos.net>

日付: 2010/01/24 01:47:07

Cc: order-update@amazon.com <order-update@amazon.com>

送信者: order-update@amazon.com <order-update@amazon.com>

件名: Your Amazon.com order

Hello from Amazon.com.

We're writing about the order you placed on June 14 2009 (unfortunately, the release date for the item(s) listed below has changed. We're providing you with a new delivery estimate based on the new release date.)

Andrei Alexandrescu "The D Programming Language"

Estimated arrival date: June 28 2010 - July 07 2010

We apologize for the inconvenience caused by this delay.

リリーススケジュール

D 1.0 (2007年1月)

- 言語仕様 Fix。安定版ブランチ

D 2.0 (2007年6月)

- 言語拡張しまくるぜブランチ設立
- 現在： D 2.039 (2010年1月)

D 3.0 (2010年5月を予定)

- TDPL本脱稿時点で D2 の仕様を固定
- 新・言語拡張しまくるぜブランチ = D3

現在：ここ2年の新機能

メタメタしたい

opDispatch

CTFEの改善

auto return
auto ref

range

inout

pure, nothrow

その他

DMDのソース公開

Tangoとランタイム統合

final switch, case range

^^演算子

契約継承

Fiber

配列演算

shared

並列並行したい

未来? : *Unofficial D wish list*

- 213 Stack tracing (#26)
- 203 Reflection API (#6)
- 133 vectorization (#10)
- 114 Multiple return values (tuples) (#28)
- 103 Multiple opCast per class (#24)
- 97 Debug check for null reference (#52)
- 90 Native AMD64 codegen (#36)
- 80 !in (#44)
- 79 Short syntax for new (#18)
- 77 unit test after compilation (#1)

過去？（消えそうなもの）

- HTMLへの埋め込み

```
<html><body>  
  <code>void main() { ... }</code>  
</code></body></html>
```

- built-in の複素数

```
creal x = 1 + 2i;
```

- typedef ← NEW!

```
alias int HOGE; // int と HOGE は同じ型  
typedef int HOGE; // 違う型
```

D 2.037 (12月, 2009)

opDispatch

～ コンパイルタイム

`method_missing` ～

おさらい

コンパイル時
定数だけを
渡せる引数

普通に
なんでも
渡せる引数

・ Dの関数は
2種類の引数を取れます

```
void print(string s)(string d)
{
  writeln(s);  writeln(d);
}
```

```
print!( "hello" )( readln() );
```

コンパイル時引数は !() で渡す

opDispatch とは？

- ・ 存在しないメソッド

obj.xxx(...)

を呼ぼうとしたら、

コンパイルエラー…にせず

obj.opDispatch! (“xxx”)(…)

を呼び

(opDispatchもなければエラー)

opDispatch とは？

```
class Abc {  
    void foo() {}  
    void opDispatch(string s)(){}  
}
```

```
abcObj.foo();
```

```
abcObj.bar();
```

```
→ abcObj.opDispatch("bar")();
```

```
abcObj.buz(12, 34);
```

```
→ abcObj.opDispatch("buz")(12, 34);
```

つかいみち？

car

cdr

cdddddddar

```
class List(T) {  
    T car; List!(T) cdr;
```

```
    auto opDispatch(string s)()  
        if( s[0]==‘c’ && s[$-1]==‘r’  
            && (s[1]==‘a’ || s[1]==‘d’) )  
    {  
        static if( s[1]==‘a’ )  
            return mixin(“car.c”~s[2..$]);  
        else  
            return mixin(“cdr.c”~s[2..$]);  
    }  
}
```

つかいみち

・RPC的なものとか

```
class WebAPI {  
    bool opDispatch(string s, T...  
                    (T params)  
    { return http.send(  
        "http://example.com/api/" ~ s  
        ~ "/"  
        ~ params.map(to!string).join(",")  
    ); }}
```

つかいみち



・スパイ大作戦！

- 標準ライブラリに↓という関数がある

```
void reverse(Range)(Range r);
```

※おさらい：Dの「コンパイル時引数」には「型」を渡せる。= C++のtemplate

- でも、Range にどういう型なら渡せるのか、（この発表を最後まで聞かないと）わからない。困った！

つかいみち



・スパイ大作戦！

```
class Spy {
    BH opDispatch(string s, T...)(T _)
        {... show!(s,T) ...}
    class BH {
        BH opDispatch(string s, T...)(T _) {...}
    }
}
reverse( new Spy );
```

```
empty()
front()
back()
popFront()
popBack()
```

おまけ: 非存在メソッド 捕捉の歴史

• opDot

D 2.013 (4月, 2008) → 消滅?

```
class A { B b; B opDot(){return b;} }  
class B { void bar(){} }  
(new A).bar(); → (new A).opDot().bar
```

• alias this

D 2.027 (3月, 2009)

```
class A { B b; alias b this; }  
class B { void bar(){} }  
(new A).bar(); → (new A).b.bar
```

- ・ 楽しい
- ・ 演算子も opBinary(“+”) や opUnary(“!”) で捕まえたいという一派がいるらしい
- ・ 僕はむしろメンバクラスを捕まえたいです

D 2.038 (12月, 2009)

inout (T)

～ Qualifier多相
(のようなもの) ～

Qualifier 多相とは

- Windows API にこんな関数があります。

```
char* CharNext( const char* p );
```

- 現在の文字コードで1文字、ポインタを進める
- 明らかに型がおかしい
- 本当はこうしたかったに違いない

```
char* CharNext( char* p );  
const char* CharNext( const char* p );
```

Qualifier 多相とは

- 本当はこうしたかったに違いない

```
char*      CharNext( char* p );  
const char* CharNext( const char* p );
```

でも、こうするには、全く同じ実装を
2カ所に書かないといけない
→ そこで Qualifier 多相!

```
Q char* CharNext<Q>( Q char* p )  
{ return p+1; }   ※構文と実装はイメージです
```

Dの場合

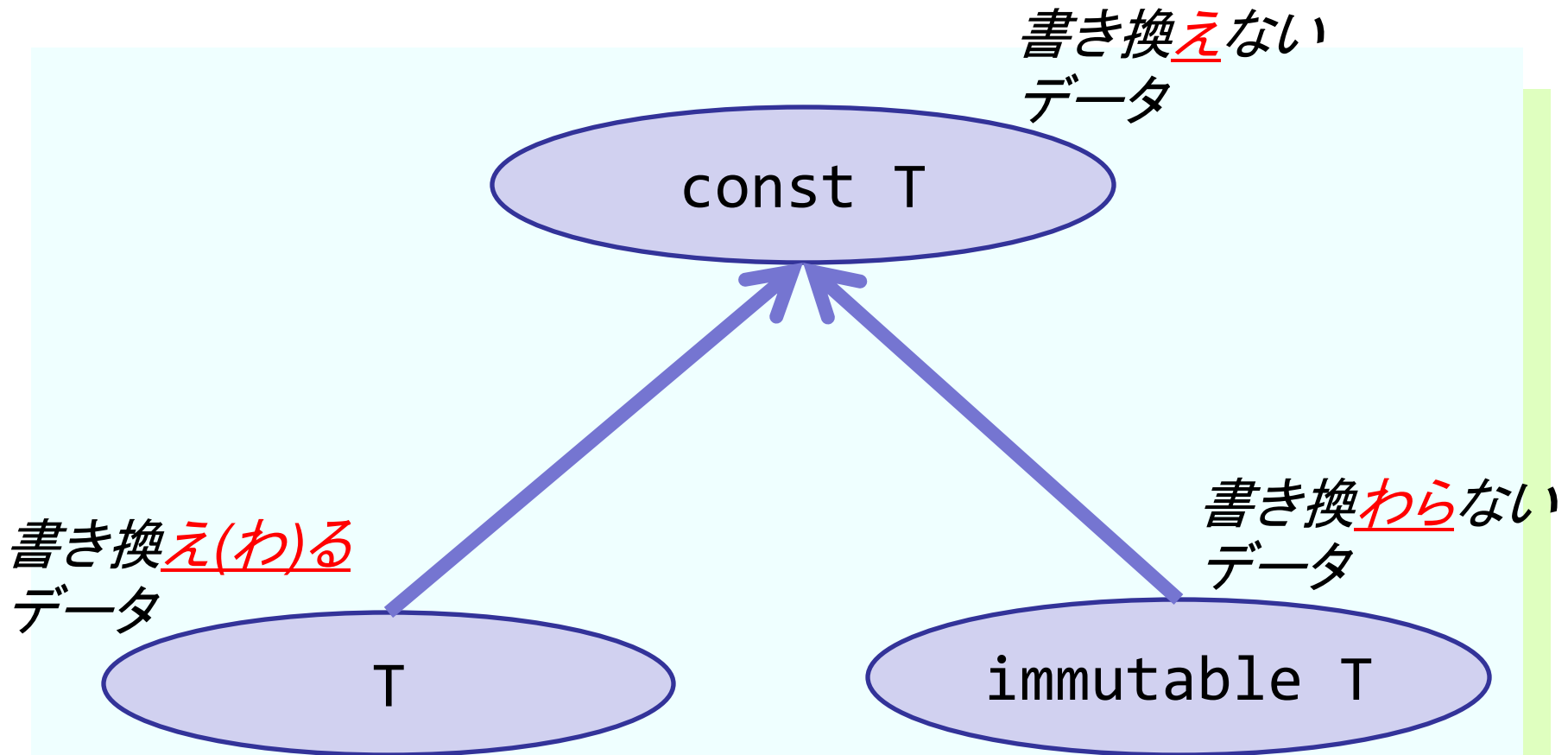
```
inout(char*) CharNext( inout(char*) p );
```

```
const(char*) cptr = "hoge".ptr;  
CharNext(cptr); // 返値は const(char*)
```

```
char* mptr = "hoge".dup.ptr;  
CharNext(mptr); // 返値は char*
```

```
immutable(char*) iptr = "hoge".ptr;  
CharNext(iptr); // 返値は immutable(char*)
```

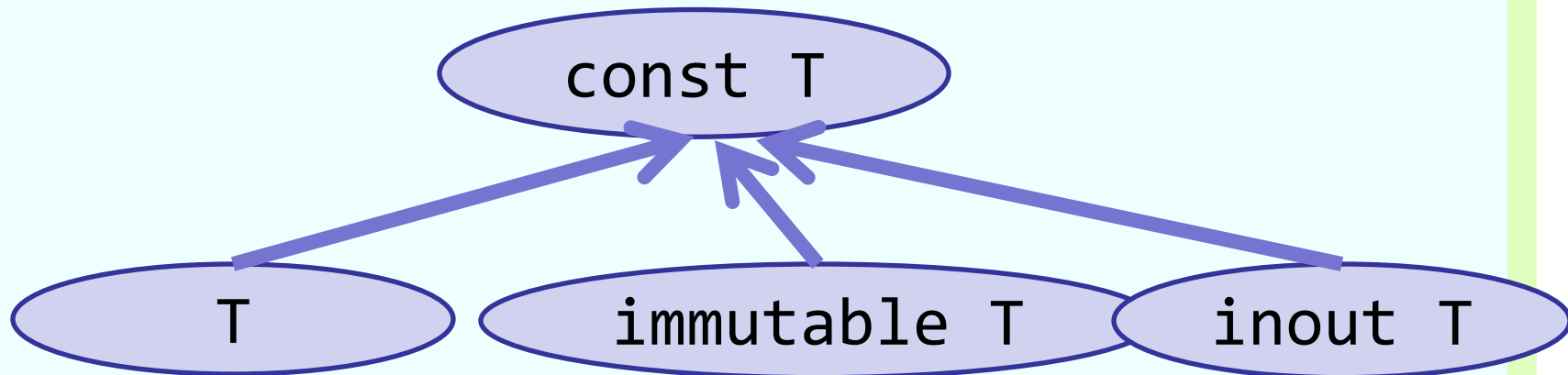
おさらい：Dのconst階層



D 2.020 (10月, 2008)
invariant → immutable

型チェックの実装

- ・ inoutつき関数の中は



のつもりで型チェック

- ・ inout関数の呼び出しは、引数の修飾を返値の修飾にする

- いい具合に手を抜いた実装で面白いと思う
- inout という名前は不評かい？

D 2.030 (5月, 2009)

shared (T)

～ グローバル変数が
TLS になった ～

Global変数はスレッドローカル

```
int global = 100;
void main() {
    global = 200;
    (new Thread({
        writeln( global ); // 100
        global = 300;
        writeln( global ); // 300
    }).start;
    thread_joinAll();
    writeln( global ); // 200
}
```

スレッドローカルに
したくない時は

shared

で明示的に修飾

shared(int) と宣言れば共有

```
shared(int) global = 100;
void main() {
    global = 200;
    (new Thread({
        writeln( cast(const)global ); // 200
        global = 300;
        writeln( cast(const)global ); // 300
    }).start;
    thread_joinAll();
    writeln( cast(const)global ); // 300
}
```

なぜか
そのままでは
型チェック
とおらない

□ 一カ変数
は共有

ローカル変数も共有

```
void main() {  
    int local = 200;  
    (new Thread({  
        writeln( local ); // 200  
        global = 300;  
        writeln( local ); // 300  
    }).start;  
    thread_joinAll();  
    writeln( local ); // 300  
}
```

感想

- ・ まだいまいち、shared で何がしたいのかよくわからない
- ・ shared回りの型付けもやや謎
- ・ 今後ちゃんとすると思う

D 2.029 (4月, 2009)

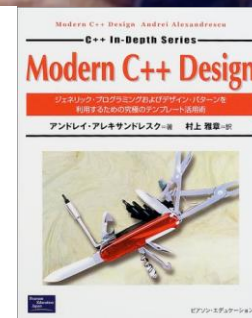
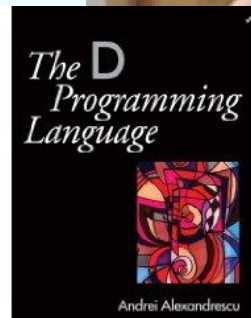
Range ☆ Algorithm

~ Iterators must go ~

A. Alexandrescu

Iterator は
腹を切って
死ぬべきで
ある。

(BoostCon'09
Keynote)



- ・ 標準アルゴリズムライブラリ
 - `std.algorithm`
 - ・ (C++の `<algorithm>` 相当)
 - が
 - ・ イテレータベース から
 - ・ Rangeベース になりました

おさらい

- ・ 内部イテレータ (≈列挙関数)

```
[1,2,3].each{ |n| print n }
```

- ・ 外部イテレータ (≈列挙オブジェクト)

```
x = [1,2,3].begin()
```

```
while( x.hasNext ) print x.next
```

- ・ C++/D のイテレータ (≈ポインタ)

```
x,y = [1,2,3].begin_end()
```

```
while( x < y ) swap(*x++, *--y);
```

イテレータの使用例

```
vector<int> vec = ...;
// vecの全体を破壊的にソート
sort( vec.begin(), vec.end() );
// 先頭100個の中に42があるか二分探索
lower_bound( vec.begin(),
             vec.begin()+100, 42 );
// 末尾から見て最初の123を線形探索
find( vec.rbegin(), vec.rend(),
      123 );
```

おおざっぱに言うと

- C++/D の Range

(≈スライス) (≈ポインタのペア)

```
// vecの全体をソート
```

```
sort( vec );
```

```
// 先頭100個の中に42があるか二分探索
```

```
lower_bound( vec[0..100], 42 );
```

```
// 末尾から見て最初の123を線形探索
```

```
find( retro(vec), 123 );
```

内部イテレータと違うの？

- ・ zip とか普通に書ける

```
zipWith(R,F)(R r1, R r2, F f) {  
  while(!r1.empty && !r2.empty)  
    f(r1.popFront, r2.popFront);  
}
```

```
int[]    x = [ 1 , 4 , 3 ];  
string[] y = ["a", "b", "c"];  
sort!(`a.at!0 < b.at!0`)( zip(x,y) );  
// x == [ 1 , 3 , 4 ]  
// y == ["a", "c", "b"]
```

<InputRange>

.empty

.front

.popFront

<OutputRange>

.put

<ForwardRange>

= コピーできる

<BidirectionalRange>

.back

.popBack

<RandomAccessRange>

operator []

C++/Dイテレータとの違いは？

- ・ コンテナ全部の処理に `begin`, `end` を毎回書かなくていい
- ・ 複数の反復の合成 (`zip`, `concat`, ...) が自然に書ける
- ・ 一部インタフェイスがキモい

一部インターフェイス

・イテレータの場合

```
// 最初の42を指すイテレータ
it = find( vec.begin(), vec.end(), 42 );
// それより右の最小値
int R = *min_element( it, vec.end() );
// それより左の最大値
int L = *max_element( vec.begin(), it );
```

一部インターフェイス

・ Range の場合

```
// 最初の42から末尾までを表すRange
```

```
r = find( vec, 42 );
```

```
// それより右の最小値
```

```
int R = min( r );
```

```
// それより左の最大値
```

```
int L = max(  
    vec[0 .. vec.length-r.length] );
```

感想

- ・ C++風イテレータはすごく便利
- ・ レンジは基本的にもっと便利
 - foreach文も使える *D 2.021 (10月, 2008)*
- ・ でもイテレータ返す系
アルゴリズムが気持ち悪い
→ どうなるのでしょうか

処理系・開発環境 の話題

年表（月表？）

- LDC (LLVM D Compiler)
D 1.035相当 (1月, 2009)
- MacOSX版DMD
D 2.025 (2月, 2009)
- DMD のソースを完全に公開
D 2.026 (3月, 2009)
- Descent 0.5.6 (コパイル時デバツカ)
D 1.045相当 (5月, 2009)
- dmd -X (JSON出力)
D 2.035 (10月, 2009)

昔からやるやる
言っていた機能
の実装

細かすぎてD!げんがーにしか 伝わらない

D 2.037 (12月, 2009)

- array.length への演算代入

```
int[] a = [1, 2, 3];  
a.length -= 2;  
assert( a == [1] );
```

- 昔 → `a.length = a.length - 2;`
- ユーザ定義プロパティはまだ

配列演算

D 2.018 (8月, 2008)

```
int[] a = [1, 2, 3];  
int[] b = [4, 5, 6];  
int[] c = new int[3];  
c[] = a[] + b[]*3;
```

- ・ SIMD命令で頑張った機械語を吐く

契約 (Contract) の継承

D 2.033 (10月, 2009)

```
class Base {  
  int method(int x)  
    in          { assert(x>=0); }  
    out(int r) { assert(r>=0); }  
    body        { ... }  
}
```

in は or で結合
out は and で結合
されるようになった

```
class Derived : Base {  
  int method(int x)  
    in          { assert(x%2==0); }  
    out(int r) { assert(r%2==0); }  
    body        { ... }  
}
```

- ・ そんなことより interface に 契約を書かせてほしい…

```
interface Base {  
    int method(int x)  
        in          { assert(x>=0); }  
        out(int r) { assert(r>=0); };  
} // 書けない↑
```

以上です。

メタメタしたい

opDispatch

CTFEの改善

auto return
auto ref

range

inout

pure, nothrow

その他

DMDのソース公開

Tangoとランタイム統合

final switch, case range

^^演算子

契約継承

Fiber

配列演算

shared

並列並行したい