

# WORD

2009.1

From College of Information Science

8

今年のバレンタインは  
これでライバルと差をつけろ！

キモオタでも出来る NAPT 越え

あきたこまち

ハジメルハスケル 2nd Season

嗚呼、オーディオ

ねるねるねるねるねる  
VG  
ねるねるねるねるねる

新連載

multicast

XNA で遊ぼう！

JA ウホッ！いいお※号

# JA ウホッ！いいお※号

## 目次

ハジメル Haskell - 2nd Season -	P. 3
LINQ in C# 3.0	P. 13
嗚呼、オーディオ	P. 26
第1回 XNAで遊ぼう	P. 35
multicast	P. 39
キモオタでも出来るNAPT越え	P. 47
C75 参加顛末記	P. 58
ねるねるねるねるねる。	P. 71
あきたこまち	P. 78
お知らせ	P. 82
coins-admin メンバ募集！	P. 84
【特報】WORD 30周年記念	P. 85
yet another wordpress	P. 86

## ハジメル Haskell そのに

文 編集部 goth

### ——ハジメル Haskell——

Haskell 記事の第二回目です。

HelloWorld がでるまでに実に 10 ページ以上もかけてしまった前回に引き続き Haskell について書いていきたいと思います。

と、思っていたのですが、諸処の事情<sup>\*1</sup> から、ちょっと横道にそれで、リストのお話となります。

### ——組(タプル)——

で、リストのお話とかいいながらさっそくズレて、タプルの話から入りたいと思います。決して前回のに書き忘れたから今回書く、とかそういう理由ではないです<sup>\*2</sup>。きっと。

さて、タプルですが、組とも呼ばれる構造で、リスト同様に複数の値をまとめることができます。リストは同じ型のものを好きなだけ並べることができましたが、タプルでは型を問わずまとめることができます。その一方で拡張したり、マッピングしたりといった操作は用意されておらず、リストとは全く異なることとなる目的で使われます。C 言語でいう構造体のような働きをするのですが、まあ、四の五の言うよりは見てもらった方が早いと思います。

```
$ ghci
GHC, version 6.10.1*3: http://www.haskell.org/ghc/ :? for help
Loading package ghc-prim... linking... done.
Loading package integer... linking... done.
Loading package base... linking... done.
Prelude> (1,2) -- 1と2の二つ組(ダブル/2-tuple)を作る。
(1,2)
Prelude> (1,2,3) -- 三つ組み(トリプル/3-tuple)も作れる。
(1,2,3)
Prelude> (1,"String") -- 型は異なってもよい。
(1,"String")
```

まあ、見てもあまり面白いものではないのでこのへんにしておきましょう。

タプルはいくらでも長くすることができますが、だいたい 2-tuple か 3-tuple ぐらいまでしか使わないと思います。それより複雑な場合は「代数的データ型」というものを使うのが

\*1 諸処の事情: お察しください

\*2 そういう理由ではないです: ということにしておいてください

\*3 version 6.10.1: ヴァージョンがいきなりあがった…

## ハジメル Haskell -2nd Season-

一般的なようです。

で、ただまとめるだけ…というのもツマラナイので、タプルを使う関数をいくつか見てみましょう。

```
Prelude> fst (1, "String") -- fst :: (a, b) -> a
1
Prelude> snd (1, "String") -- snd :: (a, b) -> b
"String"
Prelude> divMod 13 5 -- divMod :: (Integral a) => a -> a -> (a, a)
(2, 3)
Prelude> zip [1, 2, 3] ["one", "two", "three"] -- zip :: [a] -> [b] -> [(a, b)]
[(1, "one"), (2, "two"), (3, "three")]
Prelude> unzip [(1, "one"), (2, "two"), (3, "three")] -- unzip :: [(a, b)] ->
([a], [b])
([1, 2, 3], ["one", "two", "three"])
```

"**fst**", "**snd**" はそれぞれ二つ組から左の要素、右の要素を取り出す関数です。実際には パターンマッチ という機能で同様のことができてしまうので出現頻度はそんなに高くないです。3-tuple(三つ組み)の場合はどうするのか? というと、"**fst**", "**snd**" のような関数は無く、件のパターンマッチでやるしかないです。

"**divMod**" は前回でてきた"**div**" と"**mod**" の組み合わせです。念のため復習しておくと、"**div**" は整数同士の割り算の商、"**mod**" は余剰を返します。"**divMod**" はその両方、つまり「商と余剰をまとめて欲しい!」というときに使う贅沢な<sup>4</sup> 関数です。

このような値を複数返す関数は、C 言語など参照渡しだの構造体だのいろいろメンドウなのですが、Haskell ではタプルを使うことで簡単に作れてしまいます。

"**zip**" は見ていただければ分かると思うのですが、二つのリストを先頭から一個ずつタプルでくっつける関数です。何に使うんだ?とか思われるかもしれません、意外に使う機会が多い関数です。

"**unzip**" はその真逆。**zip**されたリストを二つのリストに戻します。二つのリストを返す際に、先ほどの"**divMod**" と同様にタプル使ってます。

そんなこんなでタプルのお話でした。難しい概念でもないし、使ってるうちに手足のように使えるようになってると思います。

### ——リストの数列表記と無限リスト・遅延評価——

さて、リストの話に入ります。

Haskell はリスト処理に長けていると言われる言語ですが、その処理能力をさらに便利に

\*4 贅沢な: 贅沢かどうかは別として、片方計算する際に他方も同時にでますし、理にかなった関数ではあると思います。ちなみに内部的には、div と mod は divMod を使って実装されています

するのが数列表記と内包表記です。数列表記は下の例を見ていただければ分かると思いま  
すが、連続する数の列を簡単に記述できる機能です。

```
 Prelude> [1..10]
 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
 Prelude> [-5..5]
 [-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5]
 Prelude> [1, 3..10]
 [1, 3, 5, 7, 9]
```

最後の例では単純に連続した数列ではなく、世に言う等差数列を生成しています。

また、数列表記は数字だけではなく、文字も同様に書くことができます。

```
Prelude> ['a'.. 'z']
"abcdefghijklmnopqrstuvwxyz"
Prelude> ['a', 'c'.. 'z']
"acegi klmnopqrstuvwxyz"
```

より正確には「Enum クラス」に属する型<sup>\*5</sup>ならこの数列表記が可能となります。

一般的に、数列表記は"[ (初項) ,(第二項目)..(末項)]"のように書けます。初項は省略で  
きませんが、第二項目は省略できます。第二項目が省略された場合には初項から1刻みに、  
省略されなかった場合には初項との差刻みで数列を生成していきます。

では、末項が省略されたらなるでしょう？

試しにやってみると…

```
Prelude> [1..]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27,
, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 5
1, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72,
```

のように、無限にリストが生成されていきます。これを無限リストと言います。

このままずっと止まらない<sup>\*6</sup>ので、飽きたところで Ctrl-C を押して止めてあげてください。筆者は 959081 ぐらいまで眺めてたのですが、さすがに飽きました。

さて、「無限」と聞いて思い浮かぶ単語と言えば「無限ループ」。課題などでこれに苦し  
められた、という方は少なくないと思います。そんなこんなで、つい無限と聞くとネガテ  
ィブな印象を受けがちですが、Haskell における無限リストは非常に強力なツールとして働  
きます。

そして、その無限リストを下支えする機能が遅延評価と呼ばれる機能です。

まず、下の例を見てみましょう。

\*5 "Enum クラス"に属する型: Int や Float のほか、Bool も実は Enum クラスに属している。":info Enum"で調べられる

\*6 止まらない: 当然メモリ等の関係である程度のところで強制終了しますが…

## ハジメル Haskell -2nd Season-

```
 Prelude> take 10 [1..]
 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

"take" はリストの先頭から指定された個数分の要素を取り出してくる関数です。直感的にはなんら不思議の無い実行結果なのですが、実は奇妙なことが起きます。というのも、"[1..]" の部分は「終わらない関数」であるのに、"take 10 [1..]" 全体ではちゃんと停止しています。

例えば C 言語などで、

```
int bot( int x ) {
    return bot( x );
}
int main() {
    printf( "result: %d\n", bot( 42 ) );
}
```

のように書いてしまうとこのプログラムは止まること無く、やがてエラーで強制終了してしまいます。これは当然"bot" 関数が「止まらない関数<sup>7</sup>」であり、"main" の中でそれを呼んでしまってることに起因する訳ですが、上の Haskell の例とはどう違うのでしょうか？

そもそも、無限リストなんてものはどうやって作るのでしょうか？

その答えこそが遅延評価なのです。

遅延評価を簡単に説明すると「本当に必要になるまでは放っておく」です。筆者の人生訓に似ていますが、Haskell ではこの考え方が非常に巧妙、かつ大胆に取り入れられているのです。

無限リストの仕組みを知るために、"[x..]" を自分で書いてみましょう。関数の定義の仕方はまだ説明していなかった<sup>8</sup> ので、軽く説明を加えておきます。

```
Prelude> let myEnumFrom x = x : (myEnumFrom(x+1))
Prelude> take 10 (myEnumFrom 1)
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

随分あっさりとしてますが、これで完成品です。

一行目が関数定義となるのですが、関数名が "myEnumFrom" となっています。実は "enumFrom x" という "[x..]" とほぼ等価のことをする関数が既にあるのです。ちなみに "[x, y..]" は "enumFromThen x y"、"[x..z]" は "enumFromTo"、"[x, y..z]" は "enumFromThenTo x y z" とほぼ等価となります。

\*7 止まらない関数: ちなみにこの関数は、論理的な不備がある(値が定まらない)ので、Haskell で書いてもやはり正常終了することはありません

\*8 まだ説明していなかった: べ、べつに、文章構成を間違えたとか、そういうんじゃないからねっ！

\*9 (x+1): ここでは "(x+1)" と記述していますが、実際の "enumFrom" では "succ x" という関数を使っています。"succ" はより汎用的な "+1" で、Char や Bool などの "Enum" クラスの型に適応できる関数です。例としては "succ 'a'" は 'b' に、"succ 'b'" は 'c' … という風になります。

さて、本題に入ります。

"**let**" は関数を定義するために必要なキーワードで、関数定義は"**let** (関数名) (引数)  
= (関数本体)" となります。

C 言語風に書くと次のような感じになります。

```
myEnumFrom( x ) {
    x : (myEnumFrom(x+1))
}
```

引数や返り値に型の宣言<sup>\*10</sup> がついていなかったり、"**return**" が無かったりとやや違和感を覚えるかもしれません、を無理やり C 言語風にしたのものなので勘弁していただければと思います。

関数本体部分にある"::" は前回出てきたリストを構成する演算子です。この場合"x" と "**myEnumFrom (x+1)**" をリストとしてくっつけています。後者の"**myEnumFrom (x+1)**" はその関数自身を再び呼び出す、世に言うところの"**再帰関数**" というやつです。

この関数の挙動を追ってみましょう。

<b>myEnumFrom 1</b>	-- 1 を引数に呼び出してみる。
1 : (myEnumFrom(1+1))	-- 関数を展開して、
1 : (myEnumFrom2)	-- 計算をする。
1 : (2 : myEnumFrom(2+1))	-- 再び関数を展開して、
1 : (2 : myEnumFrom3)	-- 続けていくと
1 : (2 : (3 : (4 : (5 : ...	-- たしかに無限リストになっていそうだ。

"::" によるリストの記述に慣れていないと分かりづらいかもしれません、これで無限リストを生成することができます。

"**take**" も無限リストと同様に再帰を使って定義されています。詳細な定義は省くとして、例えば"**take 3 [1, 2, 3, 4, 5]**" は「与えられたリスト(=[1,2,3,4,5])」を「先頭(=1)」と「それ以外(=[2,3,4,5])」に分離させ、"**1 : (take 2 [2, 3, 4, 5])**" のように書き換えられています。

同様に、"**1 : (take 2 [2, 3, 4, 5])**" は"**1 : (2 : (take 1 [3, 4, 5]))**"、さらには"**1 : (2 : (3 : (take 0 [4, 5])))**" となります。

"**(take 0 [4, 5])**" は与えられたリストを全て捨ててしまい、リストの終端を表す"**[]**" を返し、結果、"**1 : (2 : (3 : []))**" となります。これは"**[1, 2, 3]**" と等価であり、"**[1, 2, 3, 4, 5]**" の先頭三つと一致します。

さて、この無限リストの定義を使って"**take 3 [1..]**" の挙動を調べたいと思います。無限リストの挙動を調べるために、"**[1..]**" の代わりに先ほど定義した"**myEnumFrom 1**" を使っていきます。

まず、"**take 3 (myEnumFrom 1)**" を考えます。"**take**" は与えられたリストを「先頭とそ

---

\*10 型の宣言: Haskell は型を書かなくても、「型推論」という機能で勝手に予測してくれます。まあ、型を書いておくに超したことはないのですが…

## ハジメル Haskell -2nd Season-

「それ以外」に分離しないといけません。ところが、"( myEnumFrom 1)" の形では先頭の要素がなんであるのか、あるいは、それ以降がどうなってるかが分かりません。

そこで、「テーマの先頭要素はなんだ」とばかりに"( myEnumFrom 1)" を展開し、"1 : ( myEnumFrom2 )" という形にします。

さて、ここで注目してほしいのは"1 : ( myEnumFrom 2 )" という形です。この時点で既にすでに、「先頭とそれ以外」の分離ができているのです。つまり、先頭は"1"で、それ以外は"( myEnumFrom2 )" です。

「おうおう、あんちやん、後ろの部分がまだ計算できるやんけ」と言われそうですが、そこは Haskell、必要が無い限り計算は中途半端でよいのです。これが遅延評価の神髄となります。

リストの分離ができてしまえば、あとは同様に続けることができます。

```
take 3 (myEnumFrom1)
take 3 (1 : myEnumFrom2)
1 : (take 2 (myEnumFrom2))
1 : (take 2 (2 : myEnumFrom3))
1 : (2 : (take 1 (myEnumFrom3)))
1 : (2 : (take 1 (3 : myEnumFrom4)))
1 : (2 : (3 : (take 0 (myEnumFrom4))))11
```

あとは先ほどと同じです。

ポイントとなるのは「引数を中途半端な状態で止められる」ということです。このような性質を非正格と呼ぶのですが、まあ、覚えなくてもよいと思います。

逆に、「引数をちゃんと計算しておかないといけない」という性質は正格と呼び、C 言語などは正格な言語となります。正格な言語で先のように"take"を定義してしまうと、引数の無限リストが先に計算されてしまい"take"の処理が一切始まらない、ということになってしまいます。

"take"を含め、前回紹介しきれなかった「リストの一部を取り出す関数」を紹介しておきます。

<sup>11</sup> 1 : (2 : (3 : (take 0 (myEnumFrom 4)))): 説明では簡易のため"1+1"を計算して"2"などとしていますが、実際には遅延評価が働くため、"1 : ((1+1) : ((1+1+1) : []))"となり、画面表示するとき時になって初めて、それぞれの値が計算されます

```

 Prelude> head [1..10] -- 先頭の要素を取ってくる。
1
Prelude> tail [1..10] -- 先頭以外の要素を取ってくる。
[2, 3, 4, 5, 6, 7, 8, 9, 10]
Prelude> take 5 [1..10] -- リストの先頭から n 個の要素を取ってくる。
[1, 2, 3, 4, 5]
Prelude> drop 5 [1..10] -- リストの先頭から n 個以外の要素を取ってくる。
[6, 7, 8, 9, 10]
Prelude> takeWhile (<5) [1..10] -- 先頭から条件に一致する要素を取ってくる。
[1, 2, 3, 4]
Prelude> dropWhile (<5) [1..10] -- 先頭から条件に一致する要素以外を取つてく
る。
[5, 6, 7, 8, 9, 10]

```

これ以外にもいつかありますが、よく使うのはこのあたりでしょうか。

"takeWhile"/"dropWhile"で"(< 5)"という謎の記述がありますが、これはカリー化だと部分適応だとセクションとかいう機能によるものです。

とりあえずは「5 より小さいかどうか？」という意味と考えてもらえば問題はないと思います。同様に"(==5)"は「5 と等しいかどうか？」、「(/=5)"は「5 と等しくないかどうか？」、「(>=5)"は「5 以上かどうか？」となります。

"head"と"tail"が先の"take"の説明で出てきた「先頭とそれ以外」をそれぞれ取得する関数となります。なのですが、タプルの"fst", "snd"と同様にパターンマッチに役目を奪われがちです。

例では"[1..10]"という有限リストを与えていますが、当然、無限リストを与えることもできます。その場合、"head", "take", "takeWhile"では有限リストに切り詰めることができますが、"tail", "drop", "dropWhile"では無限リストのまとまりになります。

そんなわけで、無限リストの処理にはだいたい、"take"ないし"takeWhile"と一緒に使われることが多いようです。

## ——リスト内包表記——

さて、前述の数列表記をさらにグレイトにしてくれるのが内包表記です。まずは下の例を見てみましょう。

```

Prelude> [ 3*x | x <- [1, 2..] ]
[3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, ...]

```

結果を見ればわかるのですが、3 の倍数のリストを得ています。この表記は数学の集合論でいうところの  $\{3x \mid x \in \{1, 2, \dots\}\}$  にあたります。

もう一つ例を挙げてみましょう。

```

Prelude> [ x | x <- [1, 2..12], 12 `mod` x == 0 ]
[1, 2, 3, 4, 6, 12]

```

これも同様に数学的に表すと  $\{x \mid x \in \{1, 2, \dots, 12\}, x \equiv 0 \pmod{12}\}$  となり、12 の約数のリストとなります。

これが内包表記と呼ばれる機能で、数学的表現とほぼ同等にリストを記述することができます。内包表記の一般形は"`[ f x | x <- xs, g x]`"で、このうち"`x <- xs`"のようにリストから"`<-`"で要素を引いてくる部分を「生成部」、"`g x`"のように条件を与える部分を「ガード」と呼びます。これらによって得られるすべての要素に"`f`"を適応します。

先ほどの例では、"`x <- [1, 2..12]`"が生成部、"`12 `mod` x == 0`"がガードとなります。

この生成部やガードは複数指定することができます。ガードを複数指定した場合には、単に条件が足されていきます。つまり、例えば「2 の倍数ではなく、かつ、3 の倍数ではない」数列は"`[ x | x <- [1..], x `mod` 2 /= 0, x `mod` 3 /= 0 ]`"のように書くことができます。

一方、生成部を複数書いた場合には全ての組み合わせが生成されます。生成順序は左から順に、右の生成部を入れ子にするのですが、まあ、言葉で書くより具体例を見た方がわかりやすいでしょう。

```
 Prelude> [(x,y) | x <- [1,2], y <- ['a','b','c']]  
 [(1,'a'), (1,'b'), (1,'c'), (2,'a'), (2,'b'), (2,'c')]
```

数学の集合では順序や重複を無視するので生成順序はありませんが、Haskell のリストでは順序や重複が意味を持ちますので、生成順序には注意が必要です。

さて、この内包表記を使った関数の例を見てみましょう。

```
Prelude> let sieve xs =  
 (head xs) : sieve [ y | y <- (tail xs), y `mod` (head xs) /= 0 ]12
```

またしてもロクな説明もないまま関数定義を出してしまったわけですが、これは何をする関数なのでしょうか？

まず、関数名は"`sieve`"で、引数は"`xs`"です。

関数本体は、引数の"`head`"つまり、「引数のリストの先頭要素」を取得しています。その先頭要素と、「何か」を": "演算子でリストとして連結しているようです。その連結する「何か」に内包表記が使われています。

内包表記の中を見てみると、まず、生成部が"`y <- (tail xs)`"、つまり、「引数の先頭以外の要素」が対象となります。ガード部は"`y `mod` (head xs) /= 0`"、つまり、「引数の先頭要素で割り切れないもの」だけを集めようのです。

結局、内包表記全体では「引数の先頭以外の要素から、引数の先頭要素で割り切れないものだけを集めた」リスト、となります。その内包表記が"`sieve`"関数に再帰的に渡されます。

と、一通りの挙動は分かったのですが、具体的には"`sieve`"は何をやる関数なのでしょうか？その答えは引数に"`[2..]`"という無限リストを与えることで見えてきます。

\*12 紙面の都合上、改行していますが実際に改行する必要はありません

```
 Prelude> take 10 (sieve [2..])
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
```

そう、素数のリストとなります。

関数名の "**sieve**" はふるい(篩)、つまり「エラトステネスのふるい<sup>\*13</sup>」のことです。このアルゴリズムは非常に有名なものなので、C 言語で実装したことがある、という方も多いと思います。

C 言語では配列確保やループなどをメインに実装していきますが、関数型言語、こと Haskellにおいてはリスト中心に考えることが多いようです。

ちなみに、ではありますが、「nまでの素数」と「n個の素数」。同じようにみえますが、C 言語では実装の手間がずいぶん変わります<sup>\*14</sup>。たぶん授業なんかでやるのは前者かと思われますが、もし時間があれば後者も実装してみるとおもしろいかかもしれません。Haskell なら"**take n**" と "**takeWhile (<n)**" の違いだけなのですが<sup>\*15</sup>。

では、どのように素数列が生成されるか挙動を追ってみましょう<sup>\*16</sup>。

```
sieve [2..]
  ↓ sieve を展開する
  (head [2..]) : sieve [ y | y <- (tail [2..]), y `mod` (head [2..]) /= 0 ]
    ↓ head/tail を展開する
  2 : sieve [ y | y <- [3..], y `mod` 2 /= 0 ]
    ↓ 内包表記を処理する
  2 : sieve [3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, ...]
    ↓ も一回 sieve と head/tail を展開する
  2 : 3 : sieve [ y | y <- [5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, ...], y `mod` 3 /= 0 ]
  2 : 3 : sieve [5, 7, 11, 13, 17, 19, 23, 25, ...]
  2 : 3 : 5 : sieve [ y | y <- [7, 11, 13, 17, 19, 23, 25, ...], y `mod` 5 /= 0 ]
  2 : 3 : 5 : sieve [7, 11, 13, 17, 19, 23, ...]
  2 : 3 : 5 : 7 : sieve [ y | y <- [11, 13, 17, 19, 23, ...], y `mod` 7 /= 0 ]
  2 : 3 : 5 : 7 : sieve [11, 13, 17, 19, 23, ...]
  略
  2 : 3 : 5 : 7 : 11 : 13 : 17 : 19 : 23 : 29 : ...
```

と、定義どおり順に素数を使ってふるいがけが行われているのがわかります。

上で出てくる"..." は数列表記のものではなく、単に無限リストの後ろの方を省略しているだけです。"**sieve [2..25]**" をすれば 25までの素数が得られるはずなのですが、今

\*13 エラトステネスのふるい: 素数を求める有名なアルゴリズム。 Wikipedia にアニメーション画像つきで説明があったので興味がある方は調べてみるとよいかも

\*14 変わります: たぶん変わります。少なくとも筆者は、後者の方が複雑になりました。

\*15 だけなのですが: これを書きたかったためだけに「エラトステネスのふるい」を例題に使いました。調子に乗ってスマセン…

\*16 挙動を追ってみましょう: 実際には遅延評価があるため、より複雑になりますが、簡単のため適当に関数を展開しています

## ハジメル Haskell -2nd Season-

回の"**si eve**" の定義には少し不備がある<sup>\*17</sup> のでエラーが起きてしまいます。まあ、25まで  
の素数が欲しい、という場合には"**t akeWh i e (<25) ( si eve [ 2.. ] )**" すればよいだけな  
ので、特に問題はないのですが…。

内包表記は非常に強力なツールなので、これから先もガンガン使っていこうと思います。

基本は数学の集合と同じ書き方なので混乱は少ないとは思いますが、そもそも数学が苦手、という方もがんばって使いこなしてみてください。

---

—終わりに—

そんなこんなで Haskell 紹介の第二回目、いかがだったでしょうか。

まだ説明していない関数定義の話を持ち込んでしまったり、代数的データ型やパターンマッチなどの未説明な単語が残っていたりと、読みづらい文章になってしまった感が否めないのですが、ご容赦いただければと思います。

次回以降、詳細な関数定義の説明や、今回説明できなかった色々な概念・機能、また、実際の Haskell のコードなどなど載せられれば、と思っております。

筆者自身もまだまだ Haskell 勉強中の身。なにか、ご指摘・ご意見等があればお気軽にご連絡ください。

i0611238@coins.tsukuba.ac.jp

---

\*17 不備がある: [](空リスト)に対する処理が書いてありません。次号あたりで関数定義について書く予定なので、完全な"sieve"関数に挑戦してみてください

# LINQ in C# 3.0

文 編集部 いのひろ

こんにちは。前号<sup>\*1</sup>では「MAX プリン@雙峰祭」の記事で修羅場っていた為、「LINQ in C# 3.0」が書けませんでした。いのひろです。

さて前回は XML 文書から必要なものを抽出しましたが、さらに実用度を上げて、「並べ替えなどの処理を加えてから出力する」、「XML として出力する」などをやってみたいと思います。

今回利用するデータは、「国土交通省国土計画局（GIS）<sup>\*2</sup>」が公開している「国土数値情報ダウンロードサービス」の「日本の原子力発電所」を、使いやすく加工して利用します。こんな感じの XML です<sup>\*3</sup>。

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <Plants>
3   <Plant>
4     <Name>川内原子力</Name>
5     <Belong>九州電力</Belong>
6     <Power>89000</Power>
7   </Plant>
8   <Plant>
9     <Name>川内原子力</Name>
10    <Belong>九州電力</Belong>
11    <Power>89000</Power>
12  </Plant>
13  <!-- 省略 -->
```

要素名は以下の情報を表しています。

- Name: 発電所名
- Belong: 所属する電気会社
- Power: 最大出力 (キロワット)

---

<sup>\*1</sup> 第 7 号「WORD の原価は 4 兆ジンバブエドル」。

<sup>\*2</sup> <http://www.mlit.go.jp/kokudokeikaku/gis/>

<sup>\*3</sup> こんな XML をみてニヤニヤしているのは私だけではないはず。

## LINQ in C# 3.0

### LINQによるXMLの生成

まず復習も兼ねて、発電所名（Name要素の値）を列挙してみましょう。

```
1 class Program
2 {
3     static void Main( string[] args )
4     {
5         XDocument doc = XDocument.Load( @"plants.xml" );
6         var all = from plant in doc.Root.Elements()
7                   select plant.Element( "Name" ).Value;
8
9         foreach ( string name in all )
10            Console.WriteLine( name );
11    }
12 }
```

リスト 1

出力結果。

```
川内原子力
川内原子力
玄海原子力
玄海原子力
玄海原子力
// 省略
```

しかしこれだと複数の発電機がある変電所だと、名前が重複したまま列挙されてしまうので、あまり嬉しくありません。こういうときは、

```
1 all = all.Distinct();
```

と書く事で解決できます。Distinct メソッドはコレクションの中から重複を取り除くメソッドです。

```
川内原子力
玄海原子力
伊方
浜岡原子力
島根原子力
```

```
// 省略
```

さて、Vol.1 で「LINQ」に用いる「クエリ式」は、実は実行時に「メソッド呼び出し」に変換されて解釈されると書きました。例えば今のコードだと、

```
1 XDocument doc = XDocument.Load( @"plants.xml" );
2 var all = doc.Root.Elements()
3     .Select( name => name.Element( "Name" ).Value )
4     .Distinct();
```

## リスト 2

と書く事ができ、リスト 1 は一度リスト 2 のように変換されてから実際に実行されます。ここでの「.Select()」(3 行目)や「.Distinct()」(4 行目) を「標準クエリ演算子」と呼びます。また、この構文自体は第 1 回で説明した「拡張メソッド」と呼ばれるものですが、これらはあらかじめ定義されているメソッドです。

LINQ は、リスト 1 のような「クエリ式構文」とリスト 2 のような「標準クエリ演算子構文」の 2 つの構文で書くことができます。しかし「クエリ式構文」は「より SQL クエリのように表現するためのもの」である為、すべての標準クエリ演算子をサポートしている訳ではありません。

この為、リスト 2 のコード（クエリ演算子構文）では重複を取り除く処理も一つのクエリステートメントの中で実現していますが、リスト 1 のコード（クエリ式構文）ではクエリステートメントとは別のステートメントで重複を取り除いています（「`all = all.Distinct();`」）。

どちらの構文で書くかというのは人それぞれだと思います。とりあえずこれまで C#を書いてきた人にとっては、これらの構文は非常に「気持ち悪い」と感じるのではないでしょうか。いろいろなフィルタや集計といった処理を 1 行で書くことができるわけですから。

### 昇順／降順

それでは早速データに処理を加えてみましょう。まずはソートです。原子力発電所の出力（Power）順に並び替えてみましょう。出力としては「原子力発電所の名前（出力）」というのを目指します。

まずはクエリ式で書いてみます。

```
1 XDocument doc = XDocument.Load( @"plants.xml" );
2 var query = from plant in doc.Root.Elements()
3     orderby int.Parse( plant.Element( "Power" ).Value )
4     select new
5     {
```

## LINQ in C# 3.0

```
6             Name = plant.Element( "Name" ).Value,
7             Power = plant.Element( "Power" ).Value
8         };
9
10    all = all.Distinct();
11
12    foreach ( var element in query )
13        Console.WriteLine( "{0}({1})", element.Name, element.Power );
```

リスト 3

昇順／降順を実現しているのは「orderby 句」です。SQL クエリを扱ったことがある人ならすぐにわかると思いますが、ある値を昇順に DB からレコードを取得するときに用います。今回は「Power 要素の値<sup>\*1</sup>」を評価し、昇順で取得して、発電所名と一緒に出力しています。出力結果は以下のようになります。

```
東海(16600)
美浜(34000)
敦賀(35700)
島根原子力(46000)
福島第一原子力(46000)
// 省略
```

降順に出力する場合は、「orderby 句」の行の最後に「descending」と宣言します。

```
orderby int.Parse( plant.Element( "Power" ).Value ) descending
```

```
柏崎刈羽原子力(135600)
玄海原子力(118000)
大飯(118000)
大飯(117500)
敦賀(116000)
// 省略
```

\*1 単純に「plant.Element( "Power" ).Value」としてしまうと String(文字列) 型として扱われてしまうので、int (Int32) 型にパースする必要があります。

「orderby 句」を用いることで簡単にソートすることができました。まとめると、単純に「orderby」と書いたときと、「ascending」を宣言したときは昇順として処理され、「descending」と宣言した場合は降順として列挙されます。

新しい構文が出てきました。4～7行目です。これは「匿名型」と呼ばれるもので、あらかじめ定義されていない構造体を利用するすることができます。ここでは Select するときに「Name（発電所名）」と「Power（出力）」をプロパティに持つ宣言されていない（名前のない=匿名な）クラスをコンパイラが勝手に作ってくれます。わざわざ値を格納するクラスを宣言しておかなくてよいと言うことです。

さて、これを「標準クエリ演算子」で書き直すとどうなるでしょうか。

```

1 XDocument doc = XDocument.Load( @"plants.xml" );
2 var query = doc.Root.Elements()
3     .OrderBy( plant => int.Parse( plant.Element( "Power" ).Value ) )
4     .Select( plant => new
5     {
6         Name = plant.Element( "Name" ).Value,
7         Power = plant.Element( "Power" ).Value
8     } )
9     .Distinct();
10
11 foreach ( var element in query )
12     Console.WriteLine( "{0}({1})", element.Name, element.Power );

```

リスト 4

気持ち悪いコードですね<sup>\*</sup>。ここでも4～8行目に匿名型が用いられています。

繰り返しになりますが、やっていることはリスト 3 のコードと同等で、出力も同じなので省略します。次にクエリ演算子での降順ですが、「orderby 句（3行目）」を書き換える必要があります。

```
.OrderByDescending( plant => int.Parse( plant.Element( "Power" ).Value ) )
```

---

<sup>\*</sup>1 C#なのに「ワンライナー」ですよ。こんなのC#じゃない！と言いたい。「可読性（笑）」みたいな。でも、あれ、なんだか…気持ちよくなってきたよ…

「var query = doc.Root.Elements().OrderByDescending( plant => int.Parse( plant.Element( "Power" ).Value ) ).Select( plant => new { Name = plant.Element( "Name" ).Value, Power = plant.Element( "Power" ).Value } ).Distinct();」

## LINQ in C# 3.0

クエリ演算子で降順ソートする場合は「`descending`」などのキーワードを付け加えるのではなく、「`OrderByDescending`」メソッドを使います。

### 以上／以下

次に出力が 10 万キロワット（100000）以上の原子力発電所を、強力な順（Power 要素の値で降順）で列挙してみます。

```
1 var query = from plant in doc.Root.Elements()
2     where int.Parse( plant.Element( "Power" ).Value ) >= 100000
3     orderby int.Parse( plant.Element( "Power" ).Value ) ascending
4     select new
5     {
6         Name = plant.Element( "Name" ).Value,
7         Power = plant.Element( "Power" ).Value
8     };
9 all = all.Distinct();
```

### リスト 5

単純に「`where 句`」を書き足しただけです。「`where 句`」はこれまでの 2 回の記事でもよく出てきたので見覚えがあると思います。単純に条件を指定するキーワードです。

結果は、

```
柏崎刈羽原子力(135600)
玄海原子力(118000)
大飯(118000)
大飯(117500)
敦賀(116000)
浜岡原子力(113700)
浜岡原子力(110000)
東海第二(110000)
福島第二原子力(110000)
柏崎刈羽原子力(110000)
福島第一原子力(110000)
```

標準クエリ演算子で書き換えると、

```
1 var query = doc.Root.Elements()
2     .Where( plant => int.Parse( plant.Element( "Power" ).Value ) >= 100000 )
```

```

3     .OrderByDescending( plant => int.Parse( plant.Element( "Power" ).Value ) )
4     .Select( plant =>
5         new{
6             Name = plant.Element( "Name" ).Value,
7             Power = plant.Element( "Power" ).Value
8         }
9     .Distinct();

```

## リスト 6

「Where 句」を使います。

同じように 5 万キロワット (50000) 以下の原子力発電所を小さい順に (Power 要素の値で昇順) 列挙してみます。

クエリ式では。

```

1 var query = from plant in doc.Root.Elements()
2                 where int.Parse( plant.Element( "Power" ).Value ) <= 50000
3                 orderby int.Parse( plant.Element( "Power" ).Value )
4                 select new
5                 {
6                     Name = plant.Element( "Name" ).Value,
7                     Power = plant.Element( "Power" ).Value
8                 };
9 all = all.Distinct();

```

## リスト 7

東海 (16600)
美浜 (34000)
敦賀 (35700)
島根原子力 (46000)
福島第一原子力 (46000)
美浜 (50000)

標準クエリ演算子では。

## LINQ in C# 3.0

```
1 var query = doc.Root.Elements()
2     .Where( plant => int.Parse( plant.Element( "Power" ).Value ) <= 50000 )
3     .OrderBy( plant => int.Parse( plant.Element( "Power" ).Value ) )
4     .Select( plant =>
5         new{
6             Name = plant.Element( "Name" ).Value,
7             Power = plant.Element( "Power" ).Value
8         }
9     .Distinct();
```

リスト 8

### 平均値

発電機出力の平均値を求めてみます。単純に平均値を求める場合なら、

```
1 var query = from plant in doc.Root.Elements()
2     select int.Parse( plant.Element( "Power" ).Value );
```

こうしたり、

```
1 var query = doc.Root.Elements()
2     .Select( plant => int.Parse( plant.Element( "Power" ).Value ) );
```

こうして、Power 要素の値をすべて取得し、

```
1 Console.WriteLine( query.Average() );
```

最後に `IEnumerable` インタフェース<sup>\*</sup> の `Average` メソッドを利用すれば簡単に求めることができます。では電力会社ごとに、各電力会社が所有している原子力発電所の出力平均を降順で出力してみましょう。こういった場合は「Group-By-Into 句」を利用します。

\*1 `IEnumerable` インタフェース。リストなどいくつかの要素を配列のように持つクラスに便利な関数などが定義されている。LINQ でアクセスできるのはこの `IEnumerable` インタフェースと `IQueryable` インタフェースを実装したクラスに限る。

とりあえずコードを見てみましょう。

```
var query = from plant in doc.Root.Elements()
            group int.Parse( plant.Element( "Power" ).Value )
            by plant.Element( "Belong" ).Value
            into power
            orderby power.Average() descending
            select new
            {
                Belong = power.Key,
                Power = power
            };
```

リスト 8

Power 要素の値（発電機ごとの最大出力）を Belong 要素の値（電力会社名）でグループします。つまり、

- 九州電力
  - 川内原子力(1), 89000
  - 川内原子力(2), 89000
  - // 省略
- 中国電力
  - 島根原子力(1), 82000
  - 島根原子力(2), 46000
- // 省略
  - // 省略

といった感じでグルーピングされます。その後変数 power の平均値で降順にソートして、匿名メソッドを用いてデータを取得していきます。最後に foreach 句で出力します。

```
foreach ( var el in query )
{
    Console.WriteLine( "{0}({1}基)", el.Belong.ToString(), el.Power.Count() );
    Console.WriteLine( "{0}KW", el.Power.Average() );
}
```

リスト 9

## LINQ in C# 3.0

結果はこんな感じです。

```
東京電力(17 基)  
101811.764705882KW  
中部電力(4 基)  
90425KW  
関西電力(11 基)  
88800KW  
九州電力(6 基)  
87633.333333333KW  
東北電力(4 基)  
74975KW  
日本原子力発電㈱(4 基)  
69575KW  
四国電力(3 基)  
67400KW  
中国電力(2 基)  
64000KW  
北海道電力(2 基)  
57900KW  
北陸電力(1 基)  
54000KW
```

さすが東京電力、強いですね<sup>\*</sup>。これをクエリ演算子で書き換えてみます。

```
1 var query = doc.Root.Elements()  
2     .GroupBy(  
3             plant => plant.Element( "Belong" ).Value,  
4             plant => int.Parse( plant.Element( "Power" ).Value )  
5         )  
6     .OrderByDescending( group => group.Average() )  
7     .Select( group => new {  
8             Belong = group.Key,  
9             Power = group  
10        } )
```

---

\*1 しかし「東京電力 柏崎刈羽原子力発電所」は新潟中越沖地震の影響から現在点検・復旧作業中の為運用していません。

```

11
12 );

```

リスト 10

クエリ演算子では「into 句」は使いません。「GroupBy」メソッドの第一引数にグループに用いる値、第二引数にグループされる値を指定します。クエリ式の時と逆になっています。ややこしいですね。その後はクエリ式と同じように平均値の降順に匿名型のオブジェクトインスタンスに代入していきます。

最終的な出力はクエリ式と同じようにします。

### XML ファイルへの出力

最後にこれらの結果を新しい XML ファイルに保存してみます。これも LINQ の構文を使って記述することができます。最後の「電力会社別平均出力」を XML にしてみましょう。

ここまでくると「カオス」としか言いようがない感じですが、階層構造は

```

- Plants
  - Plant
    - Belong - PlantCounts (発電機数)
    - Power
  - Plant
    - // 省略

```

といった XML を出力します。

```

1 var outputs = new XElement( "Plants",
2   from plant in doc.Root.Elements()
3   group int.Parse( plant.Element( "Power" ).Value )
4   by plant.Element( "Belong" ).Value
5     into power
6     orderby power.Average() descending
7     select new XElement( "Plant",
8       new XElement( "Belong", power.Key,
9         new XAttribute( "PlantCounts", power.Count().ToString()
) ),
10        new XElement( "Power", power.Average().ToString() ) )
11      );
12 outputs.Save( @"outputs.xml" );

```

リスト 11

## LINQ in C# 3.0

XElement クラスには Save メソッドがあり、これを呼ぶことで任意のファイルに出力することができます。

標準クエリ演算子で書いた場合はこのようになります。

```
1 var output = new XElement( "Plants",
2     doc.Root.Elements()
3     .GroupBy(
4         plant => plant.Element( "Belong" ).Value,
5         plant => int.Parse( plant.Element( "Power" ).Value )
6     )
7     .OrderByDescending( group => group.Average() )
8     .Select( group => new XElement( "Plant",
9         new XElement( "Belong", group.Key,
10            new XAttribute( "PlantCounts", group.Count().ToString()
11        ),
12            new XElement( "Power", group.Average().ToString() )
13        )
14    );
15 output.Save( @"output.xml" );
```

リスト 12

最終的な XML は以下のようになります。

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <Plants>
3   <Plant>
4     <Belong PlantCounts="17">東京電力</Belong>
5     <Power>101811.764705882</Power>
6   </Plant>
7   <Plant>
8     <Belong PlantCounts="4">中部電力</Belong>
9     <Power>90425</Power>
10  </Plant>
11  <!-- 省略 -->
```

XElement の Save メソッドは呼ぶだけで「<?xml version="1.0" encoding="utf-8"?>」といった XML 文書の宣言を書いてくれます。もちろん Save メソッドの引数に SaveOption などを指定すること

もできます。

### まとめ

今回はかなり盛りだくさんの内容になってしまいました。どんどん「これまでの C#っぽくないコード」が出てきましたが、皆さんも「クエリ式」形式と「標準クエリ演算子」形式のどちらか書きやすい方法で LINQ to XML を試してみてください。なかなか気持ちよくほしいデータを引っ張ってくることができます。そして加工や保存も慣れれば簡単です。

## 嗚呼、オーディオ

### 嗚呼、オーディオ

文責 かづきお

## 私は自作スピーカーを愛している

私の嫁はゆいこさんである。スペックは D-118 型バックロードホーン改<sup>\*1</sup>FE88ES-R 仕様。その性格は上品で艶めかしく、たまにエロいお姉さんだ。私はゆいこさんを愛している。その歌声を聞いてから寝るのが私の日課だった。

しかし、ある日私は浮気をした。これはその記録である。



私の嫁と愛人 外側がゆいこさん、内側がことりちゃん

## ことりちゃんといっしょ

### ・出会い

私が D-83 フラミンゴ<sup>\*2</sup>を作ろうと決めたのは 2008 年の夏休みが始まる直前のことだった。き

\*1 バックロードホーン：スピーカーの種類の一。スピーカーユニット（実際音が出て振動する部分）の後ろにホーンが付いているタイプ。

\*2 D-83 フラミンゴ：故長岡鉄男氏が設計したバックロードホーンタイプのスピーカーの一。8 センチユニットを用いた小型のバックロードホーン。小型といっても 8 センチユニットを用いる割には大きい。

っかけは、師匠宅で聴いた D-88 スーパーフラミンゴ<sup>\*3</sup> のことが忘れられなかつたからだつたと記憶している。可愛い頭、すらつとしたボディ、その容姿からは考えられないような表現力。私は3年間つきあつてきゆいこさんに対し浮気をしたのである。

はじめ私はスーパーフラミンゴを作るつもりでいた。しかし FE88ES というユニットは限定品で、入手方法はオークションしかない。2008年の頭頃にはかなりオークションに出回っていたが、夏頃にはほとんどなくなつていた。

しかし、FE83 の後継機である FE83E<sup>\*4</sup> は量産品のため簡単に入手することができる。さらに、同じサイズで FF85K<sup>\*5</sup> というバイオセルロースコーンを用いた量産ユニットがあり、同じサイズ故に簡単に付け替えることができる。

フラミンゴを含むスワン族<sup>\*6</sup> と呼ばれるバックロードホーンは、複雑な組み方をするため板をカットするときの精度が閑静に大きく影響する。そこで私は専用の業者にカットを頼むことに決めた。

板は MDF ボードという板材を用いた。これはメーカー製スピーカーでも採用されており、板全体が一定の密度で均一であることがメリットだった。

また、くみ上げるに当たつて輸入品のタイトボンドという接着剤を用いた。これは通常の木工用ボンドと違い、乾燥するとヤスリで削れるほど堅くなるためスピーカー工作にもってこいだというアドバイスをいただいていたからである。

そしていよいよ組み立てに入ったわけだが、制作は難航を極めた。フラミンゴを含むスワン族と呼ばれるバックロードホーンは、複雑な立体音道を持っている。その音道を構成する複数の板を正確に接着するのはとても神経を使う作業だった。

---

\*3 D-88 スーパーフラミンゴ：故長岡鉄男氏が設計したバックロードホーンの一。D-83 フラミンゴを、6N-FE88ES という限定ユニットが使えるように最適化したバージョン。

元々 D-83 フラミンゴは、発売されないであろうといわれていた 8 センチバックロード専用ユニットを諦め、当時発売されていた FE83 というユニットを用いた苦肉の策である。

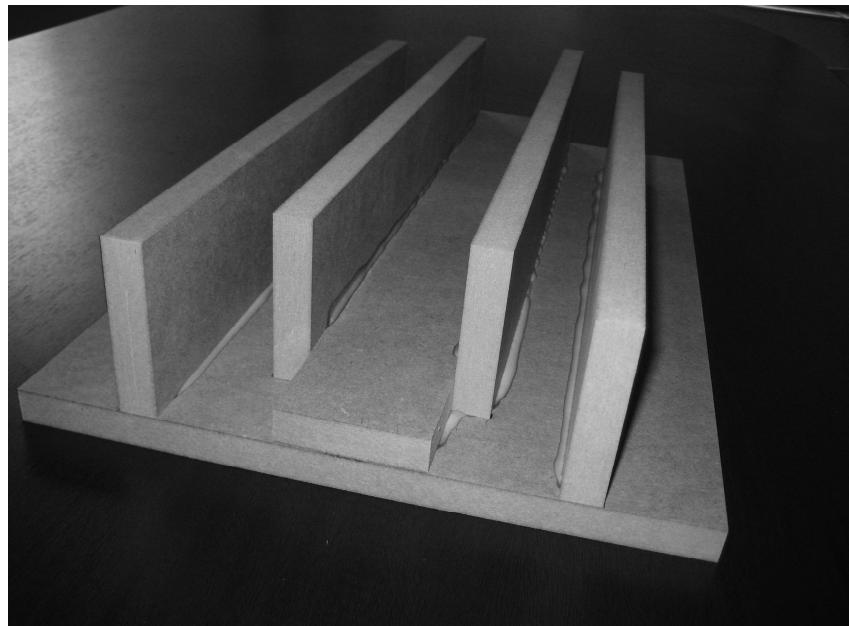
その後 6N-FE88ES という限定ではあるがバックロード専用ユニットが発売されたためそれに合わせて再設計されたのがスーパーフラミンゴである。

\*4 FE83E : Fostex が発売する 8 センチタイプのスピーカーユニット。Q0 値（最低共振周波数での共振の度合いを示す値）が大きいのであまりバックロード向けとはいえないが、FE シリーズに共通する透明感と独特の優しさをもつ良いユニットである。

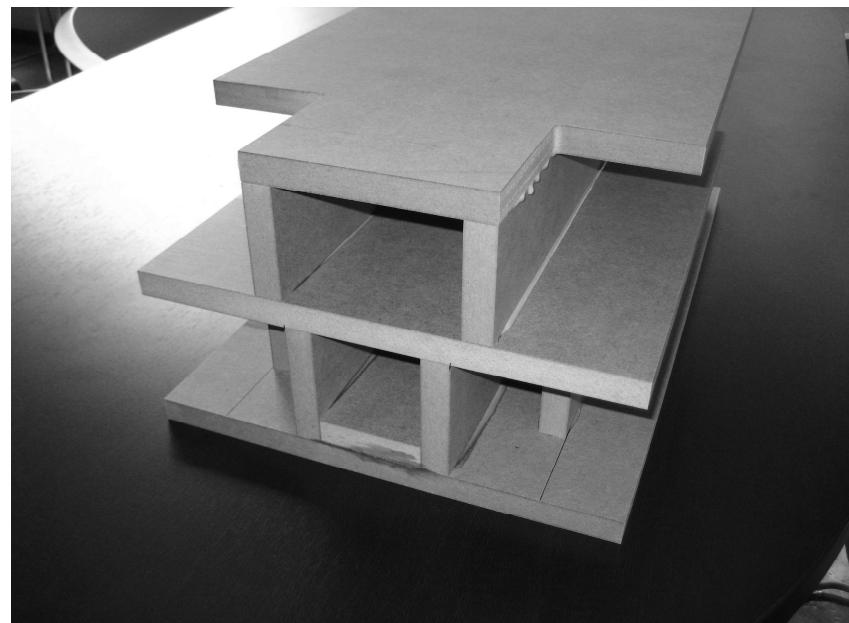
\*5 FF85K : Fostex が発売する 8 センチタイプのスピーカーユニット。Q0 値が小さいのでこちらはバックロードホーン向けのユニットである。バイオセルロースコーンを用いた、高音域に華やかさのある良いユニットだ。

\*6 スワン族：故長岡鉄男氏が設計した理想的な点音元・3 次元立体音道（後ろのホーンの部分のこと）をもつスピーカー群の総称。ペットネームは使用するユニットによって名付けられており、8 センチタイプのフラミンゴ、10 センチタイプのスワン、16 センチタイプのレア、20 センチタイプのモアがある。それぞれには限定ユニットに最適化されたスーパーが存在する。

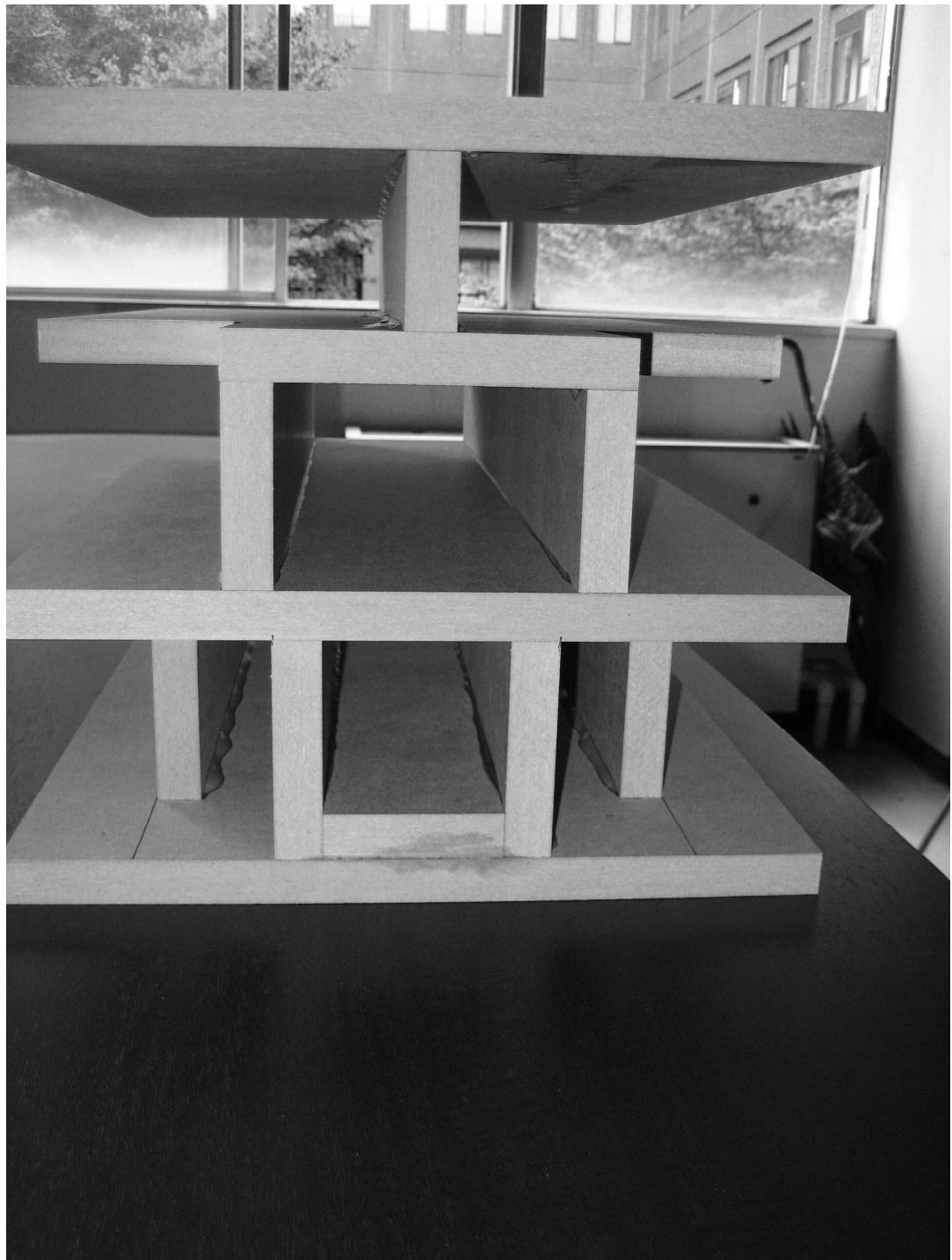
## 嗚呼、オーディオ



胴体の部分に収まっている音道部を作り始めたところ

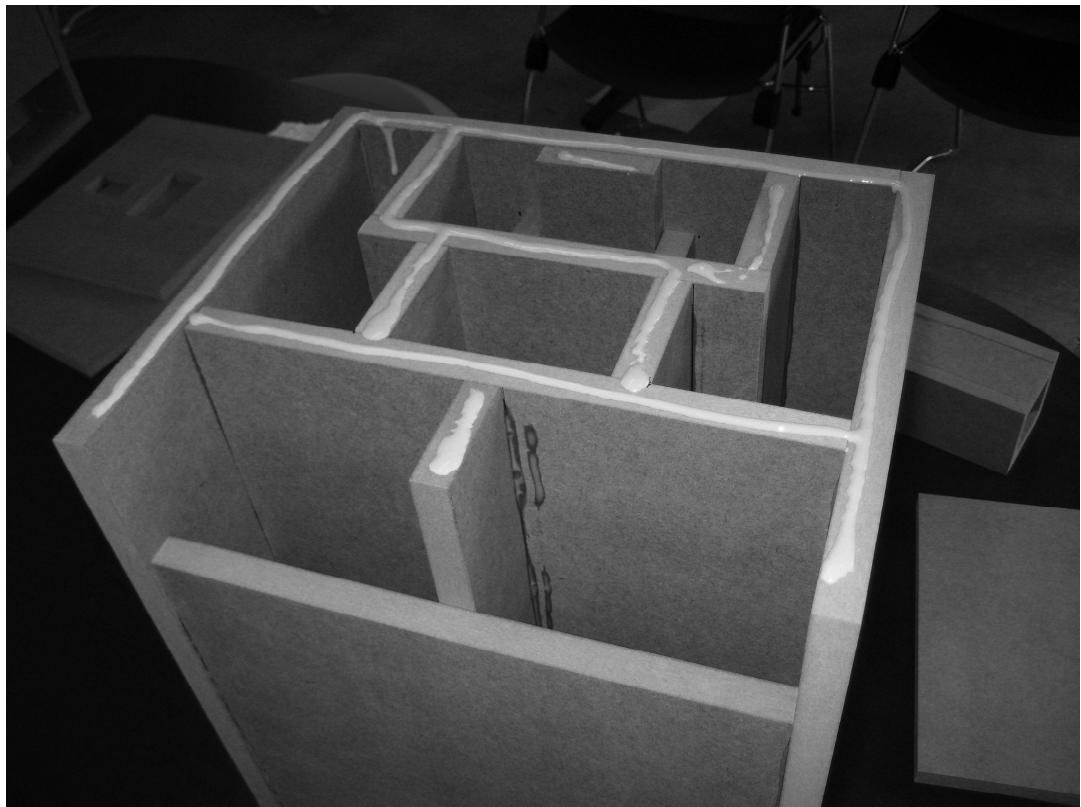


ここで横向きの3枚の板の位置取りにミスると大変なことになる



複雑な音道部。下から上へ徐々に断面積が大きくなっていくのが分かる

嗚呼、オーディオ



複雑な立体音道。ここが一番の肝である。

そして、いよいよ組み上がったのである。



組み上がったことりちゃんこと D-83 フラミンゴ。美しいプロポーションだ。

肝心のユニットは、FF85K を用いた。このユニットはバックロードであり、かなり良い感じに鳴るよという某大学教授のアドバイスがあったためである。

そういう経緯で D-83 フラミンゴ、ことりちゃんが私のものになったのである。

## 嗚呼、オーディオ

### ・幼児期

はじめ音が出たときは少し期待しすぎたかなと思ったものだ。作りたてのスピーカーがあまりうまく鳴らないのは当然のことだが、少し寂しすぎた。

全然出てこない低音、紙っぽい中音、金切り声のような高音。これでは鳴くではなく泣くではないか。

とはいえた塩にかけて育てたことりちゃんはかわいいものである。人間でも幼児のうちは手間が掛かるもの。自分好みに調教するのも一興である。

ことりちゃんはしばらく鳴らし続けているとだんだんとその才能の片鱗を見せ始めてきた。

最初にそのことに気づいたときのことは今でも覚えている。

当時買ったばかりの「スカイクロラ オリジナルサウンドトラック」を聴いていたときのことである。最初はBGM的にかけていたのだが、たまにはしっかり聴くかと思いリスニングポジションに座りボリュームをいつもより多めに上げてみた。

最初の音が出た瞬間ハッとした。スピーカーの中心から同心円状に広がっていくような音の空間。まだまだ幼稚だが私が欲していたものの片鱗がそこにはあった。

私は、こいつは将来きっと美人になると確信した。

とはいえた、まだまだやんちゃなスピーカーであった。

ガチャガチャとやかましいだけのやんちゃなスピーカーであることに変わりはなかった。

### ・反抗期

ことりちゃんは生後2ヶ月にもなる頃には大分まともに歌うようになった。低音もかなり出るようになり、高音のやかましさが消え見通しの良さが目立つようになった。

しかし、低音の増大に伴い新たな問題が発現した。

縮まりがないのである。ただ低音が出ているだけで、表現の欠片もなかった。

全くだらしがない。こんな音の垂れ流しを聴くためにここまで育ててきたわけではない。

そこで私はいよいよ道具を用いて調教を開始した。まずはユニットの裏と背中の開口部に吸音材を敷いた。吸音材に用いたのは、ニードルフェルトである。

効果は抜群だった。

ポンつき（特定の音が共鳴してぼやけること）がなくなり、幾分聞きやすくなった。

しかし、逆に低音の量自体も減少してしまったように感じられた。低音の量が減少すると全体的に迫力がなくなり、弱々しくなってしまった。

そこで、胴体の上に錘をおき、全体の重量を増すことにした。錘による低音の増大はゆいこさんで確認済みである。

利用したのはダンベル用の錘である。たまたま手元にあったからだが、なかなかどうして効果を發揮してくれた。

ポンつきは起こさない今まで低音の増大に成功したのである。

しかし、これでもまだ私の望んだ音ではない。そのためには適切な靴を履かせてやる必要がある。

今まで 10 円玉インシュレーター<sup>7</sup>（ただの十円玉ともいう）を履かせていたのだが、やはり専用のものを履かせてやりたい。

まずオーディオテクニカのハネナイトゴム製インシュレーター AT6091 を履かせてみた。  
しかし、私の好みには合わなかった。全体的に音に厚みが出たが、解像度がいまいちで、音が重なって聞こえる。やはり金属系のインシュレーターが良いのか。

そこで、ゆいこさんが履いている金属スパイクインシュレーター、TITE-25PIN を履かせてみた。  
思わずニヤリとした。おお、自分好みの音になってきたぞ……。  
引き締まった低音と、解像度の高い高音。欲を言うならもう少し音に艶っぽさが欲しいかな。  
そういえば師匠宅のスーパーフラミンゴは真鍮ベースのインシュレーターだった。  
手持ちが他にないので、またどこかでスパイクでない金属系インシュレーターを見繕ってくるかなと思いながら、フラミンゴの履き物は TAOC の TITE-25PIN になったのである。

しかし、ここで問題が発生した。ゆいこさんが拗ねてしまったのである。もともと TITE-25PIN はゆいこさんが履いていたもの。とりあえず AT6091 を履かせてみたものの、締まりのないおとで、やる気がなくなってしまったようだ。

しかし、TITE-25PIN はもうすでに生産中止になってしまっている。  
ゆいこさんにはもっと高級な履き物をプレゼントして機嫌を取ることにしよう。

#### ・そして今

ことりちゃんとの同棲生活が始まって半年が過ぎようとしていた。  
ことりちゃんはすっかり子供っぽさが抜け、時折見せる表情にドキッさせられることが多くなってきた。

正月になり、ばたばたしてなかなかゆっくり聴く時間がとれずにいた。  
そして先日、久しぶりに大きなボリュームで聴く時間がとれた。

もうこれはやばい。これだ。この音を私は求めていたのだ。何という表現力。何という解像度。

**ことりちゃんは立派な成鳥となって羽ばたいているのだ！**

特筆すべきはやはり 8 センチのユニットからは信じられないほどの低音が出ていることだろう。  
しかも、ごりごりとした押しの強い低音だ。  
ゆいこさんは、良くも悪くも上品なのだ。ボーカルを歌わせたらかなうものがないと思っていたが、ことりちゃんはそれとは全く正反対で、場合によっては下品にもなりかねないそれすれのレベルで力強い低音を発している。

中音域は、ゆいこさんが艶っぽく艶めかしく歌うのに対しことりちゃんははきはきとした張り

---

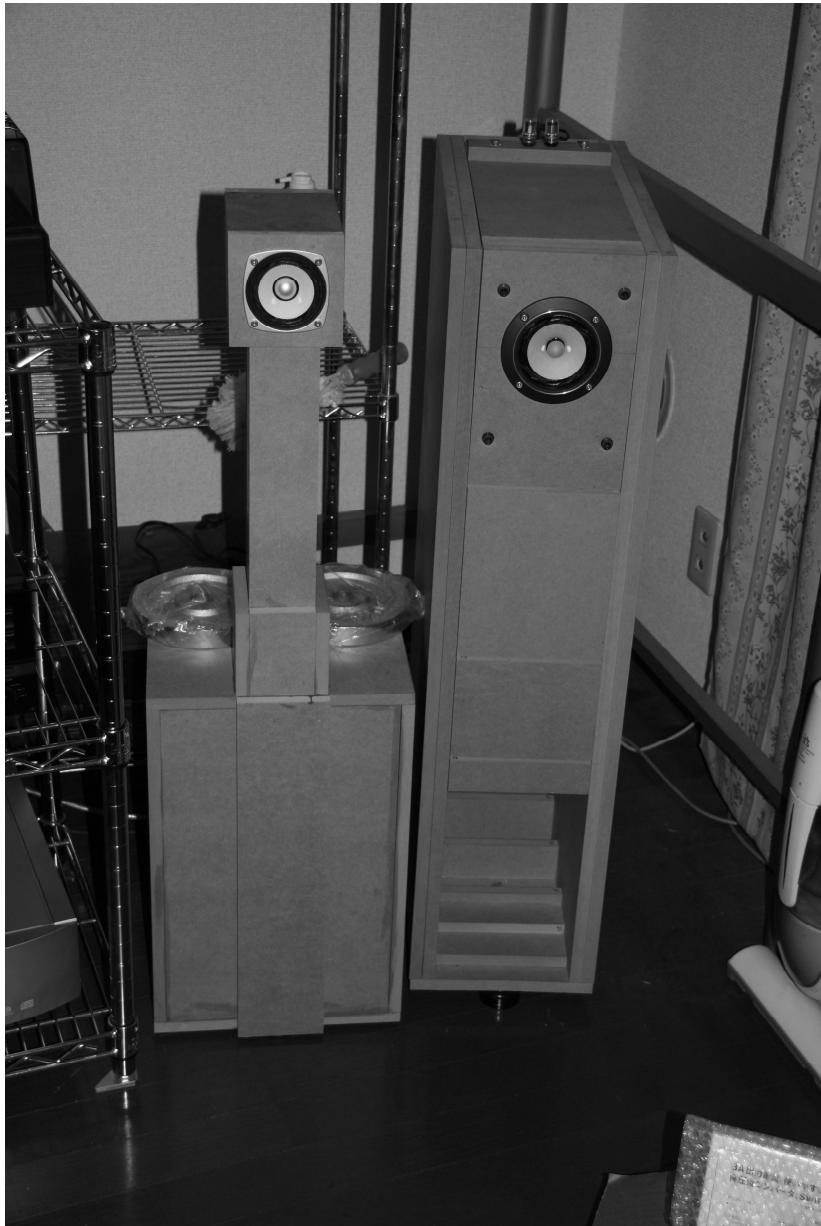
\*7 インシュレーター：スピーカーやアンプ、プレーヤーの下に敷いて床、あるいはラックなどの振動と本体の振動を遮断するためのパーツ。材質は木材やゴム、金属など様々な物が発売されている。

## 嗚呼、オーディオ

のある歌を聴かせてくれる。

高音域の透明感はゆいこさんの方が有るが、ことりちゃんには鋭く切れ込むような強さがある。

ゆいこさんのほうも、新しいブーツをプレゼントしたらとたんに機嫌を戻してくれた。そして今ではことりちゃんとコラボレーションでいつもと違った歌を聴かせてくれることもある。まったく私は果報者である。



さあ、今夜はどんな声で鳴いてくれるのかな。

# 第1回 XNAで遊ぼう

文 編集部 少佐

## XNAって？

XNA とは Microsoft が提供している新しいゲーム開発環境です。「C#」という「C 言語」よりも初心者にとって手軽な、プログラミング言語で使用することができ、容易にゲームを開発することができます。ちなみに XNA は「XNA's Not Acronymed」(XNA は頭字語でない) の再帰的頭字語です。

XNA の代表的な利点としては以下の通りです。

### **わかりやすい。**

DirectX や OpenGL のよくわからない初期化から解放されます。各種関数も遙かに他の環境よりも整っています。また、言語が C#なので、ポインタという初心者にとって煩わしいものから解放されます。

### **サンプルが豊富。**

DirectX や OpenGL のサンプルと違って実際にゲームのリソース(画像など)とコードが付属しています。初心者でゲーム開発がよくわからない人にもお勧めできます。

### **Windows、Xbox360、Zune などのマルチプラットフォーム。**

XNA で開発したゲームは、Windows でも Xbox360 でも動きます。公式でゲーム機の開発をサポートしたのは全世界でこれが初めてではないでしょうか？

### **無償。**

Visual C# 、XNA は Microsoft から無料でダウンロードできます。ただし、Xbox360 での開発には年に約 1 万円ほどかかります。それでも、商業の開発環境を整えるにはかなりの出費が必要ということを考えれば安いものでしょう。

### **全世界に販売可能。**

なんと XNA で自作したゲームは XboxLIVE を使用して全世界に販売することが出来ます。ゲームを 200 ~ 800 マイクロソフトポイント(約 10 ドル)で販売することができ、その売り上げから 30% の手数料を引かれた 70% が製作者に支払われます。この機能は今年より随時、稼動開始だそうです。日本は Xbox360 ユーザーがとても少ないので、かなり後回しになるでしょうが、今後個人でゲーム開発する方はこれを視野にいれるのもいいのでは無いでしょうか？

若干 DirectX や OpenGL に対して細かいところを弄ることが難しいという欠点はありますが、なんだかんだでその分、手軽でやさしいので初心者にとてもお勧めです。

では、次のページから、軽い使用方法とサンプルについて説明します。  
なお、「C#」に触った経験がある方を対象に解説してあります。

## 第1回 XNAで遊ぼう

### インストール

それぞれ以下のアドレスより、インストーラをダウンロードしてインストールしてください。

- Visual C# 2008 Express Edition

<http://www.microsoft.com/japan/msdn/vstudio/Express/>

- Microsoft XNA Game Studio 3.0

<http://www.microsoft.com/downloads/details.aspx?FamilyId=7D70D6ED-1EDD-4852-9883-9A33C0AD8FEE&displaylang=en>

正直、これを打ち込むよりも Microsoft XNA Game Studio 3.0 で検索したほうが速いでしょう。

これらのインストールに特に難しい点はありません。もし XNA の前のバージョンがインストールされている場合は、それをアンインストールしてからでないとインストールできません。ご注意ください。

### 新しいプロジェクトの作成

Visual C# 2008 を起動し、上部のメニューから「ファイル」>「新しいプロジェクト」を選択します。もし、インストールが成功していれば、「Microsoft XNA Game Studio 3.0」の項目が増えます。そこから、「Windows Game (3.0)」を選択します。(図 1)

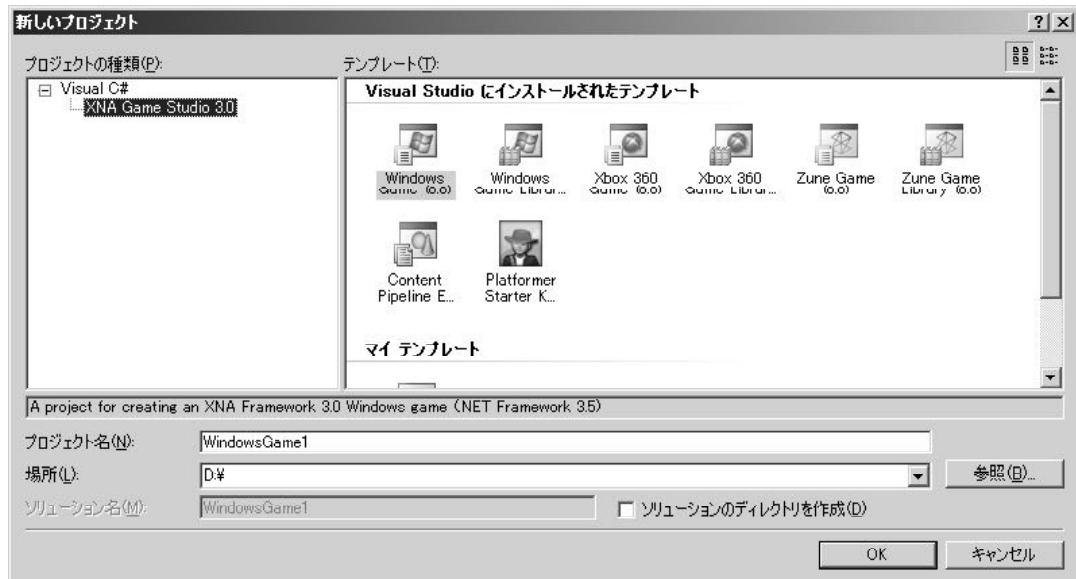


図 1 「新しいプロジェクト」

コンパイル、そして実行

何も変更を加えずに、コンパイルすると青一色のウインドウが表示されます。（図2）いくら待っても何も起きません。ですが、この時点では60FPS維持、データの更新と描画の分離等、ゲーム開発で必ず実装される機能は標準で実装されています。



図2「ウインドウ」

とりあえず、画像表示

「Pic.png」という画像を画面に描画してみます。（図4）

## ・プロジェクトに対する追加

プロジェクトに「Content」というフォルダ（図3）が生成されているので、そのフォルダの中に「Pic.png」を貼り付けます。そして、ソリューションにファイルを登録します。



図4「Pic.png」



図3「Content」

## ・ソースコードに対する追加

標準で生成されるクラス、「Game1」のメンバ変数に「Texture2D」クラスを追加します。

```
public class Game1 : Microsoft.Xna.Framework.Game {
    GraphicsDeviceManager graphics;
    SpriteBatch spriteBatch;
    Texture2D texture; //追加
```

「Texture2D」は名前のとおり、画像を格納するクラスです。

## 第1回 XNAで遊ぼう

ファイルを読み込むために、メソッド「LoadContent」に以下のコードを追加します。

```
protected override void LoadContent() {  
    spriteBatch = new SpriteBatch(GraphicsDevice);  
    texture = Content.Load<Texture2D>("Pic"); //追加  
}
```

「Content」は「Content」フォルダに追加したリソースを管理するクラスです。そのクラスに「Pic.png」を読み込むよう関数を実行します。この際、拡張子は記入する必要はありません。

メソッド「LoadContent」は、ゲーム開始時に実行されるメソッドです。

描画するために、メソッド「Draw」に以下のコードを追加します。

```
protected override void Draw(GameTime gameTime) {  
    GraphicsDevice.Clear(Color.CornflowerBlue);  
    this.spriteBatch.Begin(); //追加  
    this.spriteBatch.Draw(this.texture, new Vector2(127,127), Color.White); //追加  
    this.spriteBatch.End(); //追加  
    base.Draw(gameTime);  
}
```

メソッド「Draw」は、画面への描画時に実行されるメソッドです。

this.spriteBatch.Begin();

2Dの描画の開始を指示するためのメソッドです。

this.spriteBatch.End();

2Dの描画の終了を指示するためのメソッドです。

this.spriteBatch.Draw(this.texture, new Vector2(127,127), Color.White);

2Dの描画を行うメソッドです。

texture を(127,127)の位置に画像そのままの色で描画します。

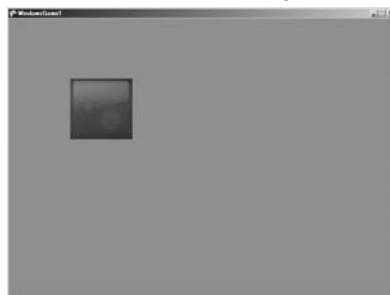


図 5 「結果」

結果はこのようになります。(図 5)

わずか 5 行の追加で画像の描画をすることができました。これを DirectX でやるとしたら、何百行書く必要があるでしょうか？

今回は簡易な解説だけでしたが、XNA の手軽さはよくわかっていただけだと思います。これからプログラムを始めてゲームを作りたい、という方は是非使ってみてください。

## マルチキャスト入門 第一回

### はじめに

さて、みなさん。突然ですがマルチキャストをご存知でしょうか？

単語を耳にしたことがある方や、実際運用している方など、様々な読者の方が居られると思います。マルチキャスト入門では、そんなマルチキャストに関する基礎から運用までを3～4回程度に分けて解説していきます。

「なんでマルチキャストなんだ？<sup>\*1</sup>」というと、マルチキャストに関する書籍があまり充実していないという現状があった<sup>\*2</sup>のと、マルチキャストをあまり知らない人が多いためです。まあ、**單にやりたかっただけ**なんんですけどね（笑）

これを通じてマルチキャストに少しでも興味を持つていただければ幸いです。

### ストリーミング配信を例に

では早速、ストリーミング配信を例にいろいろと説明していきたいと思います。なぜストリーミング配信を例としているのかというと、マルチキャストという通信モデルがTVのように、映像や音声をリアルタイムに複数の視聴者へ配信するのに適しているからです<sup>\*3</sup>。

さて、ここでは20Mbpsの映像を複数のクライアントに対して配信しているとして、以下の図.1のように映像を配信をするサーバとそれを受信して視聴するクライアントがいたとします。

---

\*1 決して建設業者が関係しているわけではない。

\*2 和書でここ3年以内に出た本は殆どありませんし、新しい技術に関する解説本なども需要の関係から出版されていないようです。Amazonで洋書も含めて検索しても古い本ばかりです。

今回参考書籍として購入した「マスタリング TCP/IP IPマルチキャスト編」ですら既に9年前の書籍です。IPマルチキャストはこの9年間の間に大きく変化を遂げました。この手の本が、もうちょっと充実してくれるといろいろ助かるんだけどなあ。

\*3OCNひかりTVのように、Bフレッツ網のような閉域ネットワークにおいてマルチキャストを用いたマルチメディアコンテンツ配信サービスがスタートしました。

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

## multicast

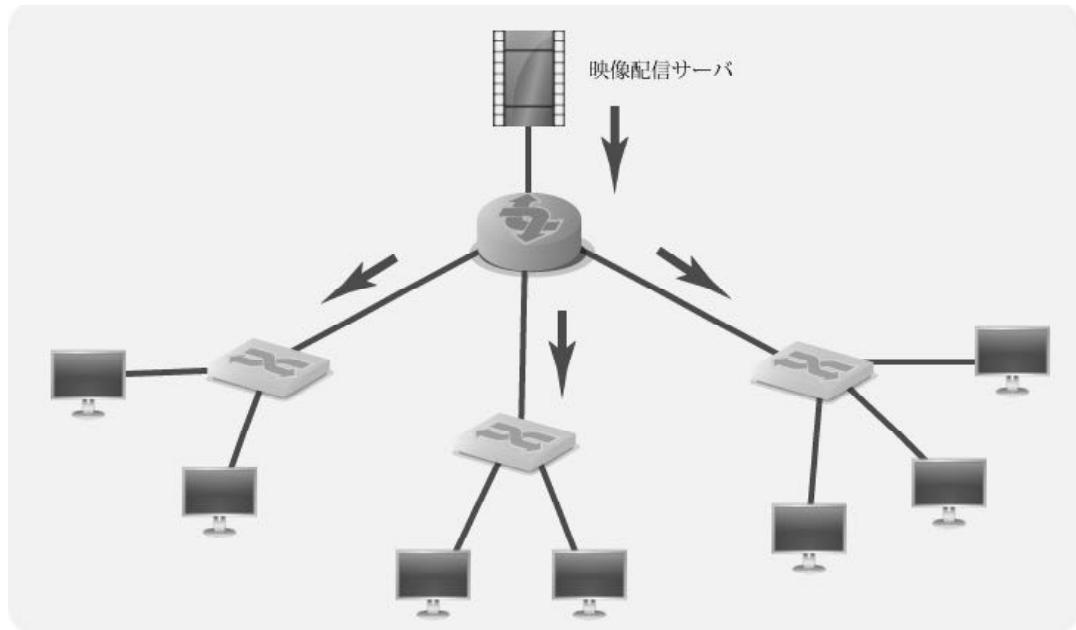


図.1 映像配信用ネットワークの例

このとき、サーバはどのような方法でクライアントにデータを送信するのでしょうか。ネットワークにおけるデータ送信の基本は、**一人に対してデータを送信するユニキャスト**と、**全員に対してデータを送信するブロードキャスト**です。というわけで早速ユニキャストでデータを送信してみましょう（図.2）。

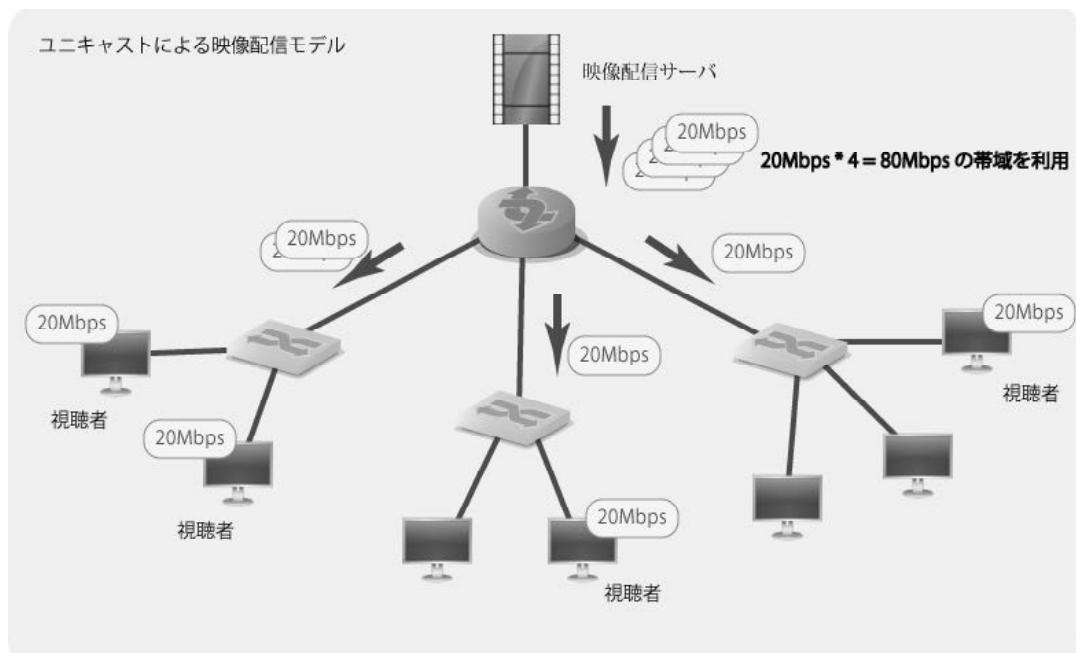


図.2 ユニキャストでの配信例

このとき問題となるのはサーバがデータを送信するときに使用するネットワーク帯域です。サーバはクライアントと、一対一の通信を行おうとします。すると、サーバはクライアントの数だけ同じデータを送信しなければなりません。たとえば **20Mbps** の映像を 100 台のクライアントに配信していたとすると、サーバが使用するネットワーク帯域は、

$$20\text{Mbps} * 100 = 2\text{Gbps}$$

となってしまうので、現実的であるとは言いがたいですね。このようにユニキャストでは、同一のデータを何重にも送信するため、ネットワークに非常に高い負荷をかけてしまいがちです。ですが、インターネットを介してリアルタイム映像配信を行うサービスの大半が、この例と同じようにユニキャストでデータを配信しています<sup>1</sup>。

というわけで次に、ブロードキャストでデータを送信してみましょう（図3）。

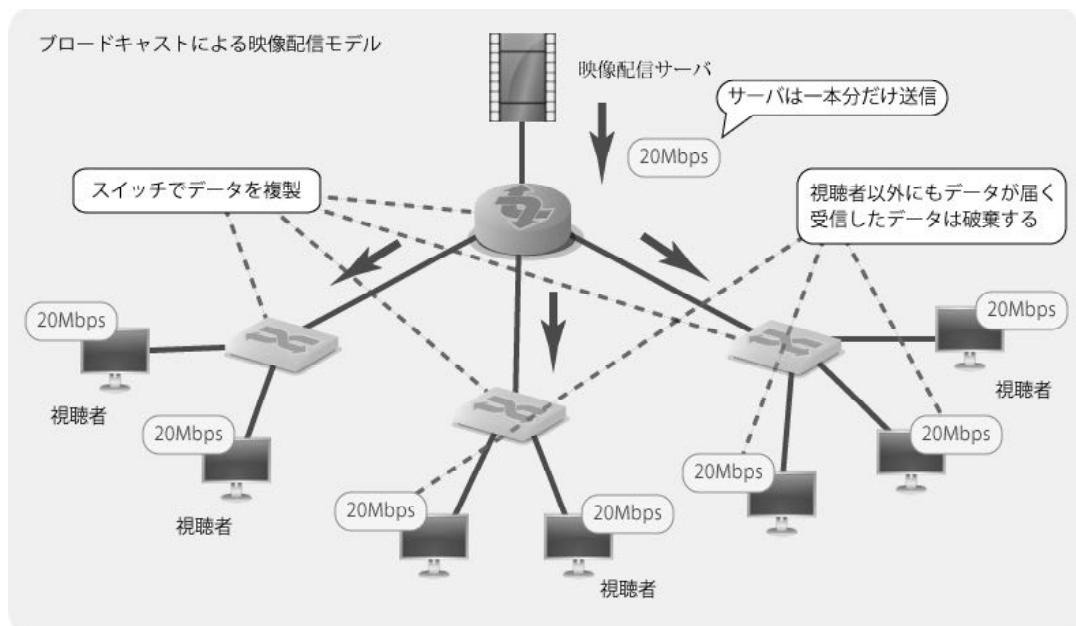


図3 ブロードキャストでの配信例

ブロードキャストの場合、データの複製はネットワークスイッチ<sup>2</sup>がやってくれるので、サーバが使用するネットワーク帯域は 20Mbps で済みます。しかし、送信されたデータは**無条件でネットワーク上の全てのホストに対して送信される**ため、受信したくないホストまでもがデータを受信し処理をしなければなりません。無理やり送られてきたにもかかわらず処理をしなければい

\*1 ニコニコ生放送 (<http://live.nicovideo.jp/>) もユニキャスト通信で配信を行っています。早くマルチキャスト配信になればいいのに。ニコ6まだー?

\*2 スイッチング HUB やルータなど。

## multicast

けないので、受信したくないホストにとってはたまたもんじやありません<sup>\*</sup>。

また、ブロードキャストはルータを超えないため、大規模なネットワーク上で運用することができません。あくまでサブネット内部の話になってしまいます。

### マルチキャスト

そこで、受信したいホストだけにデータを送信することができれば、サーバもクライアントもハッピーになれるはずです。またネットワークにもやさしいので一石二鳥です。そんなことが可能なのがマルチキャストです。

マルチキャストでデータを送信すると以下のようになります（図4）。

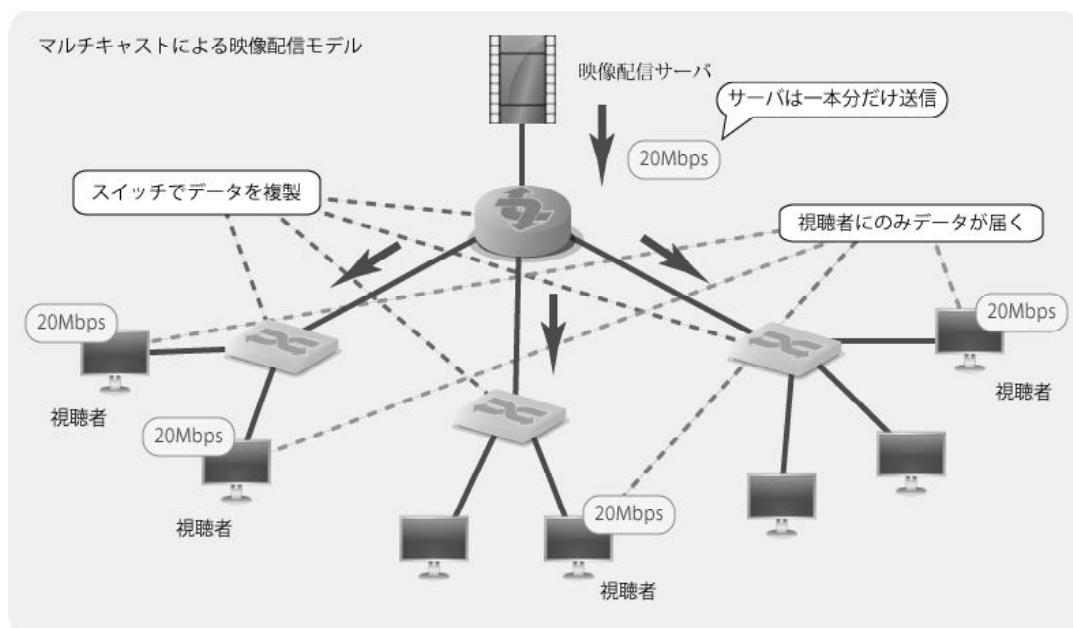


図4 マルチキャストでの配信例

このとき、サーバが使用するネットワーク帯域は 20Mbps で済み、また受信したいホストにだけデータが送られます。また、視聴者以外にはデータが送信されないため、ネットワークを**効率的に利用**しているということもできます。

また、受信したいホストをグループとして捉えると、マルチキャストはグループに対して通信をしていると見なすことができます（図5）。この考え方とは、ユニキャストやブロードキャストとまったく異なった考え方で、送信元は、受信したいホストの数などを知る必要もなく、グループに対してデータを送信するだけで済みます。これはつまり、新規に受信したいホストはグループに参加するだけで、データを受信できるようになります。

\*1 見たくもない番組を無理やり見せられる、あのやるせなさと言ったらもう。

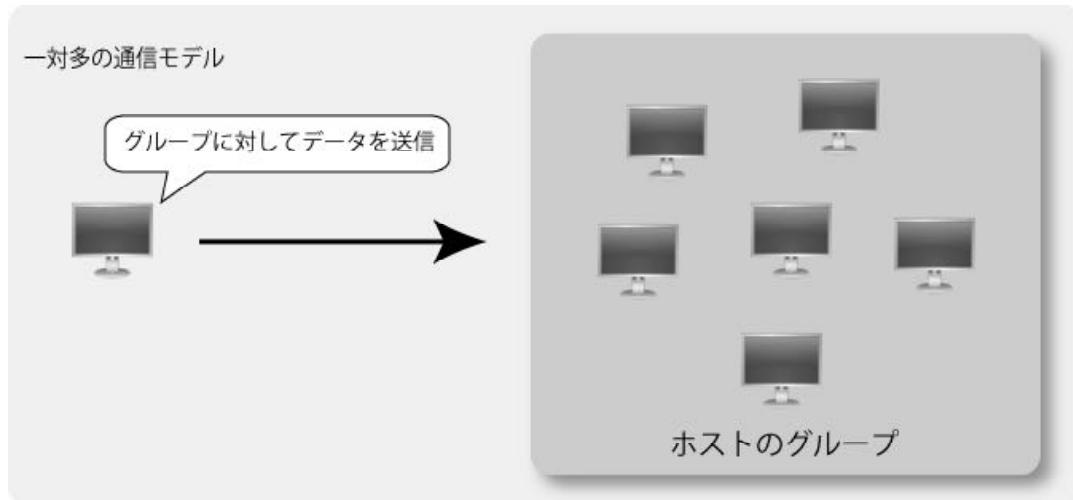


図.5 グループへ送信

グループの識別 (IP)

マルチキャストでは、送信元はグループに対して通信を行います。では、マルチキャストにおけるグループは、どのようにネットワーク上で識別されるのでしょうか？

ユニキャストとブロードキャストでは、送信先をそれぞれ IP アドレスで識別していました。マルチキャストも同様に、グループの識別を IP アドレスを用いて行います。このとき利用される IP アドレスを特に **IP マルチキャストアドレス** と言い、ユニキャストとブロードキャストで利用される IP アドレスとはまったく異なった方法で管理されています。

IP マルチキャストアドレスには、IPv4 ではクラス D (224.0.0.0 ~ 239.255.255.255) のアドレスが利用されます<sup>\*1</sup>。

全マルチキャスト対応ホスト	224.0.0.1
全マルチキャスト対応ルータ	224.0.0.2
全 OSPF <sup>*2</sup> ルータ	224.0.0.5

**IP マルチキャストアドレスの例**

これによって、グループに対してデータを送信するときも、送信側は 1 つの IP アドレス (IP マルチキャストアドレス) を宛先とするだけでマルチキャストによる通信することができます。

グループの識別 (MAC アドレス)

ところで、ある IP アドレスを持つホストにデータを送信するとき、送信元ホストは IP アドレ

\*1 詳しくは RFC2365 を参照のこと。

\*2 Open Shortest Path First の略。OSPF はルーティングプロトコルの一種で、ルータ同士が経路を交換する際に用います。OSPF では、他のルータに経路を通知する際に、IP マルチキャストアドレス宛にデータを送信します。

## multicast

スから ARP と呼ばれる仕組みを用いて MAC アドレスを解決した上で、その MAC アドレス宛にデータを送信しています。

もちろん IP マルチキャストアドレスにも、それに対応する MAC アドレスがあります。しかしこれらは ARP によって解決することができません。なぜなら、**送信先ホストが一意に定まらない**ためです。先述のとおり、マルチキャストの基本は**一対多の通信**で、送信されたデータを受信するホストは複数存在します。ですので、ARP のように送信先ホストを一意に定めようとするアーキテクチャは適さないのであります。

そこでマルチキャストには、送信先 IP マルチキャストアドレスに対応する MAC アドレスをネットワークに問い合わせることなく解決することができる仕組みが採用されています。ではどのようにして求めることができるのでしょうか。以下のような IPv4 の IP マルチキャストアドレスがあるとします。

IP マルチキャストアドレス

239.0.0.1

2 進数表記

11101111 00000000 00000000 00000001

ここで IPv4 の IP マルチキャストアドレスを対応する MAC アドレスにマッピングします。このとき、MAC アドレスの先頭 25bit を必ず以下のようにしなければならないと IEEE で定められています。

00000001 00000000 10011110 0

MAC アドレスは 48bit ですので、残りの 23bit はどのように決めるのでしょうか。そこで登場するのが送信先 IP マルチキャストアドレスです。この残り 23bit に、送信先 IP マルチキャストアドレスの下位 23bit をそのままマッピングします（図.6）。

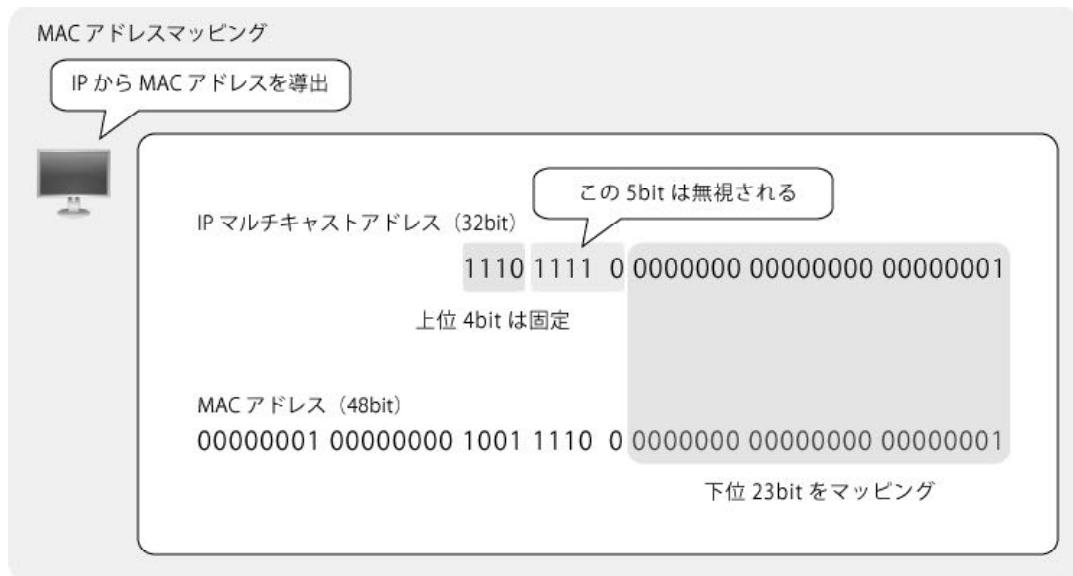


図.6 IP マルチキャストアドレスの MAC アドレスへのマッピング

これによって、先ほどの例では、IP マルチキャストアドレス「239.0.0.1」の送信先 MAC アドレスとして、「01:00:5E:00:00:01」を導出することができました。

しかし、このマッピングにおいて注意しなくてはならないことがあります。それは、MAC アドレスと IP マルチキャストアドレスが**必ずしも一対一の対応付けにはならない**ということです。先ほどのマッピングを良く思い出してください。マッピング時に、IP マルチキャストアドレスのうち **5bit を無視**しました。

つまり、IP マルチキャストアドレスのうち下位 23bit が同一であれば、**異なる IP マルチキャストアドレスが同一の MAC アドレスにマッピングされる**ということです。その例を以下に示します。

IP マルチキャストアドレス①  
239.0.0.1  
11101111 00000000 00000000 00000001

対応付けられる MAC アドレス  
00000001 00000000 10011110 00000000 00000000 00000001  
(hex:01:00:5E:00:00:01)

IP マルチキャストアドレス②  
238.0.0.1  
11101110 00000000 00000000 00000001

対応付けられる MAC アドレス

## multicast

00000001 00000000 10011110 00000000 00000000 00000001

このとき、一つの MAC アドレスに**最大 32 個**の IP マルチキャストアドレスを対応付けることができます。ネットワーク管理者はこの点を注意して、ネットワークを設計する必要があります。

### さて、来週のサザン

さて、そろそろおなかがすいたので今回はここまでです。

皆さん、マルチキャストの概要と、そのアドレッシングについてご理解いただけたでしょうか？マルチキャストと言えど、送信先の指定はユニキャスト同様に MAC アドレスと IP アドレスが用いられていましたね。

マルチキャストにおけるグループという概念は、マルチキャストを支える非常に重要な要素の一つです。次回はそのグループの管理についてお話したいと思います。

### まとめ

- ・マルチキャストの特徴
  - ・特定のホストのグループに対して、同一のデータを送信できる
    - ・映像配信などに最適
  - ・送信元が利用する帯域の負荷を低く抑えることができる
    - ・より多くのホストが参加しても大丈夫
  - ・データを受信したいホストは、任意のタイミングでグループに参加する
- ・グループの識別
  - ・IPv4 ではクラス D のアドレスを利用
  - ・MAC アドレスは IP マルチキャストアドレスから生成する
  - ・MAC アドレスには複数の IP マルチキャストアドレスが対応することがある
    - ・ネットワークの設計には注意が必要

# キモオタでも出来る NAPT 越え

文 編集部 ranha

## 0 はじめに

NAPT 越え(以下 NAPT Traversal)は、分散コンピューティング、特に P2P ソフトウェアを作る上で非常に重要な技術です。今回はその中でも基礎的な部分を中心に、実際に役立てられるようにまとめました。この記事を読むにあたっては、ルータやグローバル IP アドレス、プライベート IP アドレスといった単語を知っていて、UDP や TCP を使ったプログラムを書いた経験があると嬉しいです。もちろん、NAPT Traversal に興味があると尚良いです。

## 1 そもそも NAPT と Traversal って何？

### 1.1 Traversal

先に Traversal から。Traversal は Traverse の名詞形です。そして Traverse は「～を越える、～を横断する」という意味を持つ英単語です。障壁となる NAPT を「飛び越える」というイメージをもってください。

### 1.2 NAPT の簡単な説明

NAPT とは「Network Address Port Translation」を略したもので。文字通り IP アドレスとポート番号を変換する技術の事です。良く似たものに NAT(Network Address Translation)があり、どちらか目にした事がある人も多いと思います。この NAPT と NAT の違いを述べる事も含めて、それぞれを詳しく説明したいと思います。ただし、一般的に NAT という単語は NAPT と同じ意味で使われます。

#### 1.2.1 NAT とは？

まず NAT[1]から説明します。図 1 を参考に読んでください。

図 1 の 2つのコンピュータは、それぞれプライベート IP アドレス a と b を持っています。そしてルータには 1つのグローバル IP アドレス A が割り当てられています。また、2つのコンピュータはこの時、ルータの内側もしくはルータの後ろにあると言います。

NAT がする事は、図中点線で示している mapping に尽きると言えます。mapping とは、グローバル IP アドレスとプライベート IP アドレスを対応付ける事です。この状態で、アドレス a を持つマシンが Web サーバにアクセスしようとすると、パケットの送信元アドレス a を、ルータが勝手にアドレス A に書き換えるのです。そうすると、アクセスを受けた Web サーバは、それがたかもアドレス A から来たかの様に見えます。

しかし mapping が行われている間、他のプライベート IP アドレスを持つマシンが WAN 側にパケットを飛ばす事は出来ません。今回のケースでは、アドレス b と mapping を行うための他のグローバル IP アドレスが存在しないからです。結局、NAT はルータが所持しているグローバル IP アドレス数と同じ数の LAN 内のマシンを、同時に内部から外部へ通信させる手段なのです。

#### 1.2.2 NAT の問題点

先に述べた通り、1 台のマシンに対し mapping が行われている間は他のマシンから WAN になぐ事が出来ません。それこそ、家の中で自分以外の人間が別のマシンでネットサーフィンをしている時はそれが終わるのを待たなければなりません。

ここで少し考えて欲しいのは、完全に 1 対 1 でプライベート IP アドレスとグローバル IP アドレスを対応付けるのは無駄であるという事です。それこそ、Web ブラウジングを行う時やオンライン

## キモオタでも出来る NAPT 越え

インゲームをする時に使うポートは、通常たかだか数個です。ならばポートも使って mapping してやると、同時に複数のマシンで WAN 側と通信する事が出来るのではないか？ そしてそれを行う仕組みこそが、NAPT なのです。

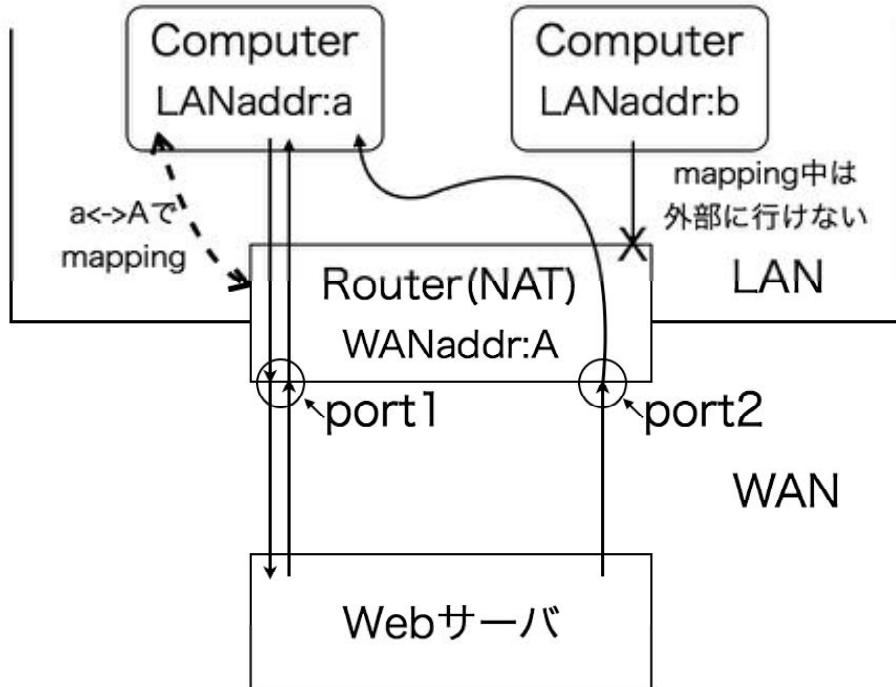


図1.NAT

### 1.2.3 NAPT とは？

いよいよ NAPT についての説明です。図 2 を参考に読んでください。

図中の丸は、ポートを表しています。一つの丸が一つのポートと対応します。

先程と違うのは、同時に 2 つのマシンが WAN 側と通信(Web サーバにアクセス)出来ている事です。これを達成するためにポートを利用した変換を行っているのです。NAPT は、IP マスカレード(IP masquerade)とも呼ばれます。

アドレス a のマシンから Web サーバにアクセスするとしましょう。アドレス a の「マシン側のポート」一つと「ルータ側のポート」一つが mapping されます。後は NAT の時と同様に Web サーバにアクセス出来ます。

次に Web サーバがレスポンスを返す時を考えます。これは非常に簡単で、Web サーバは素直にリクエスト送信元のアドレスとポートにパケットを投げ返してやれば良いのです。ルータは、アドレス A の port1 に WAN 側からパケットが飛んでくると、アドレス a マシンの対応したポートにパケットを送ります。また、マシン側のポートごとに別のルータ側ポートが割り当てられるので、図の様にアドレス a マシンの 2 つのポートはそれぞれ別のポート (port1 と port2) と mapping が行われます。

NAPT はこの様にアドレスとポート両方を利用した変換を行うので、NAT の時とは違い同時に複数のマシンが WAN 側と通信する事が出来ます。

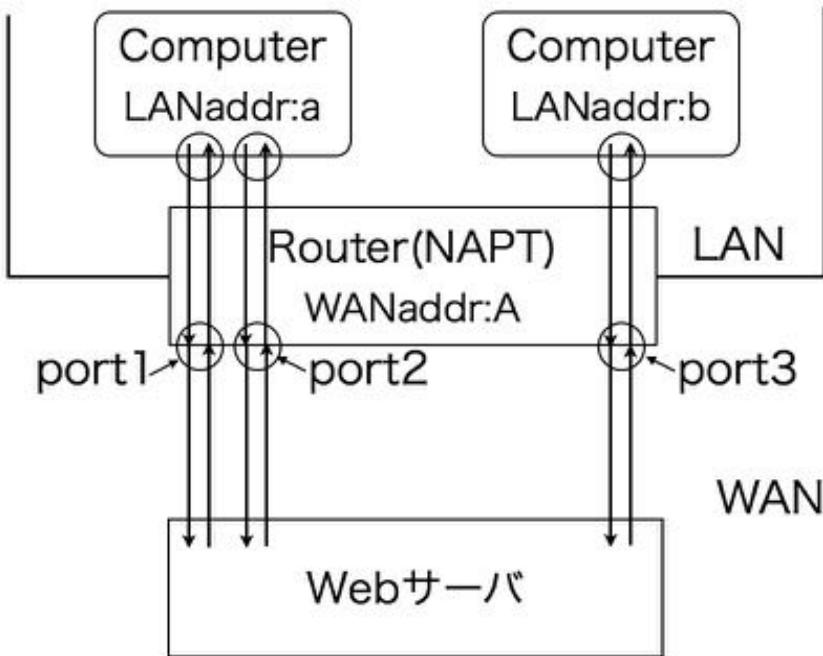


図2.NAPT

## 2 NAPT Traversal とは？

NAPT の説明が終わりましたが、本記事の目的はその NAPT を Traverse する事にあります。NAPT が邪魔であるからこそ、Traverse するのです。では邪魔に成るとはどういった時なのでしょうか？

### 2.1 NAPT は WAN 側からの通信開始時に障壁となる

例えば皆さんがあるマシンのIPアドレスを変換するルータ（NAT）を経由して、別のマシンにデータを送信する場合、NAT側でポート番号を割り当てる作業が必要になります。この作業によって、相手側のマシンがデータを正しく受け取るために必要なポート番号を知らなければなりません。

この時相手側のマシンがデータを正しく受け取るために必要なポート番号を知らなければなりません。そのためには、相手側のマシンがデータを正しく受け取るために必要なポート番号を知らなければなりません。そのためには、相手側のマシンがデータを正しく受け取るために必要なポート番号を知らなければなりません。

相手側のマシンがデータを正しく受け取るために必要なポート番号を知らなければなりません。そのためには、相手側のマシンがデータを正しく受け取るために必要なポート番号を知らなければなりません。

### 2.2 ポート番号を知る術

相手側のマシンがデータを正しく受け取るために必要なポート番号を知らなければなりません。そのためには、相手側のマシンがデータを正しく受け取るために必要なポート番号を知らなければなりません。

## キモオタでも出来る NAPT 越え

mapping された後にルータ側のポート番号知る事は出来ないのですが、mapping される前にルールを設定しておいて、その通りに mapping させる手段はあります。それは静的ポートマッピングや静的 IP マスカレード、ポート開放と呼ばれる方法です。これは、LAN 内の「あるアドレスの特定のポート」と「ルータ側の特定のポート」を mapping するための方法です。例えば、192.168.1.2 のポート 80 を、ルータ側のポート 80 とで mapping せよ、というルールを作ったとしましょう。この状態で WAN から 80 番ポートにアクセスすれば、無事に 192.168.1.2 のポート 80 番にアクセスする事が出来ます。自宅サーバを立てた事がある人は、ポート開放のお世話に成った事があるはずです。

静的ポートマッピングは基本的に手動で行う事になります。普通は Web ブラウザからルータ設定ページにアクセスして編集します。この設定は皆が簡単に出来るかと言えばそういうわけではありませんし、更にアプリケーションごとに逐一静的ポートマッピングするのもどうにも面倒くさいです。どうにか成らないのでしょうか…。

### 2.3 NAPT という壁を越える

WAN 側からルータの後ろに存在する特定マシンに対して通信を「開始」したいが、適切なポート番号が分からない…というこの状態こそが、NAPT が障壁であるという事なのです。そしてこの障壁をなんとかして乗り越えるのが NAPT Traversal なのです。

## 3 NAPT Traversal 手法紹介

### 3.1 Server-Relay による間接双方向通信

NAPT を越える方法の一つ目です。サーバを間に挟んだ Server-Relay を紹介します。

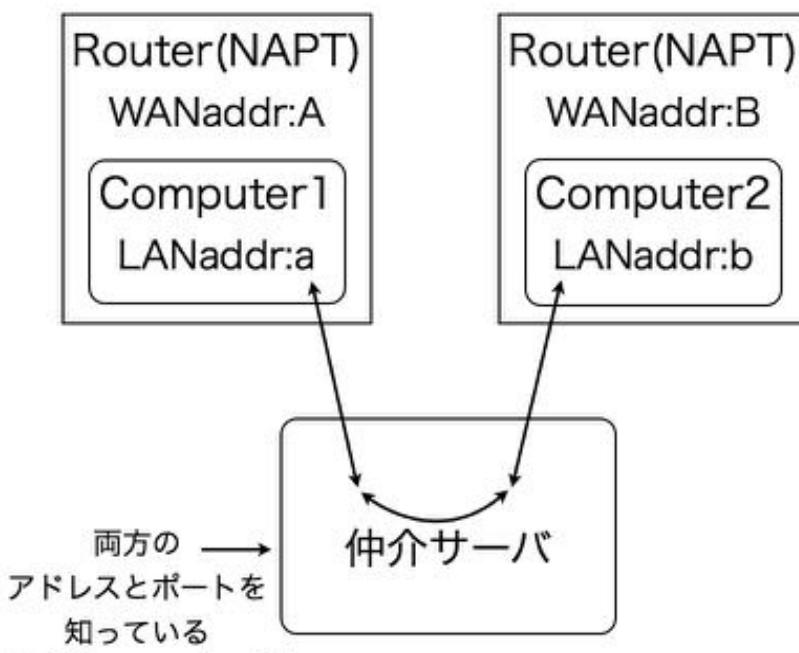


図3. Server Relay

図 3 を見てください。重要なのは、仲介サーバは両方のグローバル IP アドレスと、適切なポート番号を知る事が出来る点です。これはサーバ側で、受け取ったパケットの送信元アドレスとポートを見れば済みます。仲介サーバは、Computer2 のアドレスとポートに対して、Computer1 から受信したデータをそのまま転送してやります。これがリレー (relay) です。

しかし、この Server-Relay には問題があります。それはリレーのためのコストがかかるという事です。Computer1 が何らかの方法で、「直接」 Computer2 にデータを送る場合と比べてみてください。全て送信し終わるのにかかる時間はどちらの方が短いでしょうか？また、仲介サーバは誰が準備するのでしょうか？

ただ NAPT は越えられますから、リレーコストをどうにか出来ると考える方や、高速・低負荷 Relay-Server の実装にやりがいを感じる方は是非とも挑戦してみてください。

### 3.2 Hole Punching

Server-Relay とは別に、UDP Hole Punching と TCP Hole Punching を紹介します。これが本記事の肝です。先に UDP Hole Punching の説明をして、次に TCP Hole Punching の説明を行います。

## 4 UDP Hole Punching

UDP Hole Punching はいくつかの RFC でも取り上げられる程、有効な方法です。ただし、確実な方法ではない事に注意してください。図 4 を参考にしてください。各 step ごとに何をするかを説明します。注意すべきは、UDP Hole Punching で使うプロトコルは UDP だという事です。

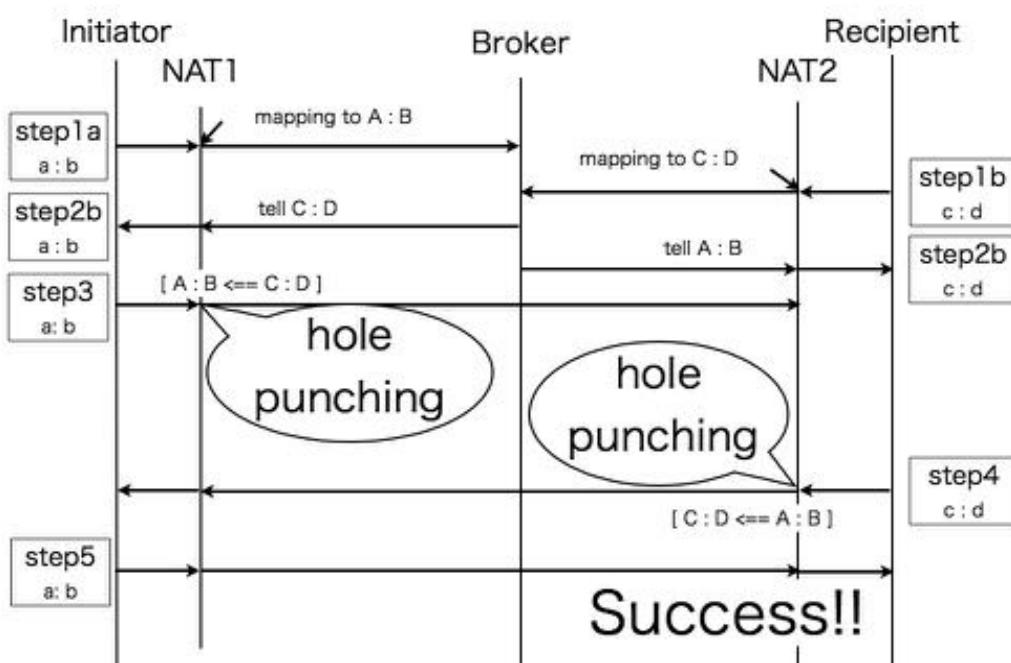


図4. UDP Hole Punching

## キモオタでも出来る NAPT 越え

### 4.1 UDP Hole Punching 解説

まず step1a、step1b では Broker にパケットを送ります。Broker は仲介サーバの役割を果たします。具体的には Initiator と Recipient のグローバル IP アドレスとルータ側ポートの把握と管理です。この step では、Broker に対して自分のグローバル IP アドレスとポートを通知するだけですので、どちらが先でも構いません。

次に step2a ですが、これは Initiator に Recipient のアドレスとポート情報を、step2b で Recipient に Initiator のアドレスとポート情報を送ります。勿論順番は問いません。

この段階で、Initiator と Recipient は共に通信したい相手のグローバルアドレスとルータ側ポートを把握している事になります。

step3 で NAT1 に対して hole punching します。ルータに対して送信先アドレスとポートを覚えさせるのです。少し詳しく書きます。

ルータは、受け取ったパケットが不正でないかを調べるためのテーブルを持っています。hole punching するとは、そのテーブルに新しく要素を追加する事だと言えます。今まさに [ A : B <== C : D ] という、アドレス A のポート B に対するアドレス C のポート D から来たパケットは正当なものである事を表す要素をテーブルに追加したのです。通常、テーブル内のルータ側ポートに対応するエントリに含まれない所から飛んできたパケットはルータ側で破棄するので、LAN 内のマシンには届かないようになっています。

step4 では今度は NAT2 に対して hole punching します。それと同時に、この段階で Initiator は Recipient の送信したパケットを受信する事が出来ます。アドレス A のポート B から来たルータ側ポート D への通信は正当である事を表すエントリが、既に step3 でテーブルに追加されているからです。

さて、最後に step5 で Initiator からパケットを送信してみましょう。NAT2 側のテーブルにもエントリが追加されていますから、今度はちゃんと Recipient まで届きます。

hole punching と仰々しく書きましたが、LAN 内から WAN 側にパケットを飛ばす時は常にしている事なのです。今回はそれに注目して貰うためにあえて書きました。

これが UDP Hole Punching です。しかし先ほど、確実な方法ではないと書きました。その理由を説明しなくてはなりません。成功するかしないかを決めるのは、ルータのタイプです。

## 5 ルータのタイプ分けと挙動

ルータのタイプには 4 つあります[5]。

それぞれ、Full-Cone、Symmetric、Restricted、Port Restricted と呼ばれます。

### 5.1 Full Cone NAT

まずは Full Cone の説明です。Full Cone NAT 側のマシンは、NAT に対して Hole Punching を行う必要はありません。図 5 を見てください。Full Cone では、アドレス A のポート P に対して飛んできたパケットは送信元がどこであるかを問わず LAN 側のマシンに通します。

### 5.2 Symmetric NAT

Symmetric NAT は最も越えにくい NAT です。特に厄介なのは、同一アドレスとポートからパケットを送っても、送信先によって mapping されるポートが代わる事です。図 5 の最後に、Initiator は Recipient にパケットを飛ばすのですが、ここで Symmetric NAT は A:P1 に mapping するのではなく、A:P2 とポートを変えて mapping します。このため、Recipient の NAT が Full Cone でないと

失敗してしまうのです。

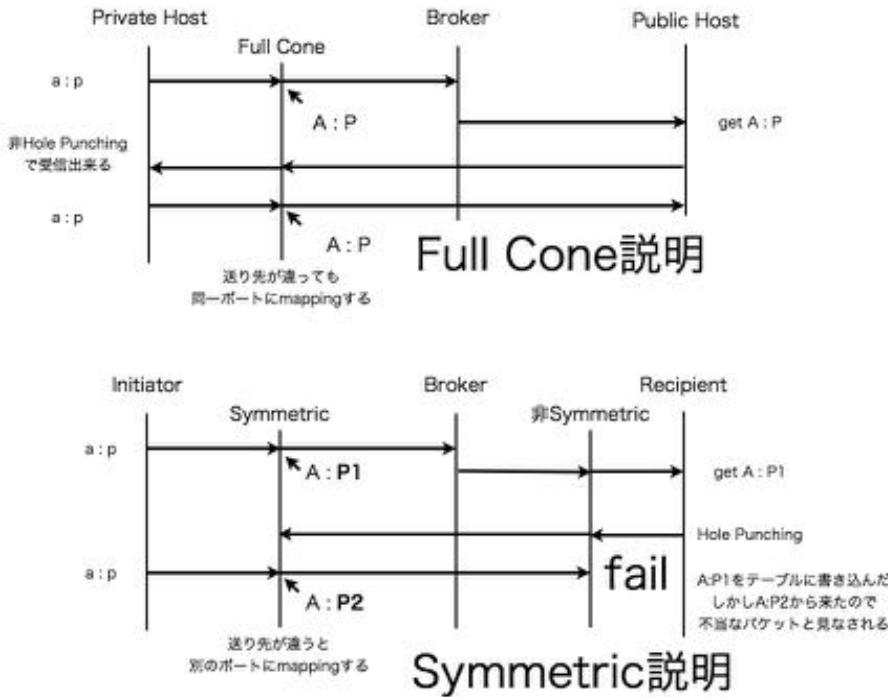


図5

### 5.3 Restricted NAT

Restricted も Full Cone と同じく、送信先が違っても同一ルータポートに mapping を行います。図 6 を見てください。

Restricted NAT では hole punching する時に相手アドレスの任意のポート(テーブルエントリの\*!は任意のポートを表します)から来る通信は正当であるというエントリを追加します。

図において、B のポート P1 からパケットを飛ばした時、これはいたって普通の通信なので成功します。更に、B のポート P2 からパケットを飛ばした時も成功します。これは、ルータが送信元のアドレスだけを見てパケットの正当性を判断しているからです。ただし、アドレス C が A:P にパケットを送ってそれを許可するエントリはテーブルには存在しないために、不当な通信と見なされ破棄されてしまいます。

### 5.4 Port Restricted NAT

Port Restricted NAT は Restricted NAT と似ていますが、これは送信元ポートも見てパケットが正当であるかどうかを判断します。

hole punching 時にルータのテーブルに「ルータポート P に対して、グローバルアドレス B のポート P1 から来るパケット」は正当なものであるというエントリをテーブルに追加します。ですから、B のポート P2 から来るパケットは Restricted の時とは違って不当な通信と見なされてしまい破棄されるのです。当然 C から来るパケットは全て破棄されます。

## キモオタでも出来る NAPT 越え

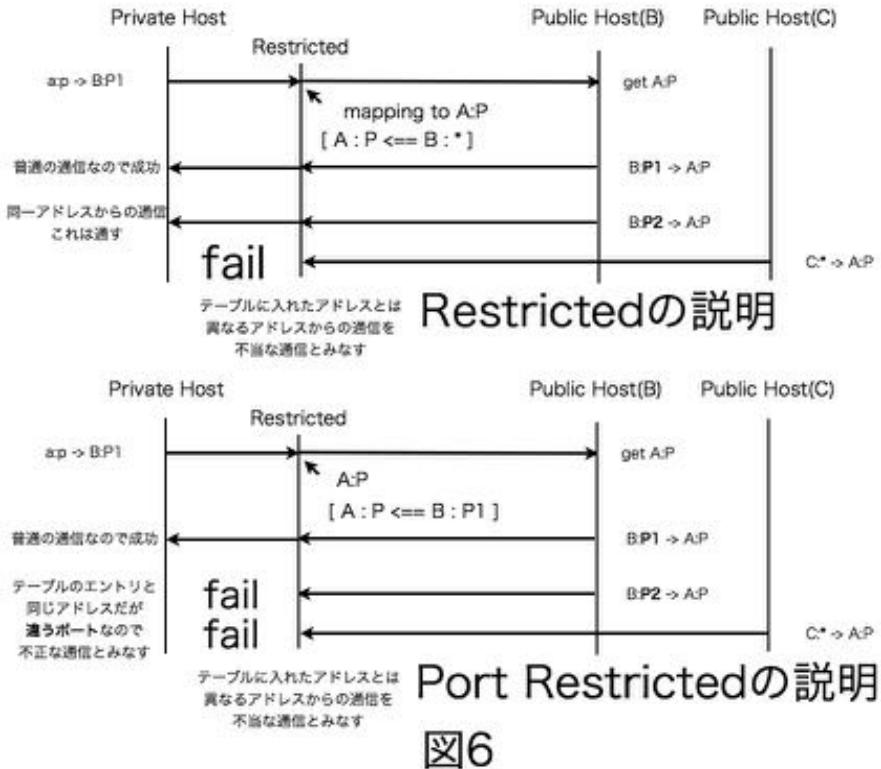


図6

## 6 TCP Hole Punching

今回説明するのは TCP Hole Punching のうち、NatTrav[6]と呼ばれる方法です。

UDP Hole Punching と大体同じ手順で行います。ただし、利用するプロトコルはその名の通り TCP です。

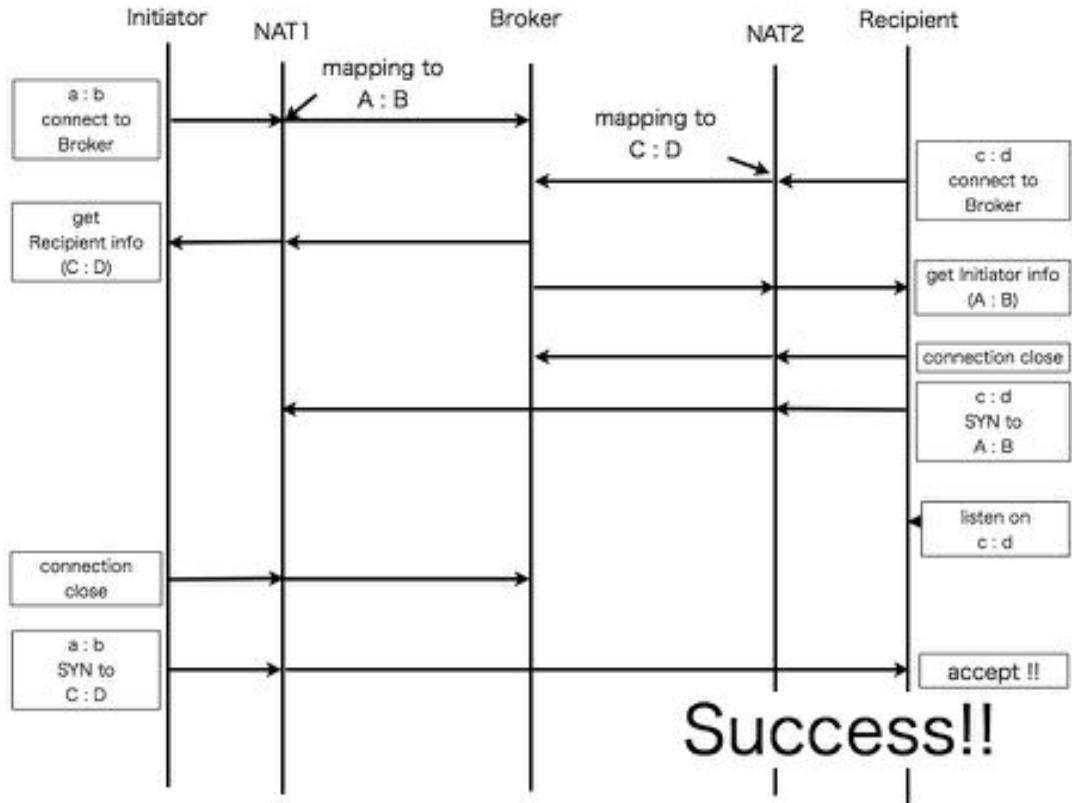
### 6.1 NatTrav 解説

図 7 を参考に読んでください。

まず Initiator、Recipient 共に仲介サーバ(Broker)に connect し、それぞれのグローバル IP アドレスとルータ側ポート番号を得ます。

次に Recipient は、Broker につないだ時と同一アドレス/ポートを用いて Initiator に connect するために、まず Broker に対しては close します。ここで重要なのは re-bind する時に”SO\_REUSEADDR”(相当の)オプションを用いる事です。SO\_REUSEADDR を使うのは、close した後にすぐ再利用出来る点などにあるのですが、詳しい説明は省きます。そして次に Recipient は hole punching するために Initiator に SYN を送ります。これは単純に connect すれば良いです。NAT1 側では A : B で listen していないので connection が確立する事はありません。Recipient は SYN を送った後に、相手側から来る SYN に備えるために listen & accept します。これで Recipient 側の準備は完了です。

Initiator は Broker との connection を close して、それから Recipient に対して SYN(connect すれば良い)を送ります。特に問題がなければ、Initiator と Recipient で直接 connection が確立されます。



## 図7.TCP Hole Punching

### 6.2 TCP Hole Punching は難しい？

TCP Hole Punching は UDP Hole Punching と比べると成功率がどうしても低くなります。TCP が多くのアプリケーションプロトコルのベースに成っているため、ルータ側で極力不正なコネクションを張らせない様にするからです。特に SPI(Stateful Packet Inspection)の機能が働くルータの場合、TCP Hole Punching を行うのはかなり難しくなります。

SPI は、SYN を送った相手から返ってくる ACK のような極めて正常なパケットしか通さないように振る舞います。TCP Hole Punching で connection が確立出来るのは、最終的に Initiator が送る SYN が Recipient まで届けられるからです。SPI が働いている状況下ではそれは許されません。ACK フラグを立てる様な単純な偽装も成功しません。SPI については、割とルータを作るベンダ秘匿な所もあるらしいので詳しい所は出てきませんが、調べると多少の情報は出てきます。興味がある人は自力で調べてみてください[7]。

## 7 その他の Topics

いくつかの話題を簡単に取り上げたいと思います。

## キモオタでも出来る NAPT 越え

### 7.1 UPnP[3]

ルータの箱に、UPnP 対応と書かれているのを見た事がある人もいるのではないでしょうか。UPnP は NAT Traversal にも十分使える技術です。簡単で高信頼な方法とも言えるので、NAPT 越えを検討する際には考慮してみてください。

### 7.2 UDP Multi Hole Punching[8]

Symmetric ルータ相手でも UDP Hole Punching を行うための手法です。

Symmetric ルータ相手では、送信先ごとに mapping されるルータ側ポートが異なってしまう事が原因で、適切なポート番号を把握する事が困難と成っていました。しかし、例え Symmetric であっても mapping のルールが存在します。そのルールを把握するために、大量のパケットを送り、大量の (Multi) Hole Punching を行って成功確率を上げるための手法です。

### 7.3 その他の TCP Hole Punching

#### 7.3.1 TCP Simultaneous Open[9]

これは、通常 TCP が 3-way-handshake と呼ばれる 3step を踏む方法とは異なる、4-way-handshake とも言える方法です。

ただし、これは SPI が働いている場合失敗します。ACK どころか、いきなり相手から SYN が飛んできますから。また、RFC793 記載の TCP 状態遷移図[10]を見ても正しい事は分かるのですが、多少イレギュラーなのでプロトコルスタックの実装によっては失敗するようです。

#### 7.3.2 NUTSS[12]

STUN[5]という、標準化された UDP による NAPT Traversal のためのアプリケーションプロトコルがあります。

これを TCP にも拡張した STUNT[11]が存在します。そして、その STUNT-Server を用いる手法が NUTSS です。

NUTSS では、STUNT-Server と呼ばれる仲介 Server を用いて、パケット中のソースアドレスとソースポートに対して偽装を行う事で、SPI にも対応した TCP Hole Punching をする事が出来ます。

#### 7.3.3 NatBlaster[13]

NatBlaster も NUTSS と同じ様にパケットの偽装を行います。こちらは SYN と ACK を偽装します。更に、Symmetric にも対応するべく策が練られています。LSR (Loose Source Rooting) という、IP パケットが経由すべきルータを送信元で指定する事が出来る方法を用いて、例え Symmetric でランダムにルータ側ポートが割り当てられたとしてもそれを経路上で捕まえて覗き見るので。

しかしながら、ソースルーティングという技術は悪用される事が多いらしく、あまり宜しくないようです。LSR が使えない時は、UDP Multi Hole Punching の時と同じく大量のパケットを飛ばす事で何とか connection を確立しようと試みます。偽装により、結果的に正当なパケットと見せかけるので、こちらも SPI に対応しています。

## 8 おわりに

自分達の知らない所で NAPT Traversal が行われている事は多いものです。例えば、Windows Live Messenger は UPnP を、Skype は UDP Hole Punching を行っている事が知られています。

NAPT Traversal がすぐにでも使える分野としては P2P や広域分散などがあるでしょう。また局

所的な負荷分散にも用いる事が出来ます。

ともかく、今回の記事が皆さんにソフトウェアを作る上で役に立ってくれると嬉しい限りです。

新しいネットワークファイルシステムを作ったる！とか、ニコニコ動画の新着ランキングの部分で P2P が出来るももっと軽くなるんじやないか！？とか、ブラウザで P2P 出来ると良いんじやないだろうかとか、沢山アイディアはあると思います。

P2P は世間的には悪いイメージを持たれがちです。是非とも皆さんに、P2P を用いた素晴らしいソフトウェアを開発してそんなイメージを払拭してください！！

## 9 References

[1] ネットワークアドレス変換 Wikipedia <http://ja.wikipedia.org/wiki/ネットワークアドレス変換>

[2] IPv4 アドレスの在庫枯渇に関して <http://www.nic.ad.jp/ja/ip/ipv4pool/>

2010 年の冬には日本の IPv4 アドレスが無くなる

<http://www.atmarkit.co.jp/news/200805/29/nttcom.html>

[3] UPnP Forum <http://upnp.org/>

UPnP の動作 <http://bb.watch.impress.co.jp/cda/bbword/10002.html>

[4] Traversal Using Relays around NAT (TURN)

<http://www.ietf.org/internet-drafts/draft-ietf-behave-turn-12.txt>

[5] RFC3489 STUN <http://www.ietf.org/rfc/rfc3489.txt>

[6] NatTrav <http://reports-archive.adm.cs.cmu.edu/anon/isri2005/abstracts/05-104.html>

[7] SPI とパケットフィルタリング <http://bb.watch.impress.co.jp/cda/bbword/7731.html>

[8] UDP Multi Hole Punching <http://www.goto.info.waseda.ac.jp/~wei/file/wei.pdf>

[9] Peer-to-Peer Communication Across Network Address Translators

<http://www.brynosaurus.com/pub/net/p2pnat.pdf>

[10] TRANSMISSION CONTROL PROTOCOL RFC793 <http://www.ietf.org/rfc/rfc793.txt>

[11] STUNT <http://nutss.gforge.cis.cornell.edu/stunt.php>

[12] NUTSS <http://nutss.gforge.cis.cornell.edu/index.php>

A SIP-based Approach to UDP and TCP Network Connectivity

<http://nutss.gforge.cis.cornell.edu/pub/fdna-nutss.pdf>

Towards a Secure Internet Architecture Through Signaling

<http://nutss.gforge.cis.cornell.edu/pub/cucs06-nutss/>

[13] NAT Blaster <http://natblaster.sourceforge.net/>

### コミケへのご参加は計画的に

文 編集部 mitty

#### 走れ<sup>\*1</sup>mitty<sup>\*2</sup>

mitty は激怒した。必ず、かの有名な同人誌即売会に参加せねばならぬと決意した。mitty は絵心がわからぬ。mitty は、筑波の AC である。Perl でコードを書き、サーバと戯れて暮してきた。けれどもお祭り騒ぎに対しては、人一倍に敏感であった。

筑波大学に入学する前、mitty はとある大学にいた。だらしない mitty は卒業することなく追い出されてしまったのであるが、2002 年入学当時から在籍しているサークルの活動には、まだ、細々と参加しているのだ。今年も、年一回のサークル展示に陣中見舞いする為、11 月 22 日から三日間 mitty は筑波を出発し、TX を使い高速バスを使い、二十里は離れた此の駒場の学園祭にやって来た。駒場祭において、そのサークルは自作のプログラム展示をしている。毎年奇抜なゲームが楽しめるその場を、mitty は心待ちにしていた。



mitty にはとある後輩があった。三年連続で駒場祭にプログラム展示をしている。毎年進化するそのアクションパズルゲームに、今年はどんな新機能が追加されたのか楽しみにしているのだ。

大気君「見ろ、参加者がごみのよう(ry)

ひとしきり、展示されているゲームをやり込み、ハイスコアを見て一喜一憂した mitty は、その後輩に今後の展望を聞いてみた。そして、C73<sup>\*3</sup> でそのゲームが頒布された事を知った。

mitty はコミケに参加した事がなかった。知人の参加報告をチャットや Blog の写真で知つては、「買ひすぎ wwwwww」と条件反射するのみであった。

後輩は楽しげな声で、弾むように答えた。

「年末も、サークル参加します。」

「なん……だと……。スペース<sup>\*4</sup>はどこなのだ。」

\*1 「(1)絶対に走らない・人を押さない」 - C75 カタログ p.8 より

\*2 この文章は「走れメロス(太宰治)」のパロディです。

\*3 コミックマーケット 73 のこと。コミックマーケットには開催順に通し番号が付けられている為、通算 73 回目の開催ということになります。C73 は 2007 年の年末に開催。今回の二つ前。

\*4 コミケ会場において、サークルに割り当てられる販売スペースのこと。SP と言つたりするらしい。コミケくらいに大規模になると、あらかじめこれを調べておくないと、当日になってから現地で目当てのサークルスペースを探して回るのは**物理的に不可能です。** そういう意味でも、カタログの購入は強く推奨します。

「ちょっと今は覚えてないので。公式サイトの方を見てください。」

聞いて、mitty は激怒した。「大した後輩だ。陣中見舞いせねばならぬ。二学期ゆっくりしすぎて情報特別演習やばいとか言っている奴はどこの私だ。ああ最近だらしないな<sup>5</sup>。」

mitty は単純な男であった。その場で「じゃあ顔出しに行くよ」と言ってしまった。

参加するとなると一日仕事である。

「情報特別演習の最終発表会は 12 月 25 日だ。それはコミケ前だから、まあいい。その後、年明けにレポートの提出がある。大丈夫なのか。」

「出来る出来る絶対出来る。」

「おまえがか？」 mitty の中の、理性的な部分は、憫笑した。「仕方のない奴だ。おまえには、反省の二文字がわからぬ。」

「言うな！」と mitty はいきり立って自分に反駁した。「だからといって、やらないうちから疑うのは、もっとも恥すべき悪徳だ。」

「疑うのが、正当の心構えだと、自分に知らしめたのは、おまえ自身ではないか。人の心は、あてにならない。mitty は、もともと怠惰な AC さ。信じては、ならぬ。」

「なんの為の冬休みだ。買って来た薄くて高い本を楽しむ為か。」

「だまれ、痴れ者。」内なる mitty は、さっと報いた。「余裕があるうちは、どんな無茶な事でも言える。提出期限前日になってから、泣いて先生に詫びたって、間に合わぬぞ。」

「ああ、理性は頑固だ。どんなに言い訳したって、もう参加することは確定的に明らかなのに。後戻りなど決してしない。ただ、」

そこまで考えて、mitty はしばしとまどい、「提出までに三日間の余裕を持つようにスケジュールを組もう。たった一つのレポートで、単位を落とすようなことにはしたくないから。三日のうちに、私はきっとレポートを仕上げ、必ず情報特別演習の単位を取ってみせる。」

「馬鹿な。」と mitty は低く笑った。

「既に 1 月 9 日ではないか。一行も書いてないレポートを、13 日までに仕上げるというのか。」

「そうだ、仕上げるのだ。」

そして mitty は気付いた。レポート以前に WORD のあきたこまち記事がまだ仕上がってない…！

「ゆっくりした結果がこれだよ！！！」

---

\*5 まあ、抑鬱状態(いわゆる鬱病)だったので仕方ないといえば仕方ないね。

## C75 参加顛末記

という訳で、C75 参加してきました<sup>\*6</sup>。前置き長くてすみません。

今回は初参加<sup>\*7</sup>ということで、かなりゆっくり参加<sup>\*8</sup>してきたのですが、それでも色々と思うところがあったので日記風(?)につらつらとまとめてみます。

### ・12月20日

東京まで出かける用事があるので、道中とらのあなに寄り、紙のカタログ<sup>\*9</sup>を購入。師走も暮れに近いのにレジ前に長蛇の列が出来ていて、購入まで 10 分近く掛かる。CD-ROM 版<sup>\*10</sup>もあって、そちらの方は検索機能が使って便利らしいが、一度紙カタログの重さを自分で持って体感してみたかったのであえて紙で。**まぁそんなに色々回らないだろうし(フレグその1)。**

しかし、25 日の情報特別演習最終発表会のプレゼンテーションで用いるプログラムの、実装がまだ終わってないのに東京まで出かけるとか、mitty は締切り間近でも構わないでゆっくりする人間だから仕方ないね。



\*6 何故かことごとく知り合いから、「初参加？ HAHAHA、**ご冗談を。**」とか言われるんですが、本当に初ですよ？

\*7 エコケット 10 というかなりマイナー(というと失礼かも知れないが)なオンリーイベント(元となる作品や、ジャンルを限定した即売会。逆に、コミックマーケットは「オールジャンル」と呼ばれ、文字通りどのようなネタであっても OK)に参加したことはあるんですけどね。ちなみにエコケットとは何かというと、東北電力がかなり昔(前世紀)にエコアイス(深夜電力で氷を作つておいて、日中はその氷を使って空調コストを削減しよう、というソリューション)の販促目的で用意したキャラクターがあり、その姿があまりにも可愛らしかったので人気となって、「えっこ」という名前が非公式で設定されたあげく、何の因果かオンリーイベントまで開催されてしまった、という…。しかも、私が参加した目的は「えっこ」ではなくて、同時開催されていた「任意なん」イベントの「¥ e」の方でして。…話がどんどん逸れるのでこれくらいで止めておきます。

\*8 到着が 11 時過ぎました。ちなみにコミケ会場の開場は 10 時。

\*9 通称**「鈍器」** or **「電話帳」**。そう呼ばれても仕方ないのはその重量のせい。**2kg 近くあるから困る。**

\*10 通称カタログ。毎年改良されて、目的のサークルがより探しやすくなっているそうです。最近某界隈で色々動きがあった DS マジコンとか、PSP とかにデータを突っ込んで、閲覧できるようにしている人も居るみたい。

ところで、知り合い二十人くらいにコミケに参加するかどうか聞いてみたら、スタッフ参加<sup>\*11</sup>が一人、サークル参加<sup>\*12</sup>が四人、一般参加<sup>\*13</sup>が私を除いて五人以上だった。どういうことなの…。

#### ・12月25日

明け方、それまで動いていた情報特別演習最終発表会プレゼンテーション用のプログラムがいきなり動かなくなる<sup>\*14</sup>。ひとしきり阿鼻叫喚したあと、マシンを再起動したら普通に直ったので一安心。

プレゼンテーション自体は特にトラブルもなく無事終了。これで安心してコミケに参加出来るぞ…！

#### ・12月28日

遅まきながらサークルチェック<sup>\*15</sup>を始める。pixiv<sup>\*16</sup>を巡回すると結構すぐ回りたいスペース溜まっちゃうんだね、仕方ないね。

チェックしながらチャットで情報特別演習の話をしていたら、プレゼンテーションで使用した動画を「うｐ！うｐ！」と言われて、ついカッとなつてニコニコ動画に投稿したりと色々変なことになる。

あと、そのチャットで何故か LAN ケーブルを自作する話で盛り上がる。同軸ケーブルの時代からの自作経験者によると、「ping したら 70% loss するケーブルとか、割と簡単につくれるｗ」らしい。怖い怖い。

「サークル参加の抽選<sup>\*17</sup>落ちたので委託<sup>\*18</sup>するー」という知人がいたので、委託先について調

\*11 コミケの設営とか運営を行うスタッフ。会場(東京国際展示場、いわゆる「東京ビッグサイト」)を借りたり、官公庁と連絡を取ったりなど、コミケの準備や運営を行なう際にどうしても必要となる法人窓口としては「有限会社コミケット」が存在していますが、コミケの準備・運営のほとんどはボランティアベースで行われています。

\*12 販売する側、ということですね。

\*13 前者二つ以外の参加者のこと。この人たちが「コミケ参加者」の中で一番多いのは言うまでもありません。

\*14 サンタさんが「もう頑張らなくても良いよ」って言ってくれたのかも知れないけれど。

\*15 コミケカタログに載っているサークル紹介(「サークルカット」と呼ばれる小さな絵。実際に会場で頒布されるものと違うものになっていることがよくあるらしい。)を見たり、各サークルのウェブサイトなどを巡回して、買い物リストを作成すること。

\*16 画像の投稿と閲覧に特化した SNS。Wikipedia、ニコニコ動画、ニコニコ大百科で懲りたはずの mitty から、貴重な(?)時間を容赦なく奪っていく魔物。前述のコミケサークルカットをユーザプロファイルに追加する機能があり、お気に入りの絵描きさんのサークル情報を簡単に確認できるようになっている。

\*17 会場の広さは有限なため、参加を申し込んだサークル全てが実際に参加できる訳ではなく、抽選の結果一部のサークルは涙を飲むことになるわけです。

\*18 自分のスペース以外で制作物を頒布してもらうこと。他のサークルに頼む「サークル委託」や、同人誌専門店に卸す「ショップ委託」などがあります。大手サークルだと、コミケ会場での頒布とは別にショップ委託を用意していることもあります。

## C75 参加顛末記

べていたらニコニコ動画上ではかなり有名な人でびびる。

ある程度チェックし終えたカタログを分冊化<sup>\*19</sup> する。ガムテープが見つからないので、一の矢共用棟の COCO ストアまで買いに行くが置いてないと言われる。仕方がないのでホッチキスで代用しようとするが、普通の印刷用紙で数枚分の厚さしかないので、分冊化したコミケカタログだと 40 枚もあって<sup>\*20</sup> 全く歯が立たない。結局、部屋中探して見つけたガムテープで片を付ける。



ばらされゆくカタログ。「勿体ない！」という人は保存用にもう一冊どうぞ。

\*19 分冊化せずに丸ごと持って行くと、非常に重いので非推奨。開催日は三日間に分かれているので、それぞれの日で必要なのは単純計算で 1/3 ずつですし。なお、分冊化の際は、ガムテープなどで補強しないと、すぐばらばらになってしまうので注意。

\*20 「B5 で厚さ 30mm 程度のサイズなのに、2kg 近い重さ」を実現する為にかなり良い紙を使っているので、仕方ないですねえ…。分解する際はカッターナイフを使ったのですが、面白いようにスッと切れて楽しい。ついでに指も切りそうになりましたが。

コミケに限らず即売会の基軸通貨は 100 円玉<sup>\*21</sup> なので、一週間前くらいから買い物の際はなるべく紙幣で払って小銭を貯めていたのだが、あまり集まらずこれで大丈夫かなあ…という印象。

1000 円は企業ブースとか大手サークルとか回るとたぶん大量に使うのだろうが、今回はあまり出番がなさそう。

夜、pixiv で大変素晴らしい缶バッジを見つける。欲しい！必ず手に入れてやる！(フラグその 2)



いきなりですが、ここで注意ポイントの説明<sup>\*22</sup>。

### ●サークルチェック

- ・紙のカタログだと最初に白地図があります。CD-ROM 版(カタロム)だと、サークルの配置を直接書き込んでからプリントアウト出来るデータが入っているそうです<sup>\*23</sup>。地図はそのままだとペラペラの紙で、当日書き足したりしにくいので、クリップボードをあわせて持って行くと吉。今回はクリップボードを用意し忘れたので、手帳に載せて追記したりしました。

- ・コミケ以外の即売会はよく知らない<sup>\*24</sup> のですが、コミケの場合はジャンル<sup>\*25</sup> ごとに大まかなブ

\*21 100 円玉の他に 500 円玉もよく使いますね。郵貯の口座がある人は大学会館前の ATM で「900 円」とか打ち込んで何度か引き出すと良いでしょう。

\*22 まだ一回しか参加したこと事ない癖に先輩面して…とか言われそうですが、許してやって下さい。

\*23 カタロムの方は買ってないので不明です。

\*24 もっとも今回の記事も、大部分は人から得た情報によって成っているわけですがね。

\*25 どの作品の二次創作か、とか、ここはオリジナル創作系だ、とか。

## C75 参加顛末記

ロックとして「地区<sup>\*26</sup>」が設定されています。カタログは「地区」ごとにページ分けされている為、まず目的のジャンルがどの辺りに載っているかを目次で調べてから、各サークルカットを細かく見ていくと良いでしょう。

・コミケにおけるサークルスペースの単位は、「長机半分」です。この長机が連続して設置されている一つの塊を「島」と呼ぶのですが、「島」には列ごとに名前が付けられており、地区名とあわせて「東 Z」や「西い」などと呼称します。東館、西館はそれぞれ「東 1」～「東 6」と「西 1」～「西 4」地区に分かれている為、より正確に言うと「東 2Z」や「西 2 い」となります。これに「行番号」を付け加えることで、サークルスペースが特定できることになります<sup>\*27</sup>。

### ●持ち物

・一つのバッグ、リュックサックだけで行動するのはかなり大変です。「肩に掛けた状態で、カタログがスッと出し入れできるトートバッグかそれに類する物」をメインのバッグとは別に用意しておくと、

1. 購入
2. まずはトートバッグに入れておく
3. ある程度溜まつたら、壁際など人が居なくて開いているところで背中のリュックに移す
4. 1.に戻る

というループが構成できて楽です。あと、列に並ぶときは、リュックは前に持ったり下におろしたりしましょう。まあこれは満員電車などの常識なので、そんな変な事じやないですね。

・雨具は、普通サイズの傘を持ち込むと危険です。折り畳み傘にしましょう。また、コミケの販売スペースは屋内ですが、開場前に並ぶ場合は屋外に並びます。しかも、数時間単位で並ぶ<sup>\*28</sup>ことになるので、帰りの列に並んでいる間に大漁の戦利品がバッグにしみこんだ雨で台無し、という悲劇をたまに聞きます。よく訓練された参加者は、市販のゴミ袋などを必ず持ち込んで、雨模様の時はそれでバッグごと覆ったりするそうです。今回は全日晴模様だったので、雨具の出番はありませんでした。

・財布から使わないカードは抜いておきましょう。学生証やキャッシュカードを落としても絶対気付きません、あの混み様だと。あと、財布自体を紛失してもらちゃんと帰還できるように、予備の財布を用意すると安全です。何が起こるか分からない場ですから…。

・交通費は基本的に Suica(か PASMO)でどうにかなります。というか現金で用意すると「購入で使い切ってしまって帰れない…」という大惨事が発生しかねないので、利便性も考えて Suica を

\*26 ただし、一作品毎に一地区、などとしているととても場所が足りない…というより分類が多くなって非効率的なので、参加サークルの数がある程度以下のジャンルはより大きなジャンル(例を挙げれば、「ゲーム(電源不要)」、「オンラインゲーム」、「ゲーム(RPG)」といった具合です)にまとめられて各地区に配置されます。作品単独で地区化しているのは「ガンダム」や「鋼の錬金術師」、「ワンピース」、「テニスの王子様(ミュージカル除く)」などです。この空間的な「地区」による分類と、「何日目」という時間的な分類が組み合わさっているので、目的とするジャンルの配置を調べるときはこの両方を意識する必要があります。

\*27 正確に言うと、これで特定できるのは長机一個までなので、さらにサークルごとに「a」「b」と分けられています。

\*28 もちろんお帰りの際も。買い物をしている時間より、並んでいる時間の方が長いのが普通です。

推奨します。

・夏コミだと飲み物の用意が特に重要らしいですが、冬も屋内はそれなりに暑い上に乾燥しているので、500mlくらいのペットボトルを持って行くのがよいでしょう。自販機があるそうですが、そんなものを探す余裕はないです。陣中見舞いする場合も、何か飲み物<sup>\*29</sup>を持って行ってあげると喜ばれます。

・暇つぶし道具があると、並ぶのもそれほど苦痛ではありません。最近は DS などの携帯機器がかなり普及しているので、結構ネタには困らないんじゃないかな、と思います。あるいは、待っている間周囲の人と濃い会話をすると、というのもあります。「午後から行けばそんなに並ばないでしょ」という人は、**帰りの際に乗車待ちの行列を見て絶望すると良いよ！**<sup>\*30</sup>

・CD や DVD を購入する際は、ケースがつぶれないようにクリアケースなどを用意しておくと良いでしょう。帰ってみたらケースがバキバキになってた、とか哀しそうです。

・防寒用具 or 日よけ。「極寒の地イバラキで鍛えられた我らがツクバリアンに有明の寒さ程度敵ではない！！」なんて言っていると、**いろんなフラグが乱立しますので、カイロや手袋を用意しましょう**。特に冬コミで朝から並ぶ人は寒さが大変らしいので、注意が必要ですね。夏は夏で、今度は炎天下の直射日光に晒されます<sup>\*31</sup>。

・その他、持ち込み禁制品等についてはカタログに注意書きがしっかりとあるので、そちらを参照してください。アドバイスも色々載っています。

### ●現地での行動

・常識を忘れなければ特に問題になるような事はないはずです。あと、スタッフの誘導には必ず従いましょう。トイレは超絶並ぶらしいので注意ですね。利用する必要がなかったので見てないですが。

・サークルスペースは非常に細かく配置されていてとても覚えにくいので、地図はすぐに取り出せる形でポケットに入れておきましょう。

・コミケにはいろんな人が参加しますが、みんな普通の人です。変な人<sup>\*32</sup> は一部です。サークルの売り子さんも普通の人なので、変に畏ったり逆に馴れ馴れしかつたりせずに、普通に会話すればいいんじゃないかな、と思います。って当たり前の事なのですが、コミケ古参の人からは、「最近は自分を『客』だと思っている人が増えた」というセリフをよく聞きます。コミケの理念として「コミックマーケットに『お客様』はいない」<sup>\*33</sup> というのがあり、参加する以上は自覚を持って<sup>\*34</sup> 参加しましょう、ということです。なんか偉そうで御免なさい。

以上、あまりきちんとまとめることが出来ませんでしたが、参考にしていただけると幸いです。

\*29 MAX コーヒーとかはやめてあげてね。

\*30 13:30 にはもう帰り支度したのに、バスに乗れたのは列に並んでから 30 分後でした。もちろんこれはかなり早く乗れた方です。15 時くらいからがやばいらしい。

\*31 每年倒れる人が出てるんじやなかつたかな…。

\*32 お前だ、って言わないで。これでも自覚しているつもりなんです。

\*33 <http://www.comiket.co.jp/info-a/WhatIs.html> 「コミックマーケットとは何か？」（日本語版 PDF）

\*34 つまり、カタログをちゃんと読んで、分からぬ事はまず自分で調べて、とその程度の当たり前の事ですね。

## C75 参加顛末記

### ・12月29日

7:30 起床。WORD の先輩には、始発どころか前日既に東京入りして、二日目は一日中壁サークル<sup>\*35</sup>で売り子をする予定、という強まつた人が居るのだけれど、さすがにそこまでするのは辛いので、普通に起きて普通に公共交通機関を使って現地へ向かう。

ところが、大学本部棟でライジングサン<sup>\*36</sup>の方々と「今日は寒いですね」といった他愛もない話をして高速バス<sup>\*37</sup>を待っていたところ、**予定時刻 8:45 より 5 分早く発車する**、という酷いバグが発生。

次のバスは 9:45 発なので、仕方なく学内循環バスを使ってつくばセンターまで行き TX に乗ろうとしたが、幸いなことにまだバスがつくばセンターで停止しており、そこで無事に乗車。なんか出鼻をくじかれる。

(睡眠という)トンネルを抜けるとそこは上野駅でした。

都内はよく混むので、上野駅から東京駅はバスから降りて JR で移動。大学本部棟で売っている高速バス往復券を買うと、東京都圏内の JR 各線が乗り放題なので便利<sup>\*38</sup>。この際に Suica の残額を念のためチェック。

八重洲中央口から改札を出て右へ南下していくと、「臨時」とかかれた立て看板の前に職員っぽい人が居たので、「ビックサイト行きはどこから乗りますか」と尋ねる。乗り場は道路を挟んで反対側らしいので、地下道を通って反対側へ行く。

11 時前ともなるとバスは意外に混んでいない。着くまでの間、暇つぶし用に持った SF<sup>\*39</sup> を読みふける。

11:20、戦地着。バス停に降りると、赤オレンジの服を着たスタッフが誘導してくれるのでホイホイ付いていく。

「ああ、次は行列だ…。」

幅 10m くらいの人の…列(?)が延々と続く面白い光景が。そして更に進むと、幅 12.5m<sup>\*40</sup> の階段

---

\*35 文字通り、会場の壁際に配置されるサークルのこと。大手サークルの代名詞と考えてまず間違いないでしょう。一般参加者(購入者)を並べるスペースや、頒布品を置くサークルスペースが広い、壁にポスターなどを貼れる、と利点は多いのですが、当然人気サークルであることが多いので、壁サークルに当たった参加者はかなりの時間的コストを覚悟する必要があります。もつとも、大手だから自動的に壁、という訳でもないそうですが(制作物の予定頒布数で決定されるらしい)。

\*36 警備員の方々。いつもお疲れ様です。

\*37 前日 28 日に調べたところ、大学中央発の高速バスは 12 月 29 日まで平常ダイヤで運行、とのことだったので、高速バスで東京駅までたどり着き、そこからは都営の臨時直行バス、という計画を立てていました。

\*38 今回はある関係ないですが、都内のどこかに行く用事がある際はとても有用。

\*39 ラノベはすぐ読み終えてしまうのであまり役に立たない。ハード SF ですね、やっぱり。早川とか創元の文庫もっと安くならんかな…。

\*40 Google マップ調べ。

いっぱいに人が詰まっている、もっと楽しくなる光景が。

特に長時間立ち止まる事もなくゆっくり進んで 12 時頃には無事入場。まあ、中に入ってからが本当の地獄<sup>\*41</sup>な訳だけど、ね。

行列の人口密度が異常だったことに比べると、屋内に入った後はそう混んでない感じを受けるが、よく考えると屋外と同じような混み方だったらそもそも身動き取れない<sup>\*42</sup>。

ところで、途中で露天でサンドイッチ売ってる店<sup>\*43</sup> があったのだが、テントなどの覆いがなくても良かったのかな、アレは…。

で、この後は最優先事項の缶バッヂが売り切れてて涙を飲んだり、新刊<sup>\*44</sup> だけでなく既刊<sup>\*45</sup> も手にしてしまったり、ある島で気付いたら 3 冊買ってたり、前日調べたときには「いや買わないだろー」と思っていた DVD に手が伸びたり、何故か 1000 円札がトートバッグに化けてたりと、色々楽しい事<sup>\*46</sup> になったあと、初回としてはこんなもんだろうということで 13:30 に撤収開始。バスに乗るまで 30 分ほど並んだ後、来た道順をちょうど逆に辿って筑波まで帰還。

ところで、屋外ではコスプレなども行われていたが、特に興味もなかったので今回はスルー。まあ、外だけじゃなくて中にも普通に居たが<sup>\*47</sup>。

\*41 並ぶのはヲタクだ！

並ばないのはよく訓練されたヲタクだ！！(といいつつやっぱりどこかで並ぶ羽目になる)

ホント コミケは地獄だぜ！フウハハハハーアー

\*42 超大手サークルだと、販売スペース前で普通に見られる光景だから困る。あと、企業ブースは更に次元が上らしいです。「TYPE-MOON」や「リリカルなのは」とかは文字通り丸一日並んだりするらしい。とてもむりっぽ…。

\*43 ビッグサイト内ではなくて、入り口からずっと迂回して続いている行列の沿線にあった店。

\*44 そのイベントが初出となる領布物のこと。

\*45 過去のイベントで既に領布されたもの。過去の在庫であったり、新たに印刷されたものであったりします。再販とも呼ばれます。

\*46 「西 1」地区にまず突入。「み」の列で記念すべき一人目の第三種接近遭遇。もっとも、知り合いがいるスペースだったので、最初から回る予定でしたけれど。ここで一つ目の購入。挨拶を済ませた後、長居もアレなので南下して後輩が居る「ゆ」の列に移動。それなりに売れているようで、ここでも「お疲れ様」等声を掛けた後、更に移動。

前日から東京入りしている先輩のスペースは壁の「れ」の列で、ちょうど一人で切り盛りしていて、「すまんが飲み物買いに行って貰えたりする？」と頼まれたので、持ち込んだ 500ml ペットをお渡しました。

挨拶しようと思っていた人はもう一人いたのですが、その人も別のサークルへ挨拶回りをしていたそうで、会えずじまい。まあ前もって「行く」と言ってないのに、当日会える事を期待しても仕方ないね。

\*47 西地区は予想通り「東方 Project」の勢力が強くて、「魔理沙」、「霊夢」辺りを筆頭に「八雲一家」、「スカーレット姉妹」、「パチュリー」、「雛」辺りが人気のようでした。「風神録」以降は「諏訪子」と「早苗さん」を見かけたくらいな気が。よく覚えていませんが。

「霖之助」が超似合っている人がいたなあ…。

## C75 参加顛末記

人口密度は意外な事に、実質「東方 Project」エリアになっていた「西」地区北側より、「VOCALOID<sup>\*48</sup>」や「THE IDOLM@STER」など音楽系エリアになっていた「西2」地区南側の方が大変なことになっていた。一部全く動けないエリアとかも。

そういえば、コミケ古参の知り合いから、「あれだけ人居るのに**不思議**と知り合いに遭遇するんだよねえ…」と聞いていたのだけれど、一般参加の人とは誰とも遭遇できず、ちょっと残念。

16:00、宿舎着。かなり早めに会場を出たおかげで、早く帰ってくることが出来た。とりあえず買ったものを並べて写真を撮ったりして遊ぶ。

戦禍戦果報告<sup>\*49</sup>。



ところで、前述の注意ポイントで述べておきながら私自身はトートバッグを持っていないので、カタログをとらのあなたが買ったときについてきたブックケースを何の気なしに持つて行ったのだけれど、これが予想以上に役に立った。常に右脇にケースをスタンバイさせて、「これは！」というものを順次突っ込む<sup>\*50</sup>という簡単なお仕事。ありがとう、とらのあなた、今回の MVP(?) は君だ。

\*48 「鏡音リン」「鏡音レン」の二人組のコスプレは覚えているけれど、「初音ミク」の格好をしている人はそんなに見なかったような気がするので、外にいたのでしょうか。

\*49 「東方 Project」二次創作ばっかり。まあ、一部そうではないのもあります。

\*50 もちろん、お金は払った上で。

そして、ここで驚くべき事実が判明。

- ・なんと一冊は 18 禁。しかもとてもペドい<sup>\*51</sup>。ペドいことは調べたときに把握してたけれど、18 禁だということには気付いてなかった、というか買うとき気付けよ、私。まあ気付こうが気付くまいが買ってただろうけど。
- ・ジャケ買い<sup>\*52</sup> したら、作者が知っている人だったという罫。きっと絵柄で釣られたんだろうけれど、無意識って怖いね。
- ・あらかじめチェックしておいた、「好きなキャラクターの本」より、チェックしてなかつた「好きなカップリング<sup>\*53</sup> の本」の方が多かつた<sup>\*54</sup>。

なお、出発前たっぷり(?)用意していった小銭は、100 円玉一個を残して全て綺麗に消化。前日までに用意した分ではやっぱり足りないだろう、と思って出発直前にコンビニと自販機で 10 枚くらい新しく作っておいて良かった。

そんなこんなで、かかった経費は交通費 2920 円、購入費 9500 円(2 冊ほどあやふやなのでもしかすると± 200 円くらいあるかも)という結果でした<sup>\*55</sup>。

参加された皆様、お疲れ様でした。

\*51 作者が自分のハンドルに一時期に「ペドい」って入れてたくらいですし…。期待は裏切らなかつた。(お

\*52 コミケなどの即売会においては、ほとんどの場合購入前に中身を立ち読みさせてもらえます。なので、いわゆる「ジャケ買い」は発生しにくいのですが、私の場合、表紙だけ見て買うことが多いので、このようなことが起こるわけですね。

\*53 特定のキャラクターとキャラクターの絡みに主眼を置いた本。あるいは、読む上で、特定のキャラクターとキャラクター間に何らかの人間関係(恋愛関係であったり友人関係であったり、敵同士であったり)。まあ一つ目であることが多い)が存在していることを前提条件とした本。一般的に「キャラ名×キャラ名」で呼称されることが多いようです(「×」は「かけ」と読むことが多い)。

元々はやおい用語(男性キャラ×男性キャラ)でしたが、最近では一般的な関係においても普通に使われています。特定のカップリングしか認めない派閥間や、場合によってはどちらが「攻め」か「受け」かで論争になってたりします。

なお、「攻め」は(主として性的な意味で)物理的・精神的に弄る方、「受け」は弄られる方の意味で、たまに「攻めは男性役」と説明されていることがあります、男性だからといっても受けな場合もあるので油断は禁物。

\*54 姉様よりもレイマリ本の方が多いという事実。「西ね」辺りがレイマリ島っぽくて、ついふらふらと気付いたら 3 冊も。いや、本当はもっと買いたかったけれど。

\*55 知り合いに三日間で 170 個/70k 円くらい消費した人が居て、曰く、「これくらい普通」だそうです。

## C75 参加顛末記

で、29日から宿舎は集中暖房が止まる<sup>\*56</sup> という落ちも付いたので、今回の記事はこの辺で<sup>\*57</sup>。

あ、情報特別演習Iの最終レポートは、無事に締切り2分前に提出できました。<sup>\*58</sup>

---

\*56 12/29 - 1/3 の一番寒い時期に止まるとか、馬鹿なの私死ぬの？ しかも、共同浴場も停止というおまけ付。

\*57 2009年夏は2008年と同じように帰省している気がするので、次回があるとすればまた年末ですかねえ…。

\*58 なんだかんだで数十枚も書いたせいで、ぎりぎりで印刷間に合わないか、ととてもヒヤヒヤ。

## ねるねるねるねるねるねる。

文 編集部 いのひろ

### Hello, ねるねるねるね World.

皆さんは「ねるねるねるね」をご存知ですか。情報の皆さんなら主食レベルだと思いますが、ご存じない人の為に軽く説明します\*1。ご存知、無いのですかっ！？

ねるねるねるねは、1985年よりクラシエフーズ（旧カネボウフーズ・ベルフーズ）から発売されている菓子。

砂糖を主成分とした粉に水を加えて「ねるねる」を作り、キャンディチップやチョコクリンチを付けて食べる。

化学実験を思わせる手順によって自ら菓子を「作る」斬新さや、作る過程でねるねるの色が変わる点などが子供の心をつかみ、発売から20年以上経った現在も根強い人気を誇っている。

ということで非常に可笑しいお菓子です。また

着色料などが体に悪いのではないかと考慮して買い与えるのを控える親もいる。

との事ですが、

ねるねるねるねの場合、着色に使われる的是アントシアニン系の色素であり、原材料は赤キヤベツやクチナシである。色の変化も酸性値の変化によるもので、これはリトマス試験紙と同じ原理である。

ということで、残留農薬や合成添加物などは含まれていません。体にとても悪い、という訳ではないのです。

今回はこの「ねるねるねるね」が急に、大量に食べたくなってしまったので、食べてみました。

### 新年あけましておめでとうございます。

時は2009年、帰省を諦めた私は編集部員のmittyと共にスーパー「カスミ」にいました。主な目的は「今日の昼ご飯の確保」だった訳ですが、お菓子コーナーに立ち寄ると、そこには懐かしい「ねるねるねるね」がありました。そこで私はふと思い出したのです。WORDの毎週開催される編集会議で「ねるねるねるねをネタにしたい」と発言していた事を。次の瞬間私は買い物かご

\*1 おしえて！ Wikipedia先生：<http://ja.wikipedia.org/wiki/ねるねるねるね>

## ねるねるねるねるねる。

にありったけの「ねるねるねるね」を放り込みました。



今回は「カスミ」にあったすべての「ねるねるねるね」を買いました。ソーダ味とぶどう味がありました、「そんなの関係ねえ」で全部。「箱でありませんか?」と店員さんに聞いたところ<sup>\*</sup>、在庫を確認してくれて「店頭に出てる物だけです」と教えてくれました。

### 調理

とりあえず作ってみましょう。すべて開封して「1ばんのこな」「2ばんのこな」「3ばんのこな」「スプーン」「容器」などに分けます。



付属の「容器」で作ると、全部作る事ができないので、MAX プリンの時にもお世話になった「片手鍋」を使います。混ぜるのにはしゃもじを使います。

\*11月の2日から何してるんでしょうね、まったく。

ねるねるねるねるねる。

「1 ばんのこな」をすべて入れます。



どう見ても「白色粉末」です。本当にありがとうございました。

次に「水」を入れます。



MAX コーヒーのように見える方もいらっしゃるかと思いますが、私たちの業界では「水」です。

## ねるねるねるねるねる。

練ります。



いい感じにまざって参りました。というかここまで「コシ」が出るとは思っていませんでした。モノクロな WORD の紙面ではこのヤバさ加減がわかつていただけないかもしれません。とりあえず色が「食べ物」では無い感じです。HTML で用いる 16 進のカラーコードでは「#999966」です\*1。

つぎに「2 ばんのこな」を入れます。何故か「一度に全部入れたい」と思ったので、付属した「容器」の一つに全部あけてから、一気に投入します。



\*1 「<body color="#999966">」と書いたファイルを html 形式で保存して、Web ブラウザで開くとどんな色かわかります。

ねるねるねるねるねる。



ちなみに「2 ばんのこな」は、カラーコードだと「#3399cc」に近い色です。練ります。

### 実食

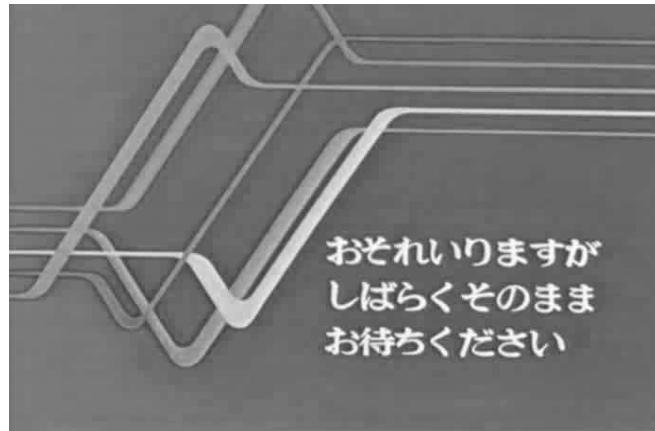
いい感じに練れたところで、「3 ばんのこな」を開けて、食べる準備をします。ラムネたくさん。付属のスプーンでは小さいので、割り箸を用います。



こうやってつけて…

ねるねるねるねるねる。

う、



なんとも言えないこの味。本当になんと言ったらわからない。酸味と甘みと、あと何かよくわからないものが同時に来て、さいごに「ほのかに MAX コーヒー」。「あの MAX コーヒーが負けている！？ 信じられない、負け知らずの MAX コーヒーが負けているだとっ！！！」

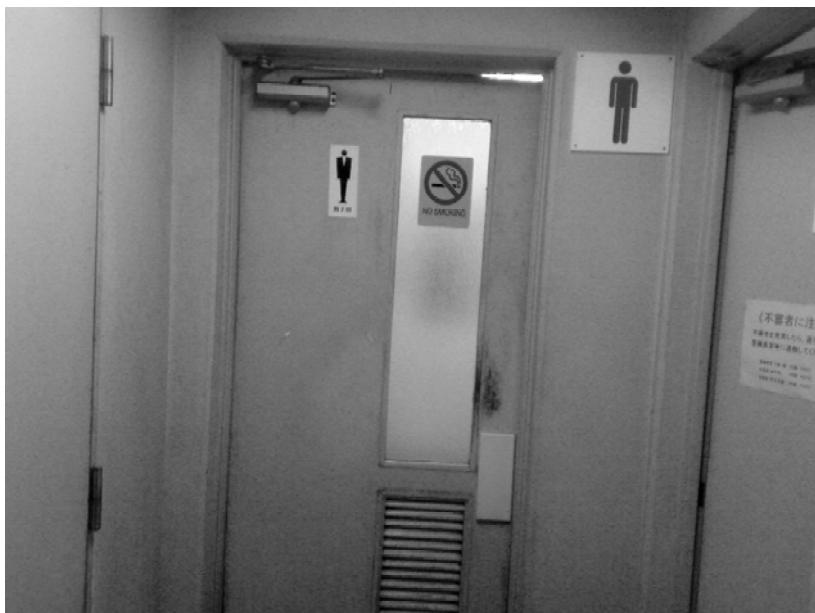


ねるねるねるねるねるねる。

若干の「ざらざら感」が良いのかもしれません、量が多すぎて「ざらざらし過ぎ」でおいしくない。というか食べ物じゃない。カラーコードは「#999999」です。完全に灰色。て言うか食べてるとなんだか胃の中がもわもわしてくるんですよね。何かが発生しているのではないか。



今回は完全に負けました。



大量の「ねるねるねるね」は編集部でおいしくいただきました。



あきたこまちは……へつへつへつへつ



磨けば磨ぐほど、(磨ぎ汁の)色が変わって……



こうやって炊いて



納豆をつけて



うまい！ \*1

\*1 テーレッテレー

炊いて美味しい、あきたこ・ま・ちっ♪



この記事はパロディです。

JA うご、  
神主ZUN、  
クラシェフーズ、  
その他実在、架空の人物、  
団体とは関係ありません。



# お知らせ

文 編集部 IX

## ある日の筑波大生

ふう……今日も疲れた……

何か……何か 「頑張った自分へのご褒美 (笑)」 的なものはないのか……



ありましたよ……

160円で幸せになれる方法が。

冷めていてもおいしいですが……

**「焼きたての方が断然おいしい！！」**

そう思いませんか？

チリドッグ（160円）

なに？ 「焼きたてなんて、そろそろ買えないよぉ (泣)」 ですって？

そんなあなたに……



テケテケン♪  
オ～ブント～スタ～♪

そう、買って来たチリドッグを5分間オーブントースターで加熱するのです。



なんということでしょう。

**匠の技**によって、

**口内を傷つけかねなかつた**

パンの硬さは程よいものとなり、

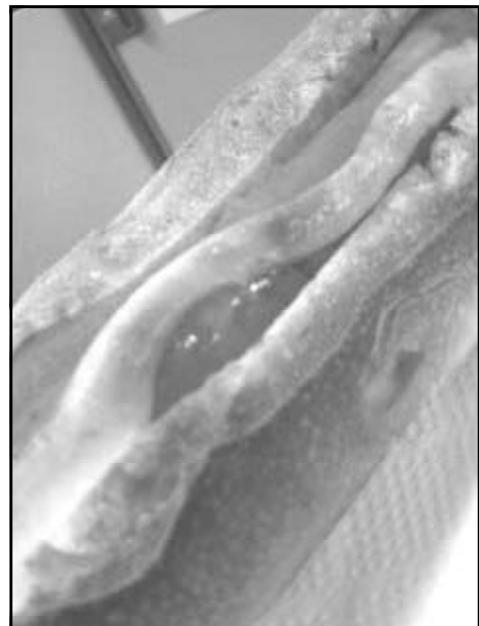
依頼者に**至福の時**をもたらします。

焼きあがってから時間がたち、

**本来のジューシーさを**

発揮できていなかつたソーセージも、

**肉汁**(水)も滴るいい女(男?)になりました。



オープントースターの可能性は**無限大**……とはいきませんが、

生活の中で**いい仕事**をしてくれている、と私は思います。

**IX のパフェクトオープントースター教室**、coming soon ……

## coins-admin メンバ募集

# coins-admin メンバ募集！

文 編集部 yasuharu

### coins-admin とは

coins-admin とは、情報科学類計算機の学生管理者のことです。情報科学類・情報学類生の学生と大学院生で構成されています。

coins-admin では、情報科学類計算機運用委員会のもと、教員や技術職員の方々と協力しながら情報科学類計算機で使用する計算機を利用しやすいようにする活動を行っています。

### coins-admin の仕事

日常的な業務として、利用者からの質問に答えることや、coins のシステムの障害対応などを行っています。また、計算機をより便利に利用できるようにするために、新しいサービスの提案などを行うこともあります。

具体的には、以下のような仕事を行っています。

- ・計算機の障害対応
- ・VPN サービス
- ・便利なフリーソフトウェアのインストール
- ・便利なソフトウェアの開発
- ・卒研配属支援 CMS の設置、管理
- ・学類内情報流通
- ・印刷用紙の監視と補給
- ・情報科学類「教育用計算機システム使用の手引き」執筆

また、open-coins という有志で提供しているサービスがあります。open-coins では、情報科学類の余った機材を利用して、メーリングリストや IRC など学類では提供していないサービスを行っています。

open-coins について詳しい情報は、<http://www.open.coins.tsukuba.ac.jp/>をご覧下さい。

### メンバ募集

coins-admin では、常に新しいメンバを募集しています。技能・経歴などは問いません。coins-admin はその活動を通して、計算機の管理や環境整備に関する知識を得る場でもあります。

紹介を読んで興味をもたれた方は、[coins-admin@coins.tsukuba.ac.jp](mailto:coins-admin@coins.tsukuba.ac.jp) にメールをしてください。もしくは、WORD メンバには coins-admin メンバが多くいるので、興味がある方は WORD まで来てください。coins-admin になる方法など、詳しい内容を説明いたします。

# WORD

## 30th Anniversary

Coming Soon...

文 編集部 いのひろ



## WORDの新しいWebサイト、 「WORD Press」の登場です。

\*画面はイメージではありません。

WORD Press は、過去の WORD から選りすぐった記事や、編集部員の独り言、記事執筆における裏話など紙面には収録しきれないアレグな内容まで、さらに WORD を楽しんでいただく為の Web サイトです！！！

**<http://www.word-ac.net/>**

是非ブックマーク、RSS リーダに登録してこれまで以上に WORD の記事をお楽しみください！

# 情報科学類誌

# WORD

## JA ウホッ！いいお※号

発行者

情報科学類長

編集長

柴田 泰晴

製作・編集

筑波大学情報学群  
情報科学類WORD編集部  
(第三エリアC棟212室)

印刷

情報科学類印刷室

2009年1月

初版第一刷発行