君は知っているか……。
ひっくり返しても、ひっくり返さなくても、
WM はWMなのだということを。

ふるつき

2018 年 1 月 21 日

1　それってマジクールじゃね？

　ようお前ら、ロックしてるか？　俺はいつでもロックだ。なぜなら俺が、俺こそがロックだからだ。今日はちょっとお前らに聞きたいんだが、マジでクールなお前らはどんな OS を使ってるんだ？　……オーケー、わかった。聞き方が悪かったんだろ？　で、なんて名前のディストリビューションなんだ？　……。……まあいい、俺は Ubuntu を使ってるんだぜ？　マエストロは「Ubuntu は中学生の OS」って言うけど、そんなに悪いもんでもないって俺は思うぜ。

　ところでこの Ubuntu なんだが、少し前までは「Unity」っていう WM がメインを張ってたんだ。その前は「Gnome2」ってやつだったんだが、Unity ですっげぇ格好良くなってよ。モーションの一つ一つに俺の心が踊ったね。でもその代わり、すっげぇ重いんだよ。ガチでやってらんないよな。だから最近は「Gnome3」がメインなんだよな。俺もめっちゃ使ってる。文句なしだね。

　お前らの OS ではどんな WM が動いてるんだ？　え？　WM って何かって、そりゃあ……**Window Manager** じゃんかよ。お前の PC つけたら広がってる世界のことだよ。俺の周りではやっぱり Awesome が人気だよな。あれはすっげぇいいタイリングマネージャだと思うよ。俺も一時期使ってたんだが、Chrome のタブと相性が悪くてやめちまった。あとは Xfce なんて鉄板だな。軽いのに痒いところにも手が届く力強さがあるって思うね。KDE、Cinnamon、Mate なんかはちょっと俺には違いがわからねぇが、どれもこれもめっちゃ良く出来てるよな。他の WM については悪いな、全然わからねぇ。

えっ、まじかよ twm 使ってるのか！？ twm っていやぁ Linux でもデフォルトの WM じゃねぇか。さすがだぜ。あのシンプルを極めて極まってる感じがすげぇよな。ハッカーって感じだぜ。まあ情報工学科でも FVWM を使ったり LXDE をつかったりするんだけどな。

まあ、何が言いたいかって、WM っていうのはお前らが OS を起動してログインして、一番最初に出会う相手ってことだよ。WM がユーザインタラクションの 9 割を支配してるってことだ。WM にはこだわっていきたいよな。

どうやってこだわるか、わかるか。一つには、設定ファイルをめっちゃ書くってのがあるよな。俺は Super+Return でターミナルが起動するようにしてるけど、これはまじでクールだ。よく使うアプリケーションは一瞬で立ち上がって欲しいし、ウィンドウやワークスペースの切り替えをクールにこなしたいよな。そのために俺の最強のコンフィグを作るんだ。これは燃えるね。

でももっと燃えることもある。それは **WM を作っちまう** ってことだ。これには心が震えるね。俺らみたいな人間が作りたいものといやぁ、OS、プログラミング言語、CPU の **OPC** って相場が決まってるが、WM だって作りたいね。そう、WM を作る。これってマジクールじゃね？

2　お前は WM がどういう奴か、知ってるか

WM がなにをやってるのか、それって難しい問いだと思うぜ。だってよ Window Manager なんて抽象的すぎるんだ。Window を Management するなんて、何も言ってないのと一緒だぜ。お前は答えられるか？ WM って何だ？

俺もわからなかったから、調べたんだ。そう、わからなければ調べりゃいいんだ。かんたんなことだろ？　それで答えが手に入るとは限らないけどな。でも今回は、それなりにわかったぜ。答えは一つじゃないってことも。

例えば WM は「Window を終了」するためのインターフェースを提供するんだ。お前の WM では、Window の右上か左上にアプリケーションを終了するボタンが表示されてないか？　あるいは、Alt+F4 とか、Super+q とか、とにかく「この方法で Window を閉じれるぜ」って方法が。その仕事をやってくれているのは WM だったんだ。もちろん他にもあるぜ。「Window を動かす」「フォーカスを変更する」のも WM の仕事なんだ。WM がない世界では、Window は表示された場所から動けないか、すべてのアプリケーションが、自分自身を動かすための処理を内部に持っていなくちゃダメになっちまう。それって興味がないことにリソースを割くってことだろ、全然クールじゃないな。スタック

型の、つまり「Window が重なって表示される」WM で、後ろの Window をクリックしたり、マウスを乗っけたりしたらその Window が手前に出てくる、これも WM が仕事してくれてるからなんだぜ。

それだけじゃない、お前は「仮想デスクトップ」って使ってるか？　あれすっげぇ便利だよな。あれは全部 WM がやってくれてることなんだぜ。WM がなけりゃお前のディスプレイはターミナルやブラウザやエディタで埋まっちまうってわけだ。

今まで当然のように使ってた機能だけど、それは WM が支えてくれていたんだってことがわかるよな。もちろんそれを知らなくて「俺はそれがどうやって動いてるのかわからねぇ。でも俺はそれを知る必要はないんだ。それってすっげぇクールなことだろ？」っていうのもマジかっけぇと思うよ。でもそれと同じかそれ以上に「俺はまじでこの WM に感謝してるんだ。だって俺が Window を動かせて、デスクトップを切り替えられるのは WM のおかげだからな」って言ってるやつもクールだね。

3　じゃあ早速 WM 作ろうぜ。こんなにクールなことがあるか？

まあ待て、そう焦るんじゃねぇ。気持ちが燃えてるのはすっげぇいいことだと思うよ。だけど「急いては事をし損じる」んだぜ。まずやるべきことがあるってこった。

俺たちは「WM が何をするか」はだいたいわかった。でも「WM がどうやってそれをやってるか」、その How がまだわかってないんじゃねぇのか？ まずはざっとでいいから概念を習得することは重要だ。伝説のギタリストだって生まれたときから F のコードを奏でられたわけじゃないってことだ。

俺の調べてきたところだと、WM でもなんでもとにかくグラフィカルなものはみんな **X Window System** ってやつの一部で、 **X Server** ってやつの上で動いてるんだ。この X Window System は今日まで Linux や Unix の Window をすべてまとめあげてきたサイコーのプロトコルなんだ。X Window System はサーバクライアント型のアーキテクチャを採用していて、X Server のようなサーバはクライアントの要求を受け付けて、Window を描画してくれるマジですげぇ奴なんだよ。WM はクライアント側で、他のすべてのクライアントを下に置く、すっげぇ親分って感じなんだな。

それでこの X Window System に従ってアプリケーションを作るなら、それ用にすっげぇライブラリが用意してあるんだ。**XLib** とか **XCB** ってのがそうなんだよな。Xlib は古くから使われてる、唯一無二のスタンダードってやつだった。これはサイコーだと

俺も思ったんだが、XCB ってのが出てきたんだよ。XCB は Xlib を置き換えるライブラリで、Xlib よりも使いやすく、小さく、速くなってるって言うんだからまじで GOD だよな。なんでも XCB のほうがより、X Window System のプロトコルそのものに近いとか、XCB は非同期性が高いとか、マジでイケてるらしい。

だから俺は XCB を使うことにしたよ。ロックの心は古いものも大事にするけど、新しいことに挑戦したい気持ちを止めることはできねぇんだ。おっと待て、Wayland のことは言わないでくれ、あれは俺にはよくわからなかったんだ。恥ずかしい話だけどそういうこともあるんだよ。

4　俺たちはどんな WM を作りたいんだ？

目標を定めるって大事なことだよな。俺は飽きるまでやればいいと思うんだけど、せっかちなお前らのために、この記事でどこまで作るかを書いておいてやるよ。それで、「ふーんこの程度か……」と思って読むのをやめても構わない。ハートが乗ってないときには何をやっても身につかねぇからな。

それで今回は「Window を表示」できて「動かせ」て「フォーカスを変更」できて「終了」できるところまでを作るぜ。それより先はもうお前らが自分でやらなきゃならねぇってことだ。

ついでだ、このセクションで「どの言語でつくるか」っつー、プログラマ的には一番重要なところも決めていくぜ。今回使うのは……そう、**D 言語** だ。X のプログラミングをするなら C 言語が基本ってのは常識だが、D 言語は C 言語に対しても高い互換性を持ってる。それにすっげぇ書きやすいし……なにより、**俺が愛してる**ってことが重要だ。俺の熱い spirit が、D 言語を選べって言ってるのさ。開発環境は Ubuntu だ。俺は最近出たばかりの *Bionic Beaver* こと 18.04 を使うが、*Xenial* とかでもきっと動作は変わらねぇ。

5　まずは慣れていくところから始める。基本のキだぜ

D 言語の開発準備はととのったか？ **DMD**[*1] を落としてくるだけで、D 言語が使えるようになる。最近はまじでイケてるインストールスクリプトがあって、もうそれだけでお前は dmd コマンドが使えるようになる。更にパッケージマネージャの **DUB**[*2] もあったほうがいい。こういうツールを使いこなして、本当に重要な事をやるってのが、クールっ

[*1] https://dlang.org/download.html
[*2] https://code.dlang.org/download

てもんだ。これはインストールスクリプトでインストールしたときは一緒に入ってくるけど、 apt−get install −y dub でも手に入るから、そっちを使ってもいいぜ。

そこまでできたら、今度はプロジェクト用のディレクトリを作らなくちゃな。 mkdir くらいはできるよな？ 俺は joken をマジリスペクトしてるから、今回はプロジェクト名を **jokenwm** にしちまおう。そういうわけで、 mkdir jokenwm を実行するんだ。ちなみに俺が個人的に作ってる WM は **wmderful**[*3] って言うんだぜ。マジでイカした名前だろ？ 俺はこの名前を思いついちまったから WM を作り始めたと言っても過言じゃないね。脚注に示した URL のコミットが、今回作るところまでだから、参考にしてもいいぜ。

おっと、話が逸れちまった。それで、 jokenwm に cd したら、 dub init だ。これは「D 言語のプロジェクトを作るから、必要なもんを準備してくれよな」って意味なんだよな。そしたら対話が始まって色々聞かれるけど、基本的にデフォルトの選択肢を選んでいればいい。つまり俺たちは Return を押すだけの猿になれるってことだ。だけど最後の Add dependency の項目に xcb−d を入れるのだけは忘れないでくれよ。これがなくちゃ XCB でプログラミングできないんだ。

```
/tmp/jokenwm> dub init
                                                    14:23:14
Package recipe format (sdl/json) [json]:
Name [jokenwm]:
Description [A minimal D application.]:
Author name [theoldmoon0602]:
License [proprietary]:
Copyright string [Copyright (c) 2018, theoldmoon0602]:
Add dependency (leave empty to skip) []: xcb-d
Added dependency xcb-d ~>2.1.0
Add dependency (leave empty to skip) []:
Successfully created an empty project in '/tmp/jokenwm'.
Package successfully created in .
/tmp/jokenwm>
```

[*3] https://github.com/theoldmoon0602/wmderful/tree/b4d69371449c66575ef998bf328d1ab65adf9644

```
14:23:28
```

さて、これで大体の準備が終わったってわけだぜ。この状態で dub build すればバイナリ、ここでは ELF が生成できるし、 dub run すればバイナリを実行できる。バイナリの名前は jokenwm になっていると思うから ./jokenwm としてもいい。ソースコードは src/ にあって、エントリポイントは src/app.d の main って関数だ。まあとりあえずは dub run してみることをおすすめするね。ここで、例えば Error: linker exited with status 1 ってエラーが出て、エラーメッセージの中に /usr/bin/ld: cannot find −lxcb を見かけたら、 すかさず apt−get install −y libx11−xcb−dev だ。必要なライブラリが手に入る。

そしたら早速、XCB でのプログラミングを始めよう。XCB でプログラミングするときはまず、 X Server に接続することが必要だ。X Server と通信をするアプリケーションを書くんだからな。そのためにはずばり、xcb_connect 関数を使う。どうだ、この完璧な名前は？　最高だろ？　そして、接続は失敗することもあるからエラーチェックもしないといけない。そしてその後は、スクリーンとルートウィンドウの取得だ。これが何かは今は知らなくても構わない。あとで散々お世話になるってことだけわかってればな。まずはここからだ。かんたんだから、 *src/app.d* をそのまま見せてやる。

```
import xcb.xcb;

import std.experimental.logger;

void main()
{
  auto conn = xcb_connect(null, null);
  if (conn is null || xcb_connection_has_error(conn) != 0) {
    fatal("failed to connect X Server");
  }
  auto screen = xcb_setup_roots_iterator(xcb_get_setup(conn))
      .data;
  auto root = screen.root;
}
```

D 言語はよくできた言語だから、自明な型は **auto** で済ませてもいい。便利だろ？　も

ちろん xcb_connectino_t* とか xcb_screen_t* とか xcb_window_t とか書いてもいいが、面倒だろ？

6 お楽しみの時間だ！　本格的に Programming していこうぜ!

さて、お次は、フックするイベントの登録だ。WM は X Server への要求をインターセプトして Window の管理を行う。その要求のうち、ほしいものはちゃんと登録しなくちゃならねえってことだ。一気にややこしくなった気がしちまうが、怖がることはねぇ。

```
uint ROOT_EVENT_MASK = XCB_EVENT_MASK_SUBSTRUCTURE_REDIRECT|
  XCB_EVENT_MASK_SUBSTRUCTURE_NOTIFY|
  XCB_EVENT_MASK_STRUCTURE_NOTIFY|
  XCB_EVENT_MASK_ENTER_WINDOW|
  XCB_EVENT_MASK_POINTER_MOTION|
  XCB_EVENT_MASK_POINTER_MOTION_HINT|
  XCB_EVENT_MASK_KEY_PRESS|
  XCB_EVENT_MASK_BUTTON_PRESS|
  XCB_EVENT_MASK_BUTTON_RELEASE;

auto cookie = xcb_change_window_attributes_checked(conn, root
    , XCB_CW_EVENT_MASK, &ROOT_EVENT_MASK);
if (xcb_request_check(conn, cookie) !is null) {
  fatal("another window manager is already running");
}
xcb_flush(conn);
```

どんなイベントを受け取るかは xcb_change_window_attributes_checked 関数で指定する[*4]。第三引数が、そういう要求をしてて、第四引数に渡してる値が、「どんなイベントを」を指定してるってわけだ。そしてこの中の「SUBSTRUCTURE_REDIRECT」っつー値を指定できるのはたった一つのアプリケーションだけで、そいつが WM というわ

[*4] checked がつかないものもある。違いについては次を参照
https://wiki.mma.club.uec.ac.jp/clear/wm_devel/2012-12-23

けなんだな。

ところで、ここまで作ってきて dub run とかしたときに another window manager is already running ってメッセージと一緒にアプリケーションが死んじまうって場合があると思うんだが、むしろ出ないときはお前の環境すげえなって思うが、こういう場合には **Xephyr** が本当にオススメだ。Xephyr を使うと仮想の X Server をもう一個立ち上げることができるんだ。 apt−get install −y xserver−xephyr でインストールできて、使うときは Xephyr :1 だ。これで jokenwm を立ち上げるときに DISPLAY=:1 ./jokenwm とか DISPLAY=:1 dub run とかすれば、そっちには WM はないから怒られなくて開発がスムーズってモンよ。

さて、ここまでできたら、骨格のほとんどが完成したと言っていい。あとは、「イベントループ」を書くだけだ。イベントループというのはかんたんで、「ずっとループを回し、イベントを受け取ったらそれに合わせて処理を行う」ことだ。最小のイベントループだと、こんな感じになるな。

```
while (true) {
  auto event = xcb_wait_for_event(conn);
  switch (event.response_type) {
    case XCB_MAP_REQUEST:
      auto e = cast(xcb_map_request_event_t*)event;
      // 何かする
      break;
    default:
      break;
  }
}
```

これは XCB_MAP_REQUEST だけを処理するループだけど、他のイベントも同様に処理できるってのはわかるだろ？　おいおい、もうほとんどできちまったぜ。

そういえば、D 言語では構造体のポインタから要素を参照するときに −> 演算子じゃなくて . 演算子なんだって知ってたか。アロー演算子を書かなくて済むのは楽だよな。

7　え、そろそろ Window 出せるんじゃね？　イヤッホウ！

お前らに朗報だ。ここから先はもう、各種イベントを処理していくだけなんだぜ。もちろんそれが難しいといえばそうなんだが。まあ気持ちが上がってくるだろ。いいことだ。

あるアプリケーションが Window を描画したいと思うとする。するとそいつは xcb_create_window 関数で Window を作る。この状態ではまだ Window は作られただけで描画はされていない。そこで xcb_map_window 関数を使って X Server に Window の表示を要求するのさ。このときに飛んでるイベントが XCB_MAP_REQUEST ってわけ。だが、ここに割って入るのが WM だ。WM は X Server に向けて要求された XCB_MAP_REQUEST をインターセプトしてくる。これができるのは WM だけだし、このイベントを受け取るためのマスクが、XCB_EVEMT_MASK_SUBSTRUCTURE_REDIRECT だったんだよな。

それで、WM はこれをインターセプトしてきて、Window が作られようとしていることを知り、自分の管理下に置くわけ。ほとんどの WM ではここで「フレーム」っつー Window を作って、これを要求してきた Window の親 Window にして、タイトルバーとかボタンとかをフレームに描画するわけだが、それは面倒なので jokenwm ではやらない。jokenwm でやることは作られた Window をいつでも操作できるように持っておくことだけだ。

これらの処理を終えたあと、WM から X Server に向けて xcb_map_window 関数を実行すると、晴れてその Window は描画されるってわけだ。大体わかっただろ？　この処理を書いていくぜ。

```
case XCB_MAP_REQUEST:
        auto e = cast(xcb_map_request_event_t*)event;
        xcb_map_window(conn, e.window);
        xcb_flush(conn);
                auto new_client = new Client(e.window);
                clients[e.window] = new_client;
        break;
```

変数eは色々なメンバを持っているが、そのうちの window はそのまま、表示を要求している Window の id なんだ。それを xcb_map_requstに渡す。かんたんだろ？ xcb_flush は忘れやすいから注意だな。これを忘れて「なんでバグってんだっけ？」ってなることも

多い。

clients とか Clientっていうのは、ま、こんな定義だ。

```
class Client {
public:
  xcb_window_t window;
  this(xcb_window_t window) {
    this.window = window;
  }
}
Client[xcb_window_t] clients;
```

Clientクラスはそのまま、描画されてる Window のことだ。いまは window だけしかメンバに持っていないから無用に見えるかも知れないが、今後フレームを実装するとか、そういうときのためにクラスにしておいた。

そして clients は、xcb_window_t をキーに持つ連想配列だ。連想配列にしておくと便利なんだよな。xcb_window_t は uint のエイリアスだからコピーしても重たくならないんだぜ。

さて、これで jokenwm を起動して（Xephyr を起動した状態で DISPLAY=:1 dub run でいいぜ)、イベントループに入ったことがわかったら、別のターミナルから DISPLAY=:1 gnome−terminal をしてみてくれ。別に gnome−terminal にこだわることはないけどな。するとどうだ！？　Xephyr に確かに gnome−terminal の Window が出てきだろ？　イヤッホウ！　いい感じだぜぇ。

8　Window を動かしたいだろ？　そうだろ？

さあ、ここまで来たらあとはほとんど説明なんていらねぇよな。コードがすべてを語ってくれるぜ。だけどそれじゃあ不親切だから、少しは解説させてくれ。何しろ不親切ってのは全然クールじゃないからな。

次は Window を動かしてぇ。こういうときは xcb_configure_window 関数を使う。これも xcb_change_window_attributes 関数みたいなやつで、色々設定できるんだが、その中の一つに、Window の座標もあるってわけだ。

これをいつ行うかが問題だが、もちろんお前らはマウスで操作したいよな？　そういうときは　XCB_BUTTON_PRESS、　XCB_MOTION_NOTIFY、

XCB_BUTTON_RELEASEの各種イベントの出番だ。それぞれ、「マウスのボタン押し込み」「マウスの移動」「マウスのボタン押し終わり」のイベントになってる。完璧だろ？

だが、問題もあるんだな。現状ではこれらのイベントを受け取ることはできねぇ。普通はできるんだが、肝心のとき、「マウスカーソルが Window の上にあるとき」にこれらのイベントは、Window 側に吸われちまうんだ。

そいつは困る。だからちゃんと、そういうときにもイベントを通知してくれるようにする関数があるんだぜ。それが、 xcb_grab_button だ。これは Window に対して適用する関数で、その Window に対してボタンが押されたときに、イベントの通知を WM に投げてくれるってわけよ。引数によって、「WM にだけ投げる」「WM にも投げるけど Window にも投げる」とか、「イベントを投げたら止まる」「止まらない」を選べるスグレモノなんだぜ。

だから先にこっちを書いちまう。これは、 xcb_map_window の手前に書くといいぜ。

```
xcb_grab_button(conn, 1, e.window,
    XCB_EVENT_MASK_BUTTON_PRESS|
    XCB_EVENT_MASK_BUTTON_RELEASE|
    XCB_EVENT_MASK_POINTER_MOTION,
    XCB_GRAB_MODE_ASYNC,
    XCB_GRAB_MODE_ASYNC,
    e.window, XCB_NONE,
    XCB_BUTTON_INDEX_1,   // LEFT BUTTON
    XCB_MOD_MASK_1|XCB_MOD_MASK_2);   // Alt
```

これはちょっとマジでややこしいし勘弁してほしいよな。でもまあ、そんなに気にすることはねえよ。こういうもんだと思っててていいし、気になるなら調べりゃ済むことだ。ただ、XCB についての情報はインターネットの海広しと言えどもまるで転がってないけどな。

とにかくこれを書いておけばアンタイってわけよ。そしたらやっと XCB_EVENT_MASK_BUTTON_PRESS あたりを処理できるな。こっちはコードを見たほうが早いだろうから、コードで示しちまうか。

```
case XCB_BUTTON_PRESS:
  auto e = cast(xcb_button_press_event_t*)event;
```

```
    if (e.detail == XCB_BUTTON_INDEX_1 &&
        (e.state & XCB_MOD_MASK_1) && focusing !is null) {
      auto geometry_c = xcb_get_geometry(conn, focusing.window)
          ;
      auto geometry_r = xcb_get_geometry_reply(conn, geometry_c
          , null);
      if (geometry_r is null) {
        break;
      }
      is_moving = true;
      oldx = geometry_r.x - e.root_x;
      oldy = geometry_r.y - e.root_y;
    }
    break;
  case XCB_MOTION_NOTIFY:
    auto e = cast(xcb_motion_notify_event_t*)event;
    if (is_moving && focusing !is null) {
      uint[] values = [
        oldx+e.root_x,
        oldy+e.root_y,
      ];
      xcb_configure_window(conn, focusing.window,
          cast(ushort)(XCB_CONFIG_WINDOW_X|XCB_CONFIG_WINDOW_Y)
              ,
          values.ptr);
      xcb_flush(conn);
    }
    break;
  case XCB_BUTTON_RELEASE:
    is_moving = false;
    break;
```

ちょっと長いけど頑張って読んでくれ。こっちでも解説するからよ。まあ上から行くんだ

が、まず detailだが、これには「マウスのどのボタンが押されたか」という情報が格納されている。XCB_BUTTON_INDEX_1は左クリックだ。 state には「マウスのボタンが押し込まれたとき一緒に押されていた制御キー」が格納されていて、
XCB_MOD_MASK_1 は Alt キーのことだ[*5]。focusingは今フォーカスしている Client のことで、実は XCB_MAP_REQUESTの処理にちょっと足してある。こんな風にしたってわけ。

```
xcb_map_window(conn, e.window);
xcb_flush(conn);

auto new_client = new Client(e.window);
clients[e.window] = new_client;
focusing = new_client;
break;
```

このfocusingがなけりゃ、WM はどの Window を動かせばいいのかがわかんなくなっちまうからな。
そいで、xcb_get_geometryだが、そのまま Window の座標を取得する関数だ。XCB は非同期だから xcb_get_geometry自体は cookie と呼ばれる値を返してきていて、本当に欲しい値は xcb_get_geometry_reply で取得する。
この Window の座標から、マウスの座標、これが (root_x, root_y)だが、これを減じたものを (old_x, old_y)にしてある。なんでこうなってるかは、XCB_MOTION_NOTIFY の処理を見たらわかるな？　正直言ってここその次の XCB_BUTTON_RELEASE についてはもうなんにも説明することがない。せいぜいvaluesのptrは D 言語の配列である valuesを C 言語の配列として扱うための処理だってことくらいかな。
まじでここまでプログラムを書いて、実行して、Window の上にカーソルを持ってきて、Alt キーを押しながらドラッグしたら Window が動いてすっげぇ感激するから見てくれ。これがパッションってやつなんだ……！

[*5] ちなみに XCB_MOD_MASK_2は Numlock を表し、これがないとうまく行かないが、なぜなのかはわからなかった

9　ここまできたらフォーカスなんてマジ楽勝だよな

次は、「二つ Window を出したときのフォーカス」だ。今はfocusingは Window を作ったときにしか変更されない。これじゃあ動かせるのは最後に作られた Window だけになっちまう。だからフォーカスを変更したい。こういう自然な流れをマジリスペクトしていきたいぜ。

今回の jokenwm では、フォーカスが移るタイミングを「Window 領域にマウスが乗ったタイミング」としたい。なぜならこれが楽だからだ。最初は楽なところから小さく始める。これが肝心だってことは俺にはもう完全にわかってるね。そして、X Server が送ってくるイベントの中には XCB_ENTER_NOTIFY っつー、このために存在するようなイベントがある。

フォーカスが移ったら、その Window が最前面に出てきて欲しい。これはまた、xcb_configure_window で実現できる。こんな感じだ。

```
case XCB_ENTER_NOTIFY:
  auto e = cast(xcb_enter_notify_event_t*)event;
  if (auto client = e.event in clients) {
    uint[] values = [XCB_STACK_MODE_ABOVE];
    xcb_configure_window(conn, client.window,
        XCB_CONFIG_WINDOW_STACK_MODE, values.ptr);
    xcb_flush(conn);
    focusing = *client;
  }
  break;
```

たったこれだけで、フォーカスの切り替えが実装できる。おっと、見慣れない if 文かも知れないが、これは最高にクールで、「clients に eventをキーにする要素があればそれのポインタを client として、if 文の内部を実行」してくれるんだぜ。

だが残念なことがある。このままでは XCB_ENTER_NOTIFYイベントが飛んでこない。このイベントは Window 側で受取の設定をしてやらないとダメなんだ。なんでそんな設計なのかは俺にはちょっとわからねえが、とにかくそうなってるんだ。Window 側で受取の設定をするなら、お前ならどこにそのコードを書く？　そうだ、xcb_map_window の手前あたりに書きたくなるよな。

```
uint window_event = XCB_EVENT_MASK_STRUCTURE_NOTIFY|
  XCB_EVENT_MASK_ENTER_WINDOW;
xcb_change_window_attributes(conn, e.window,
    XCB_CW_EVENT_MASK, &window_event);
```

まあこれは……説明しなくてもいいだろ。これでフォーカスの変更までできたってわけだ。もうほとんどできたも同然じゃねぇか。

10 Window を終了させる……。クールな響きだ……

もうあとはやるだけだって思うだろ？　Window を終了させるだけなら実はそうなんだ。xcb_destroy_window を呼ぶだけでいい。だが、一体いつそいつを呼び出すって言うんだ？　やっぱりここは、キーバインドが欲しくならねぇか？　俺はほしいね。Alt+q とかで終了できたらもう、最高じゃねぇか。ロックだよ。
キーが押された通知を受け取るには……そう、わかってきたな。そういうイベントが確かにある。XCB_KEY_PRESSと XCB_KEY_RELEASEがそうだ。だが、こいつもやっぱり「Window の上では」動作しない。xcb_grab_keyしてやる必要があるというわけだ。
更に問題が出てくるのが辛いところだ。これは D 言語というか xcb−d特有の問題なんだが、xcb_grab_key はもちろん、どんなキーを grab したいかを指定する。これは xcb_keycode_t 型で、まあ実質 ubyte なんだが、普通 XK_Return や XK_q のような形で定義されている。だが、xcb−d ではこの定義が行われてねぇってこった。
幸いにもこいつらは定数だから、適当に調べてきて自分で定義してやりゃあいいんだが……すっきりとは行かねぇなあ。クールじゃねぇ。
なんて言ってても仕方ねぇ。実装するぞ。

```
xcb_grab_key(conn, 0, e.window,
    XCB_MOD_MASK_1|XCB_MOD_MASK_2,  // Alt
    cast(xcb_keycode_t)24,  //q
    XCB_GRAB_MODE_ASYNC,
    XCB_GRAB_MODE_ASYNC);
```

このコードをどこに書くべきか、何をしているかはもうわかるよな？　お前もどんどんロックになってるってことだぜ。もうガンガン容赦なく行くからよ。

```
case XCB_KEY_PRESS:
  auto e = cast(xcb_key_press_event_t*)event;
  if (e.detail == 24 &&
      (e.state & XCB_MOD_MASK_1) &&
      focusing !is null) {
    xcb_destroy_window(conn, focusing.window);
    xcb_flush(conn);
    focusing = null;
  }
  break;
```

つってもこれだけだ。これで当初の目標は達成したし、jokenwm は一応の完成を見たってわけだ。ここから先は、お前らがもっとクールな WM を作っていってくれよな。

11 ふるつき

ふるつきです。5 年生です（ほんまか）。今まで 5 年生と言えば引退していて OB としてこの JokenOffline の記事を執筆していたわけですが、我々は joken に籍を残しているので現役です。
まあそんなことはどうでもよくて、唐突に WM を作りたくなったので WM の記事を書きました。jokenwm についても記事を書きながらコードも書いたので、結局 github においておくことにします（ https://github.com/theoldmoon0602/jokenwm ）。
この記事の執筆にあたって、というかその前の WM の作成にあたって、いろいろなサイト、コードを参照しましたので紹介しておきます。

1. https://wiki.mma.club.uec.ac.jp/clear/wm_devel ほぼ唯一の日本語情報です
2. https://seasonofcode.com/posts/how-x-window-managers-work-and-how-to-write-one-part-i.html 内容は読まなくてもいい感じですが、Part III の図が参考になります
3. https://tronche.com/gui/x/xlib/events/processing-overview.html Mask と飛んでくるイベントの関係がまとめられています。
4. https://xcb.freedesktop.org/manual/index.html XCB の API 一覧が並んでいるので参考になります。特に構造体のメンバを見に行くことが多いです。
5. https://github.com/venam/2bwm/ XCB で書かれた小さな WM です。

6. https://github.com/awesomeWM/awesome awesome も XCB で書かれています。めちゃくちゃ読みやすいです
7. https://github.com/Eelis/i3 i3 もメジャーなタイリングマネージャで、やはり XCB で書かれています。awesome と見比べたりしました。

joken ではこのように急に WM が書きたくなって仕方がなくなり他のことに身が入らなくなるけどすぐ飽きる人間やその他が活動しています。悪いことは言わないからプログラミングをやりたいなら joken に入ったほうがいい。あるいはどの部活にも入らないで家でもくもくしていたほうがいい。でもそれは joken に入るのの下位互換なきがしてる（部費が発生するので false)。
競技プログラミングをやりたい人間、高専プロコンに出たい人間、セキュリティに興味がある人間なども激しく歓迎します。
えーとえとばいばい。見返したら本当にイカれてんなこの文章。

```
import xcb.xcb;

import std.experimental.logger;

class Client {
  public:
  xcb_window_t window;
  this(xcb_window_t window) {
    this.window = window;
  }
}

void main()
{
  auto conn = xcb_connect(null, null);
  if (conn is null || xcb_connection_has_error(conn) != 0) {
    fatal("failed to connect X Server");
  }
  auto screen = xcb_setup_roots_iterator(xcb_get_setup(conn))
      .data;
```

```
auto root = screen.root;

uint ROOT_EVENT_MASK = XCB_EVENT_MASK_SUBSTRUCTURE_REDIRECT
    |
  XCB_EVENT_MASK_SUBSTRUCTURE_NOTIFY|
  XCB_EVENT_MASK_STRUCTURE_NOTIFY|
  XCB_EVENT_MASK_ENTER_WINDOW|
  XCB_EVENT_MASK_POINTER_MOTION|
  XCB_EVENT_MASK_POINTER_MOTION_HINT|
  XCB_EVENT_MASK_KEY_PRESS|
  XCB_EVENT_MASK_BUTTON_PRESS|
  XCB_EVENT_MASK_BUTTON_RELEASE;

auto cookie = xcb_change_window_attributes_checked(conn,
    root, XCB_CW_EVENT_MASK, &ROOT_EVENT_MASK);
if (xcb_request_check(conn, cookie) !is null) {
  fatal("another window manager is already running");
}
xcb_flush(conn);

Client[xcb_window_t] clients;
Client focusing = null;
int oldx, oldy;
bool is_moving = false;
while (true) {
  auto event = xcb_wait_for_event(conn);
  switch (event.response_type) {
    case XCB_MAP_REQUEST:
    auto e = cast(xcb_map_request_event_t*)event;
    xcb_grab_button(conn, 1, e.window,
        XCB_EVENT_MASK_BUTTON_PRESS|
        XCB_EVENT_MASK_BUTTON_RELEASE|
        XCB_EVENT_MASK_POINTER_MOTION,
```

```
            XCB_GRAB_MODE_ASYNC,
            XCB_GRAB_MODE_ASYNC,
            e.window, XCB_NONE,
            XCB_BUTTON_INDEX_1,   // LEFT BUTTON
            XCB_MOD_MASK_1|XCB_MOD_MASK_2);
        xcb_grab_key(conn, 0, root,
            XCB_MOD_MASK_1|XCB_MOD_MASK_2,
            cast(xcb_keycode_t)24,   //q
            XCB_GRAB_MODE_ASYNC,
            XCB_GRAB_MODE_ASYNC);
        uint window_event = XCB_EVENT_MASK_STRUCTURE_NOTIFY|
          XCB_EVENT_MASK_ENTER_WINDOW;
        xcb_change_window_attributes(conn, e.window,
            XCB_CW_EVENT_MASK, &window_event);
        xcb_map_window(conn, e.window);
        xcb_flush(conn);

        auto new_client = new Client(e.window);
        clients[e.window] = new_client;
        focusing = new_client;
        break;
    case XCB_BUTTON_PRESS:
        auto e = cast(xcb_button_press_event_t*)event;
        if (e.detail == XCB_BUTTON_INDEX_1 &&
            (e.state & XCB_MOD_MASK_1) && focusing !is null) {
              auto geometry_c = xcb_get_geometry(conn, focusing
                  .window);
          auto geometry_r = xcb_get_geometry_reply(conn,
              geometry_c, null);
          if (geometry_r is null) {
            break;
          }
          is_moving = true;
```

```
      oldx = geometry_r.x - e.root_x;
      oldy = geometry_r.y - e.root_y;
    }
    break;
  case XCB_MOTION_NOTIFY:
    auto e = cast(xcb_motion_notify_event_t*)event;
    if (is_moving && focusing !is null) {
      uint[] values = [
        oldx+e.root_x,
      oldy+e.root_y,
      ];
      xcb_configure_window(conn, focusing.window,
          cast(ushort)(XCB_CONFIG_WINDOW_X|
              XCB_CONFIG_WINDOW_Y),
          values.ptr);
      xcb_flush(conn);
    }
    break;
  case XCB_BUTTON_RELEASE:
    is_moving = false;
    break;
  case XCB_ENTER_NOTIFY:
    auto e = cast(xcb_enter_notify_event_t*)event;
    if (auto client = e.event in clients) {
      uint[] values = [XCB_STACK_MODE_ABOVE];
      xcb_configure_window(conn, client.window,
          XCB_CONFIG_WINDOW_STACK_MODE, values.ptr);
      xcb_flush(conn);
      focusing = *client;
    }
    break;
  case XCB_KEY_PRESS:
    auto e = cast(xcb_key_press_event_t*)event;
```

```
        if (e.detail == 24 &&
            (e.state & XCB_MOD_MASK_1) &&
            focusing !is null) {
              xcb_destroy_window(conn, focusing.window);
          xcb_flush(conn);
          focusing = null;
        }
        break;
      default:
        break;
      }
    }
}
```