yui

hello world!

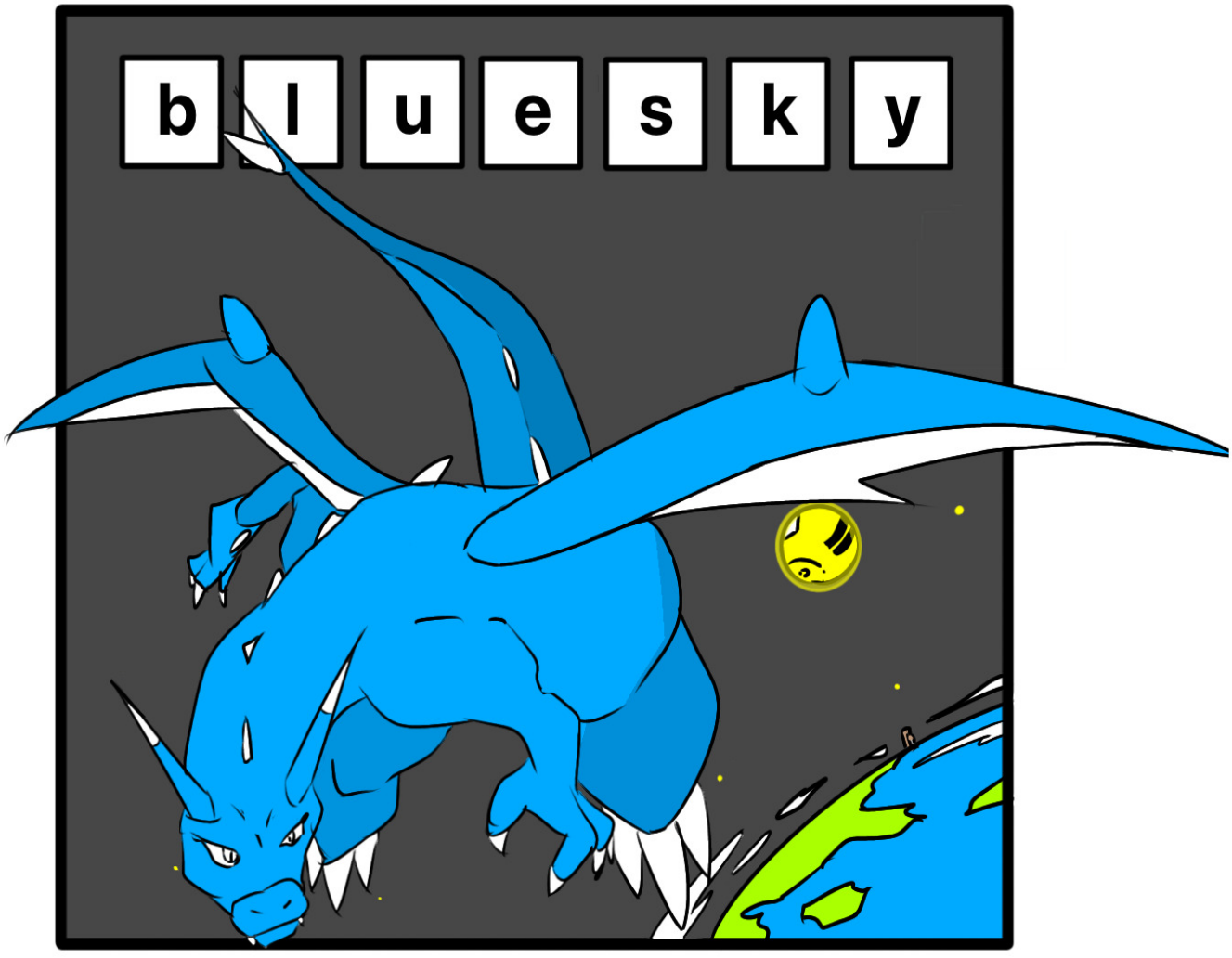


Table of Contents

hello world! bluesky

download | web

card

card

この書籍の第一版にはリアルカードが付属しています。

全3種類のうち1枚がランダムで当たります。

card	%
龍卵	14/20
青空	5/20
???	1/20

ランダムの仕組み

発送順の数字とカードの数字が紐付けられています。

この情報は暗号化され公開されています。

/card/book_0_public.pem

```
-----BEGIN CERTIFICATE-----
MIIC4TCCAcmgAwIBAgIUS5jY7UgomgdXw17v9c1DPCjFd78wDQYJKoZIhvcN
AQEL
BQAwADAeFw0yMzA3MjMwOTE1MTVaFw0yMzA4MjIwOTE1MTVaMAAwggEiMA0G
CSqG
SIb3DQEBAQUAA4IBDwAwggEKAoIBAQDJktVVRo5n2GvwwJFSeKGj7tnQsCTD
LSpr
1Q62zwXh4VsgGjoyo5+2QfXwQourEfDW/up4yG5YrO7m0utc0PF0DQKbsnze
qdkg
HWMUAiZGk1qI9QFE9jSs2O+O5+tljHQYxLNhHfcQ+dIF0kUWDpVer0k1t4xc
4HjJ
xvlUUEfOybMe2D44vLSjMWNcf61yzTkJWuMEn/ICK6/WzhH/1fGqn56F93s/
Lo1B
lc514Cioa9MMsLFb91wTqNPkoF3QHz4GuOC+DxHz5cKi9TtdztQ33Kh372hU
4Lkf
VXi8/61aKxLWbaly9UISJLbNgBkyX8pEtZRzwVmm8dVTr5Sh/a7DAgMBAAGj
UzBR
MB0GA1UdDgQWBBSHTOQhmfrn2ENIjPscI8ZFINFTdTAfBgNVHSMEGDAWgBSH
TOQh
mfrn2ENIjPscI8ZFINFTdTAPBgNVHRMBAf8EBTADAQH/MA0GCSqGSIb3DQEB
CwUA
A4IBAQAqa8d/wkBWzB6xBgD9GBewnKrstxoLg8K0fcxfIUS1EeBchkdpepeq
6UkG
blrHjibfPwFJ822oSy71vUTNcPt1Hpdp93xrz7DBD3Q5EdLsgJNH65vDA0KJ
K9nj
UfTYvU4Wt4xL9DxL/WqDsbNFPkNaztGWtZG41nFRKuGj0714e9G3RiImWjO8
mFpg
OI+/qQKlF6DdDXjuzNJJ7QDZ4gsxg5HqmCc8OjQRWDuVhJrvS4JH2O+4TH59
1CPA
VrSPWuw6kSAbY7iVNXlpAOWM6jGOu37ZEdyhlmYpXGG7SbX2lswUoIqkm8eo
vjHt
RYI2FkFATbwxAdNp9aNFdamFKF+s
-----END CERTIFICATE-----
```

encrypt

すべての発送が決まった段階で秘密鍵が公開され、復号化できるようになります。

認証手順

```
pri=book_0_private.pem
enc=book_0.enc
openssl smime -decrypt -in $enc -binary -inform DEM -inkey
$pri
```

作成手順

```
f=book_0.json
pri=book_0_private.pem
pub=book_0_public.pem
enc=book_0.enc
openssl req -x509 -nodes -newkey rsa:2048 -keyout $pri -out
$pub -subj '/'
openssl smime -encrypt -aes256 -in $f -binary -outform DEM -
out $enc $pub
openssl smime -decrypt -in $enc -binary -inform DEM -inkey
$pri
```

part 1

part 1

本書はbluesky(https://bsky.app/)のユーザーを対象としたプログラミング入門です。

主に、`rust`というプログラミング言語とosのターミナル環境を使用します。

内容としては、`bluesky`または`mastodon`で遊べるカードゲーム用の簡単なプログラムを作成します。

このプログラムを作ると手持ちのカードを育成できます。

本書では、blueskyのapiを叩くこと、rustでのコマンド作成などを学ぶことができます。

この章のクイックスタートでは、技術者を対象に必要最小限の説明を行います。

初めての方は、この章を飛ばしてpart 2から始めてください。

updated : 2023-07-29

quick start

クイックスタート

```
handle=yui.syui.ai
curl -sL
"https://bsky.social/xrpc/com.atproto.repo.listRecords?
repo=${handle}&collection=app.bsky.feed.post&limit=1"
```

@yui.syui.aiに以下のmentionを送ります。

```
@yui.syui.ai /card egg
```

すると、タマゴのカードがもらえます。この隠しコマンドは誰でも実行できます。既に持っている人はその旨が表示されます。

このカードは自分のdidをbase64に変換して@yui.syui.aiに送ることとで育成できるようになっています。

```
$ echo did:plc:4hqjfn7m6n5hno3doamuhgef|base64
ZGlkOnBsYzo0aHFqZm43bTZuNWhubzNkb2FtdWhnZWYK

@yui.syui.ai /egg
ZGlkOnBsYzo0aHFqZm43bTZuNWhubzNkb2FtdWhnZWYK
```

1日のバトルポイントを消費するので注意してください。

これをコマンドで送る場合は以下のようになります。

env

```
data=`curl -sL -X POST -H "Content-Type: application/json" -
d "{\"identifier\":\"$handle\",\"password\":\"$pass\"}"
https://bsky.social/xrpc/com.atproto.server.createSession`
token=`echo $data|jq -r .accessJwt`
did=`echo $data|jq -r .did`
base=`echo $did|base64`

handle_m=yui.syui.ai
did_m=`curl -sL -X GET -H "Content-Type: application/json" -
H "Authorization: Bearer $token"
"https://bsky.social/xrpc/app.bsky.actor.getProfile?
actor=${handle_m}"|jq -r .did`
at=@${handle_m}
s=0
e=`echo $at|wc -c`
text="$at /egg $base"
col=app.bsky.feed.post
created_at=`date --iso-8601=seconds`
```

json

```
json="{
        \"did\": \"$did\",
        \"repo\": \"$handle\",
        \"collection\": \"$col\",
        \"record\": {
            \"text\": \"$text\",
            \"\$type\": \"$col\",
            \"createdAt\": \"$created_at\",
            \"facets\": [
            {
                \"\$type\": \"app.bsky.richtext.facet\",
```

```
                \"index\": {
                    \"byteEnd\": $e,
                    \"byteStart\": $s
                },\"features\": [
                {
                    \"did\": \"$did_m\",
                    \"\$type\":
\"app.bsky.richtext.facet#mention\"
                }
                ]
            }
            ]
        }
}"
```

post

```
curl -sL -X POST -H "Content-Type: application/json" \
         -H "Authorization: Bearer $token" \
         -d "$json" \

https://bsky.social/xrpc/com.atproto.repo.createRecord
```

example

example

ここではlexiconsの使用例を紹介します。

option

```
# reverse
curl -sL
"https://bsky.social/xrpc/com.atproto.repo.listRecords?
repo=${handle}&collection=app.bsky.feed.post&reverse=true"
```

login

```
handle=yui.syui.ai
pass=xxx
curl -sL -X POST -H "Content-Type: application/json" \
        -d "
{\"identifier\":\"$handle\",\"password\":\"$pass\"}" \

https://bsky.social/xrpc/com.atproto.server.createSession

# token
token=`curl -sL -X POST -H "Content-Type: application/json"
-d "{\"identifier\":\"$handle\",\"password\":\"$pass\"}"
https://bsky.social/xrpc/com.atproto.server.createSession|jq
-r .accessJwt`

# did
did=`curl -sL -X POST -H "Content-Type: application/json" -d
"{\"identifier\":\"$handle\",\"password\":\"$pass\"}"
https://bsky.social/xrpc/com.atproto.server.createSession|jq
-r .did`

# profile
curl -sL -X GET -H "Content-Type: application/json" \
        -H "Authorization: Bearer $token" \

"https://bsky.social/xrpc/app.bsky.actor.getProfile?
actor=${handle}"

# notify
curl -sL -X GET -H "Content-Type: application/json" \
        -H "Authorization: Bearer $token" \

https://bsky.social/xrpc/app.bsky.notification.listNotificat
ions
```

post

```
col=app.bsky.feed.post

created_at=`date --iso-8601=seconds`

json="{
    \"repo\": \"$handle\",
        \"did\": \"$did\",
        \"collection\": \"$col\",
        \"record\": {
            \"text\": \"hello world\",
            \"createdAt\": \"$created_at\"
```

```
        }
}"

# post
curl -sL -X POST -H "Content-Type: application/json" \
         -H "Authorization: Bearer $token" \
         -d "$json" \

https://bsky.social/xrpc/com.atproto.repo.createRecord
```

mention

example.json

```
{
  "did": "did:plc:4hqjfn7m6n5hno3doamuhgef",
  "repo": "yui.syui.ai",
  "collection": "app.bsky.feed.post",
  "record": {
    "text": "test",
    "$type": "app.bsky.feed.post",
    "createdAt": "2023-07-20T13:05:45+09:00",
    "facets": [
      {
        "$type": "app.bsky.richtext.facet",
        "index": {
          "byteEnd": 13,
          "byteStart": 0
        },
        "features": [
          {
            "did": "did:plc:4hqjfn7m6n5hno3doamuhgef",
            "$type": "app.bsky.richtext.facet#mention"
          }
        ]
      }
    ]
  }
}

# mention
col=app.bsky.feed.post
handle_m=yui.syui.ai
did_m=`curl -sL -X GET -H "Content-Type: application/json" -
H "Authorization: Bearer $token"
"https://bsky.social/xrpc/app.bsky.actor.getProfile?
actor=${handle_m}"|jq -r .did`
at=@${handle_m}
s=0
e=`echo $at|wc -c`

json="{
        \"did\": \"$did\",
        \"repo\": \"$handle\",
        \"collection\": \"$col\",
        \"record\": {
            \"text\": \"$text\",
            \"\$type\": \"app.bsky.feed.post\",
            \"createdAt\": \"$created_at\",
            \"facets\": [
            {
                \"\$type\": \"app.bsky.richtext.facet\",
                \"index\": {
                    \"byteEnd\": $e,
                    \"byteStart\": $s
                },\"features\": [
```

```
                {
                    \"did\": \"$did_m\",
                    \"\$type\":
\"app.bsky.richtext.facet#mention\"
                }
                ]
            }
            ]
        }
}"

curl -sL -X POST -H "Content-Type: application/json" \
         -H "Authorization: Bearer $token" \
         -d "$json" \

https://bsky.social/xrpc/com.atproto.repo.createRecord
```

output

```
{"uri":"at://did:plc:4hqjfn7m6n5hno3doamuhgef/app.bsky.feed.
post/3k2wkbvcasf24","cid":"bafyreiecswq5qhk7f4xxztevzbfynocs
gmjrmr3hwqoluhhzvqgowalivi"}
```

reply

example.json

```
{
  "repo": "yui.syui.ai",
  "did": "did:plc:4hqjfn7m6n5hno3doamuhgef",
  "collection": "app.bsky.feed.post",
  "record": {
    "text": "reply",
    "createdAt": "2023-07-20T13:05:45+09:00",
    "reply": {
      "root": {
        "cid":
"bafyreiecswq5qhk7f4xxztevzbfynocsgmjrmr3hwqoluhhzvqgowalivi
",
        "uri":
"at://did:plc:4hqjfn7m6n5hno3doamuhgef/app.bsky.feed.post/3k
2wkbvcasf24"
      },
      "parent": {
        "cid":
"bafyreiecswq5qhk7f4xxztevzbfynocsgmjrmr3hwqoluhhzvqgowalivi
",
        "uri":
"at://did:plc:4hqjfn7m6n5hno3doamuhgef/app.bsky.feed.post/3k
2wkbvcasf24"
      }
    }
  }
}

# reply
col=app.bsky.feed.post
uri=at://did:plc:4hqjfn7m6n5hno3doamuhgef/app.bsky.feed.post
/3k2wkbvcasf24
cid=bafyreiecswq5qhk7f4xxztevzbfynocsgmjrmr3hwqoluhhzvqgowal
ivi

json="{
    \"repo\": \"$handle\",
        \"did\": \"$did\",
        \"collection\": \"$col\",
```

```
        \"record\": {
            \"text\": \"reply\",
            \"createdAt\": \"$created_at\",
            \"reply\": {
                \"root\": {
                    \"cid\": \"$cid\",
                    \"uri\": \"$uri\"
                },
                \"parent\": {
                    \"cid\": \"$cid\",
                    \"uri\": \"$uri\"
                }
            }
        }
}"

curl -sL -X POST -H "Content-Type: application/json" \
         -H "Authorization: Bearer $token" \
         -d "$json" \

https://bsky.social/xrpc/com.atproto.repo.createRecord
```

like

```
# reply
col=app.bsky.feed.like
uri=at://did:plc:4hqjfn7m6n5hno3doamuhgef/app.bsky.feed.post
/3k2wkbvcasf24
cid=bafyreiecswq5qhk7f4xxztevzbfynocsgmjrmr3hwqoluhhzvqgowal
ivi

json="{
    \"repo\": \"$handle\",
        \"did\": \"$did\",
        \"collection\": \"$col\",
        \"record\": {
            \"createdAt\": \"$created_at\",
            \"subject\": {
                \"cid\": \"$cid\",
                \"uri\": \"$uri\"
            }
        }
}"

curl -sL -X POST -H "Content-Type: application/json" \
         -H "Authorization: Bearer $token" \
         -d "$json" \

https://bsky.social/xrpc/com.atproto.repo.createRecord
```

follow

```
col=app.bsky.graph.follow
handle_m=yui.syui.ai
did_m=`curl -sL -X GET -H "Content-Type: application/json" -
H "Authorization: Bearer $token"
"https://bsky.social/xrpc/app.bsky.actor.getProfile?
actor=${handle_m}"|jq -r .did`

json="{
    \"repo\": \"$handle\",
        \"did\": \"$did\",
        \"collection\": \"$col\",
        \"record\": {
            \"createdAt\": \"$created_at\",
```

```
            \"subject\": \"$did_m\"
        }
}"

curl -sL -X POST -H "Content-Type: application/json" \
         -H "Authorization: Bearer $token" \
         -d "$json" \

https://bsky.social/xrpc/com.atproto.repo.createRecord
```

unfollow

```
$ curl -sL -X GET -H "Content-Type: application/json" \
         -H "Authorization: Bearer $token" \

"https://bsky.social/xrpc/app.bsky.graph.getFollowers?
actor=${handle}&cursor=${cursor}" \
         |jq -r ".cursor"

1688489398761::bafyreieie7opxd5mojipvk3xe3h65u3qvpungskqxaml
depctfbd6xhdcu

cursor=1688489398761::bafyreieie7opxd5mojipvk3xe3h65u3qvpung
skqxamldepctfbd6xhdcu

$ curl -sL -X GET -H "Content-Type: application/json" \
         -H "Authorization: Bearer $token" \

"https://bsky.social/xrpc/app.bsky.graph.getFollowers?
actor=${handle}&cursor=${cursor}" \
         |jq -r ".followers|.[0].viewer.followedBy"

at://did:plc:uqzpqmrjnptsxezjx4xuh2mn/app.bsky.graph.follow/
3k2wkjr6cnj2x

col=app.bsky.graph.follow
rkey=at://did:plc:uqzpqmrjnptsxezjx4xuh2mn/app.bsky.graph.fo
llow/3k2wkjr6cnj2x

handle_m=yui.syui.ai
did_m=`curl -sL -X GET -H "Content-Type: application/json" -
H "Authorization: Bearer $token"
"https://bsky.social/xrpc/app.bsky.actor.getProfile?
actor=${handle_m}"|jq -r .did`

json="{
    \"repo\": \"$handle\",
        \"did\": \"$did\",
        \"collection\": \"$col\",
        \"rkey\":\"$rkey\",
        \"record\": {
            \"createdAt\": \"$created_at\",
            \"subject\": \"$did_m\"
        }
}"

curl -sL -X POST -H "Content-Type: application/json" \
         -H "Authorization: Bearer $token" \
         -d "$json" \

https://bsky.social/xrpc/com.atproto.repo.deleteRecord
```

part 2

part 2

この章では、よく使う単語の説明と環境の説明を行います。

主に、osによって動作環境が異なるため、それに向けた解説です。

本書で使用するパッケージのインストールなどをまとめて紹介します。

よくわからない方はこの章をご参照ください。

bluesky

bluesky

bluesky(プロトコル)にatを採用したsnsです。

atは以降、atprotoとします。

bluesky.teamが開発、運用しているatprotoのモデルサービスという位置づけです。

bsky.teamの目標は、様々なサービスの裏でatprotoが採用され、サービス間で意思疎通を図れるようにすることです。

今までのサービスはそのサービス内でしかアカウントが有効ではありませんでした。そのためサービスごとにアカウントを切り替えてやり取りする必要がありました。これを変えていこうという試みです。

bluesky.は、pds,plc,bgsで動作することを予定しています。

それぞれの役割を述べると、pdsがbluesky本体です。

plcはdnsのようなものでhandleとdidを登録し、名前解決を行います。

bluesky.は、基本的にpdsのみで動作します。

しかし、アカウント作成時はplcに接続するため、plcへの接続がないとエラーになります。

アカウントが作成されている状態ではplcは必ずしも必要ありません。handleの登録や変更があったときに必要になります。

bgsは他のpdsとつながる際、アカウントのtimelineを構築します。

graph TD; A[pds]-->B[plc]; C[pds]-->B[plc]; D[pds]-->B[plc];
graph TD; A[pds]-->B[bgs]; C[pds]-->B[bgs]; D[pds]-->B[bgs];

dns

上記のdnsとは何かというと、インターネット上で名前解決を行うサーバーのことを言います。

インターネットではip addressという数字で繋がります。

例えば、googleに接続する場合は172.217.25.174です。

```
$ dig google.com
google.com.             291     IN      A
172.217.25.174
```

試しにブラウザにこの数字を入れてみてください。google.comにつながるはずです。

しかし、数字というのは人間にとって覚えにくく、扱いにくいものなので、通常は、アルファベットに置き換えた名前がつけられます。

その名前をip addressという数字につなげる役割を持ったサーバーをdnsと言います。

以下は、目的のホストまでの経路を表示するコマンド。いくつかのサーバーを経由して繋がっている事がわかる。

```
$ traceroute google.com
20.27.177.113
17.253.144.10
172.217.25.174
```

自分のip addressを知りたければ、`ipinfo.io`を使うと便利。

```
$ curl -sL ipinfo.io
20.27.177.113
```

plc

現時点でよく使われているplcです。すべてbsky.teamが提供しています。

https://plc.directory

https://plc.bsky-sandbox.dev

具体的には以下のように使います。

`https://plc.directory/export`

`https://plc.directory/export?after=1970-01-01T00:00:00.000Z`

`https://plc.directory/did:plc:oc6vwdlmk2kqyida5i74d3p5`

`https://plc.directory/did:plc:oc6vwdlmk2kqyida5i74d3p5/log`

.env

```
#DID_PLC_URL=https://plc.directory
DID_PLC_URL=https://plc.bsky-sandbox.dev
```

便利なサービス

blueskyはapiもpdsも公開されているので様々なサービスが開発されています。

代表的なサービスを紹介します。

https://firesky.tv : グローバルタイムラインのストリーム。色々と設定できたり、フィルタリングできたり

https://bsky.jazco.dev : ユーザーの視覚化

https://bsky.jazco.dev/stats : ユーザーのポスト数

https://vqv.app : ユーザーのプロフィール集計など

https://atscan.net : pdsのスキャンやdid

https://skybridge.fly.dev : mastodonのclientでblueskyをやるためのurl

https://tapbots.com/ivory : mastodon clientのivoryに対応

https://skyfeed.app : feedの生成

terminal

terminal

ここでは、osごとに必要なコマンドや環境を用意することを目標にします。

`terminal(ターミナル)`とはwindowsでいうcmd(コマンドプロンプト)が有名です。わかりやすく言うと黒い画面を指します。端末や`term`などとも呼ばれたりします。

terminalにも色々なterminal、つまり、アプリ(ソフトウェア)があります。

個人的にはwezermがオススメですが、ここでは、os固有のものを使用します。

package manager

最初に、`package manager(パッケージ・マネージャー)`の解説を行います。

今回、それぞれのosでパッケージ・マネージャーの導入が必要です。

なお、ここでのパッケージやプログラムはアプリと言いかえることもできます。

パッケージ・マネージャーは、アプリのインストールを簡単にしてくるものと考えてください。

通常、アプリはsource(ソース)をbuild(ビルド)またはcompile(コンパイル)し、作成されたbinary(バイナリ)を実行することで動作します。

windowsでいうと`.exe`がbinaryに当たります。

binaryは、osによって異なります。

ちなみに、sourceは`src`と略され、binaryは`bin`に略されることが多いです。

話を戻すと、どのパッケージ(binary)をどこからダウンロードし、どこに置くのか、それらを自動処理してくれるのがパッケージマネージャーです。

なぜこのようになっているのかというと、sourceのbuildには時間がかかるからです。

ですから、大体のパッケージは既に当該osでbuildされているbinaryをserver(サーバー)からダウンロードしてくるだけです。

その役割を担っているのが大半のパッケージ・マネージャーと呼ばれるものになります。

以降、このパッケージ・マネージャーを導入して`terminal`から使うことになります。

windows

windowsユーザーの方に向けて必要な環境を解説します。

- winget
- scoop
- windows terminal

- wsl

windows環境は注意が必要です。windowsは基本的にdocs通り動かないということを覚えておいてください。

例えば、github/microsoftに書いてあるコマンドを実行しても、大半は動きません。動かないことがあります。

したがって、ご自身の環境に合わせて設定したり読み替えたりする必要が出てきます。

まず、windowsのパッケージ・マネージャーとしてwingetを導入します。

`win+r`を押して`powershell`と入力し、powershellを起動します。powershellは以降、`pwsh`と略します。

以下のコマンドを実行します。

pwsh

```
Install-Module -Name Microsoft.WinGet.Client
```

```
Untrusted repository
You are installing the modules from an untrusted repository.
If you trust this repository, change its
InstallationPolicy value by running the Set-PSRepository
cmdlet. Are you sure you want to install the modules from
'PSGallery'?
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend
[?] Help (default is "N"): A
```

次にwindows terminalをインストールします。

パッケージの検索

```
winget search "windows terminal"
```

```
 Name                      Id
Version      Source
 ----------------------------------------------------------
---------------------
 Windows Terminal          9N0DX20HK701
Unknown      msstore
 Windows Terminal Preview 9N8G5RFZ9XK3
Unknown      msstore
 Windows Terminal          Microsoft.WindowsTerminal
1.16.10261.0 winget
 Windows Terminal Preview Microsoft.WindowsTerminal.Preview
1.17.10234.0 winget
```

terminalのインストール

```
winget install 9N0DX20HK701
or
winget install Microsoft.WindowsTerminal
```

windowsはshellが非常に扱いづらいので、wslでlinux(ubuntu)を動作させます。基本的にrustやshellはlinux環境を前提に解説します。

wslの導入

```
wsl --install
wsl --install -d Ubuntu
```

もしlinuxではなくwindows環境がいい場合は、パッケージ・マネージャーのscoop などからcurlなどをインストールして対応してください。

scoopのインストール

```
Set-ExecutionPolicy RemoteSigned -Scope CurrentUser #
Optional: Needed to run a remote script the first time
irm get.scoop.sh | iex
```

pwsh

```
scoop install curl git rust
```

pwshをinstall, updateします。

```
winget install Microsoft.PowerShell
winget upgrade --all
```

mac

- terminal
- homebrew

macの場合はデフォルトのterminalを使用します。

finderを開いてcmd+shift+uを押します。その中にterminal(ターミナル.app)があると思います。

まずパッケージ・マネージャーのhomebrewをインストールします。

brewのインストール

```
/bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/HEAD/inst
all.sh)"
```

```
brew install curl git zsh rust
```

linux

linuxユーザーは説明が不要なので省略します。

私はarchlinuxを使用します。

```
pacman -Syu curl git zsh rust
```

```
$ cargo version
cargo 1.70.0
```

shell

shell

パッケージ・マネージャーが導入できたと思うので、まずは`curl`をインストールしてみてください。

```
# windows
scoop install curl

# mac
brew install curl

# linux(ubuntu)
sudo apt install curl
```

そして、terminalで以下のコマンドを実行してください。

```
curl https://bsky.social/xrpc/_health
```

結果は、以下の通り。bluesky(bsky.social)のpdsのversionが返ってくるはずです。

output

```
{"version":"b2ef3865bc143bfe4eef4a46dbd6a44053fa270d"}
```

なお、`curl`がうまく動作しない場合、インストールされたbinaryにpath(パス)が通っていない事が考えられます。

これも主にwindowsで発生しやすいと思われます。

ここで、pathについて少し解説します。

path

terminalを起動すると、そこではshellと呼ばれるプログラムが待機されています。

ユーザーはこのshellを通してコマンドを実行することになります。

shellにも色々なshellがあります。

例えば、windowsにはmicrosoftの`cmd`, `pwsh`というshellがあります。

unix(mac), linux(ubuntu)の場合は`bash`, `zsh`などがあります。

shellはコマンドを実行する際、`PATH`に追加されているディレクトリを省略することができます。

例えば、`curl`が`/usr/bin/curl`にインストールされたとしましょう。この場合は、shellは以下のコマンドを実行しなければなりません。

```
/usr/bin/curl --help
```

しかし、`PATH`に`/usr/bin`が追加されている場合、ディレクトリの記述を省略することができます。

```
curl --help
```

プログラム本体(binary)がどこにあるのか探す場合は、以下のようなコマンドを使います。

```
which curl
```

ただし、pathが通っていないと使えません。

pathを通すには、環境変数に当該ディレクトリを記述します。

```
PATH=$PATH:/usr/bin
```

なお、ディレクトリ(directory)は`dir`や**フォルダ**と呼ばれることがあります。

記法の"$"

次に、shellの記述方式に関する注意点を書きます。

```
which curl
```

```
$ which curl
```

これらは同じ意味を持ちます。

もし文章でshellの実行を説明する場合、先頭に`$`を記述する慣習があります。

この`$`は「shellで実行します」という意味です。

例えば、コマンドの実行結果と一緒に載せたい場合、下記のようになります。

```
$ which curl
/usr/bin/curl
```

このようにコマンドと実行結果を一緒に載せたい場合がよくあり、もし`$`がないと、どちらがコマンドで、どちらが実行結果かが分かりづらくなるからです。

本書では、コピーの弊害などを考慮して、`$`をなるべく省略しています。

しかし、本来であれば、全てのコードレイアウトにshellで実行する場合は`$`を記述すべきと考えられています。

shebang

次に、shell script(シェル・スクリプト)やshebang(シバン)について解説します。

この辺はshellによって違いがありますが、`bash`を前提に話をします。

テキストファイルに以下を記述して、実行権限を与えて実行してください。

test.sh

```
#!/bin/bash
curl https://bsky.social/xrpc/_health
```

実行権限の付与、及び実行は以下のコマンドです。

```
chmod +x test.sh
./test.sh
```

するとbsky.socialのversionが出力されます。

```
{"version":"b2ef3865bc143bfe4eef4a46dbd6a44053fa270d"}
```

テキストファイルの最初の行`#!/bin/bash`がシバンと呼ばれるものです。

ここで、テキストファイルをどのプログラム言語で実行するのかが指定されます。

次はプログラム言語の簡単な解説を行います。

rust

rust

次にプログラミング言語の`rust`をインストールします。

```
brew install rust
```

rustは`cargo`というパッケージ・マネージャーを通して動作します。

バージョン(version)を調べるには`cargo`を利用します。

```
$ cargo version
cargo 1.71.0
```

rustは、様々なプログラミング言語の中で非常に難易度が高い言語と言われています。

特徴としては、一度構築すると安定して動作するけど、動かすまでに時間がかかるというイメージです。

また、新しい実装を追加するのも他の言語と比べ時間がかかるかもしれません。

lang

プログラミング言語は、`lang`と略されることがあります。

例えば、`go`というプログラム言語があります。

しかし、goという単語には色々な意味があります。そのため`golang`, `go-lang`と呼ばれることがあります。

part 3

part 3

この章では、rustで具体的なコードを書いて、プログラムを動かします。

hello world

init

まず、rustでプログラムの雛形を作ります。

```
mkdir -p ~/rust
cd ~/rust
cargo init
```

```
.
├── Cargo.toml
└── src
    └── main.rs
```

これらのファイルは自分で作成してもいいし、`init`で作成してもいいです。

Cargo.toml

```
[package]
name = "rust"
version = "0.1.0"
edition = "2021"

# See more keys and their definitions at https://doc.rust-
lang.org/cargo/reference/manifest.html

[dependencies]
```

src/main.rs

```
fn main() {
    println!("Hello, world!");
}
```

ちなみに、コンピュータ上ではディレクトリもファイルなのです。この話は面倒なので省略します。

editor

次にプログラムの中身を確認してみます。

確認するには、editor(エディタ)を使用します。私は`vim`を使いますが、visual studioがオススメかな。

```
brew install vim
vim src/main.rs
```

src/main.rs

```
fn main() {
    println!("Hello, world!");
}
```

これは`hello world!`という文字列を出力するプログラムです。

build

このsrcを`build`して`binary`、つまり、アプリ本体に変換することで、そのパソコンで実行できるようになります。

```
cargo build
```

target/debug/rust

```
target
└── debug
    ├── rust ← このファイルがbinary
    └── rust.d
```

rustはワンバイナリと言って、コンパイル結果が単一ファイルなので、非常に良い言語です。

```
$ ./target/debug/rust
Hello, world!
```

seahorse

seahorse

次にksk001100/seahorseというframework(フレームワーク)を導入します。

このフレームワークはcli(command line interface)を書くためのものです。

cliは、簡単に言うと、これまで実行してきた`which`や`curl`と同じコマンドのことです。今から自分のコマンドを作成します。

難しそうと思われる方がいるかもしれませんが、`seahorse`という素晴らしいframeworkを使えば簡単です。

まずは`seahorse`をインストールするわけですが、rustでlibrary(ライブラリ)をインストールするには、`Cargo.toml`にpackage nameを書きます。これでbuildする際に自動でインストールされます。

なお、libraryは`lib`と略されることがあります。

Cargo.toml

```
[package]
name = "rust"
version = "0.1.0"
edition = "2021"

[dependencies]
seahorse = "*"
```

そして、`seahorse`を使う本体コードを書いていきます。

src/main.rs

```
use seahorse::{App, Context};
use std::env;

fn main() {
    let args: Vec<String> = env::args().collect();
    let app = App::new(env!("CARGO_PKG_NAME"))
        .action(s)

        ;
    app.run(args);
}

fn s(_c: &Context) {
    println!("Hello, world!");
}
```

内容はとてもシンプル。コマンドを実行すると`Hello, world!`が出力されます。

```
$ cargo build
$ ./target/debug/rust
Hello, world!
```

今までと何が違うのかと言うと、例えば、`help`オプションが自動でついています。

```
$ ./target/debug/rust -h

Name:
        rust
Flags:
        -h, --help : Show help
```

seahorseの凄さを理解してもらうため、読者自身に応用を考えてもらいましょう。

src/main.rs

```
use seahorse::{App, Context, Command};
use std::env;

fn main() {
    let args: Vec<String> = env::args().collect();
    let app = App::new(env!("CARGO_PKG_NAME"))
        .action(s)

        .command(
                Command::new("yes")
                .alias("y")
                .action(y),
                )
        .command(
                Command::new("no")
                .alias("n")
                .action(n),
                )

        ;
        app.run(args);
}

fn s(_c: &Context) {
    println!("Hello, world!");
}

fn y(_c: &Context) {
    println!("yes");
}

fn n(_c: &Context) {
    println!("no");
}
```

これを書いて、作成したコマンドを実行してみてください。

```
$ ./target/debug/rust

$ ./target/debug/rust y

$ ./target/debug/rust n
```

コードの差分、要点は以下になります。

```
  use seahorse::{App, Context
+ , Command
  };

+
.command(
        Command::new("yes")
        .alias("y")
        .action(y),
```

```
        )

fn y(_c: &Context) {
    println!("yes");
}
```

これらの値を書き換えたり、追加したりして、自由にコマンドを作ってみてください。

ここで、`Command::new`で指定した値はオプション名を意味します。

この場合は`rust yes`がこのコマンドの発行です。

`alias("y")`で省略を指定することができます。この場合は`rust y`になります。

`action(y)`は関数の`fn y`を指定しており、その中身が実行されます。`action`にコマンド本体の処理を書くことになります。

ちなみに、`action`は必ずしも関数を使う必要はありません。

例えば、以下のコードを所定の場所に追加してみてください。コマンドは`rust t`または`rust t foo`です。

src/main.rs

```
.command(
        Command::new("test")
        .alias("t")
        .action(|c| println!("Hello, {:?}", c.args)),
        )
```

```
$ ./target/debug/rust t bluesky
Hello, ["bluesky"]
```

cli

- CLI

`cli`は様々な意味を持ちます。上記のようなcli toolのことを指すこともあれば、terminal操作全般を指すこともあります。

cuiとgui

- CUI, GUI

cuiとguiというものがあります。今使っているのは`cui`です。

`cli`もほぼ同じような意味で使用されています。

cuiはterminal操作を意味し、guiはグラフィカルなos上の操作を意味します。

c-uiと分けられ、uiは単なるuiです。user interfaceの略。

windowsやmacなど一般的なosは、すべてgui操作が基本です。

author

seahorseの作者はkskさんです。

素晴らしいframeworkを感謝します。

reqwest

reqwest

rustのseanmonstar/reqwestを解説する前に、以下のコマンドを実行してみてください。

```
$ curl -sL
"https://bsky.social/xrpc/com.atproto.repo.listRecords?
repo=support.bsky.team&collection=app.bsky.feed.post"
```

これはsupport.bsky.socialのtimelineを取得するapiを叩いています。

reqwestは主にapiを叩くためのrustのlibだという理解で構いません。

では、実際にコードを書いてみます。

Cargo.toml

```
[package]
name = "rust"
version = "0.1.0"
edition = "2021"

[dependencies]
seahorse = "*"
reqwest = "*"
tokio = { version = "1", features = ["full"] }
```

```
use seahorse::{App, Context, Command};
use std::env;

fn main() {
    let args: Vec<String> = env::args().collect();
    let app = App::new(env!("CARGO_PKG_NAME"))
        .action(s)
        .command(
                Command::new("yes")
                .alias("y")
                .action(y),
                )
        .command(
                Command::new("no")
                .alias("n")
                .action(n),
                )
        .command(
                Command::new("test")
                .alias("t")
                .action(|c| println!("Hello, {:?}",
c.args)),
                )
        .command(
                Command::new("bluesky")
                .alias("b")
                .action(c_list_records),
                )

        ;
        app.run(args);
}

fn s(_c: &Context) {
```

```
    println!("Hello, world!");
}

fn y(_c: &Context) {
    println!("yes");
}

fn n(_c: &Context) {
    println!("no");
}

#[tokio::main]
async fn list_records() -> reqwest::Result<()> {
    let client = reqwest::Client::new();
    let handle= "support.bsky.team";
    let col = "app.bsky.feed.post";
    let body =
client.get("https://bsky.social/xrpc/com.atproto.repo.listRe
cords")
        .query(&[("repo", &handle),("collection", &col)])
        .send()
        .await?
        .text()
        .await?;
    println!("{}", body);
    Ok(())
}

fn c_list_records(_c: &Context) {
    list_records().unwrap();
}
```

これをcargo buildしていつものようにコマンドを実行します。

```
$ ./target/debug/rust b
```

以降はexample、つまり無駄なコマンドオプションを削除したコードを記述します。

コードの要点は以下の通り。

```
.command(
        Command::new("bluesky")
        .alias("b")
        .action(c_list_records),
        )

#[tokio::main]
async fn list_records() -> reqwest::Result<()> {
    let client = reqwest::Client::new();
    let handle= "support.bsky.team";
    let col = "app.bsky.feed.post";
    let body =
client.get("https://bsky.social/xrpc/com.atproto.repo.listRe
cords")
        .query(&[("repo", &handle),("collection", &col)])
        .send()
        .await?
        .text()
        .await?;
    println!("{}", body);
    Ok(())
}

fn c_list_records(_c: &Context) {
```

```
    list_records().unwrap();
}
```

query

queryの追加をしてみます。これで出力が1行になり、古い順になります。

src/main.rs

```
async fn list_records() -> reqwest::Result<()> {
    let client = reqwest::Client::new();
    let handle= "support.bsky.team";
    let col = "app.bsky.feed.post";
    let body =
client.get("https://bsky.social/xrpc/com.atproto.repo.listRe
cords")
        //.query(&[("repo", &handle),("collection", &col)])
        .query(&[("repo", &handle),("collection", &col),
("limit", &"1"),("revert", &"true")])
        .send()
        .await?
        .text()
        .await?;
    println!("{}", body);
    Ok(())
}
```

part 4

part 4

この章では、`seahorse`, `reqwest`を使いながらrustのコードを書き上げ、プログラムを完成へと導きます。

blueskyのlexiconsが重要になります。

もしわからない場合はpart 1をご参照ください。

ai

ai

ここでは、遊び要素を交えて、オリジナリティを追求します。各自、好きなものに設定してください。

まずコマンドアプリの名前ですね。今までは`rust`を使っていました。なぜなら`cargo init`で作成されたプログラム名が`rust`だからです。これは自動でフォルダ名が付けられます。

これを`ai`に変更してみます。

Cargo.toml

```
[package]
name = "ai"
```

なお、好きな名前を設定した場合、以降の解説ではコマンド名などを読みかえてください。

```
$ cargo build
$ ./target/debug/ai -h

Name:
        ai
Flags:
        -h, --help : Show help
Commands:
        y, yes     :
        n, no      :
        t, test    :
        b, bluesky :
```

cleanup

次に名残惜しいですが、いらないコマンドオプションを減らしましょう。

```
use seahorse::{App, Context, Command};
use std::env;

fn main() {
    let args: Vec<String> = env::args().collect();
    let app = App::new(env!("CARGO_PKG_NAME"))
        .action(c_list_records)
        .command(
                Command::new("bluesky")
                .alias("b")
                .action(c_list_records),
                )

        ;
        app.run(args);
}

#[tokio::main]
async fn list_records() -> reqwest::Result<()> {
    let client = reqwest::Client::new();
    let handle= "support.bsky.team";
    let col = "app.bsky.feed.post";
    let body =
```

```
client.get("https://bsky.social/xrpc/com.atproto.repo.listRecords")
        .query(&[("repo", &handle),("collection", &col),("limit", &"1"),("revert", &"true")])
        .send()
        .await?
        .text()
        .await?;
    println!("{}", body);
    Ok(())
}

fn c_list_records(_c: &Context) {
    list_records().unwrap();
}
```

ascii

いわゆる`ascii art`を入れてみます。

TheZoraiz/ascii-image-converterを使って生成します。

```
$ ascii-image-converter ai.png -H 50
```

`src/main.rs`の`.action(c_list_records)`を`.action(c_ascii_art)`に変更します。

src/main.rs

```
//.action(c_list_records)
.action(c_ascii_art)
```

なお、rustでは文頭の`//`がコメントになります。

コメントというのは、コードとして解釈されません。したがって、なにかメモを書くとき、これを使いましょう。

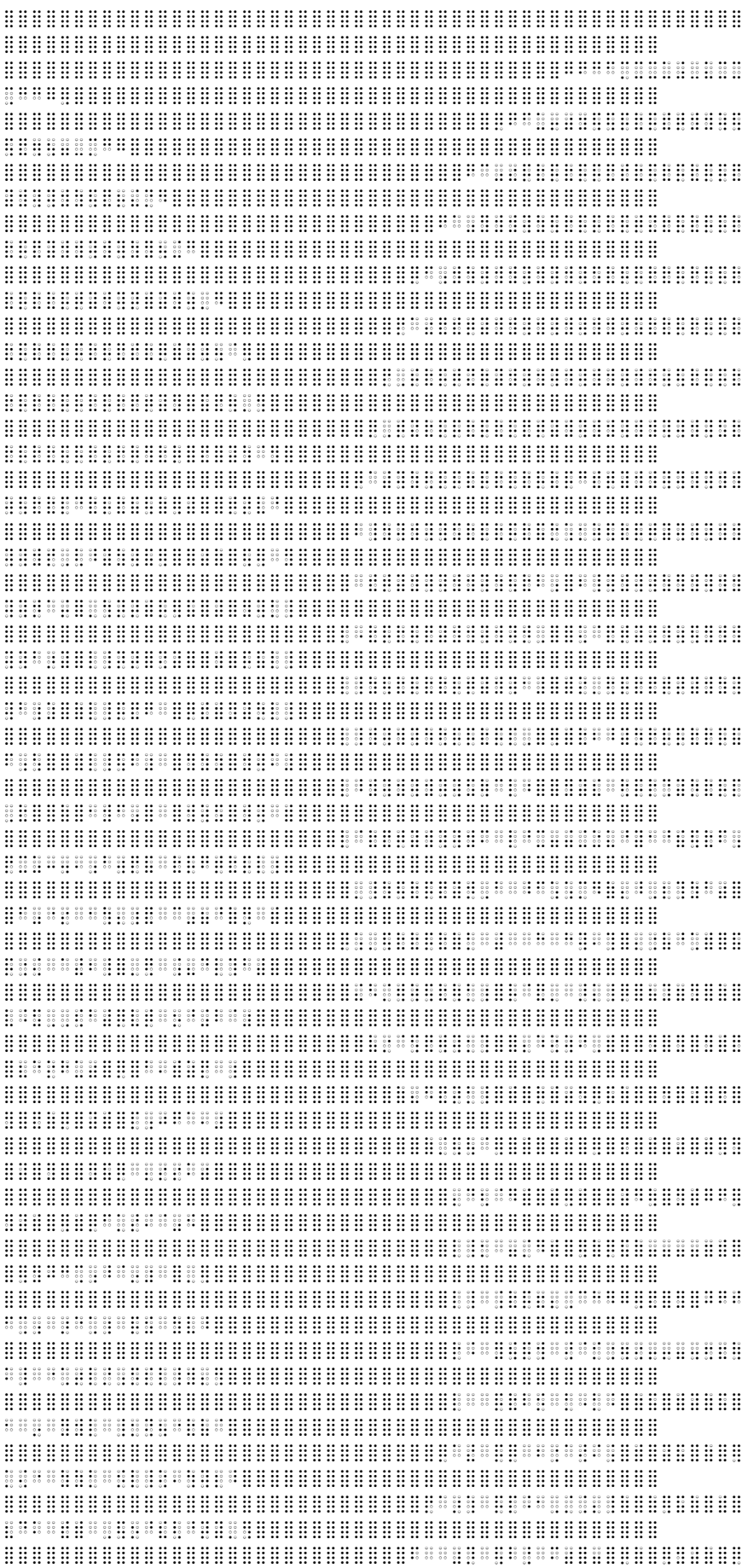
src/main.rs

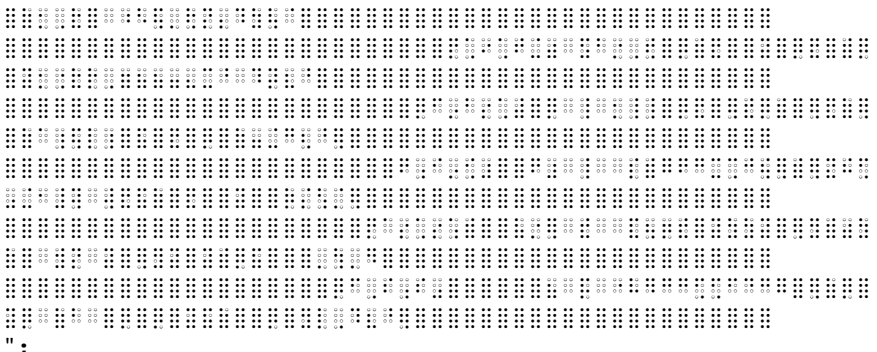
```
// これはアイのアスキーアートです
// 内容は関数の"c_ascii_art"を見てください
.action(c_ascii_art)
```

そして、`src/main.rs`の最後の行に以下のコード(関数)を追加します。

src/main.rs

```
fn c_ascii_art(_c: &Context) {
   let body = "
⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⠟⠉⣁⠉⠻⢿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿
⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⠿⠟⠛⠋⣉⣉⠡⠤⠤⠤⠤⠤⠤⠤⠤⠬⠤⣁⣉⡉⠛⠻⠿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿
⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⡿⢉⣤⡖⠚⣉⣡⣤⣦⣶⣶⣶⣶⣶⣶⣶⣶⣶⣶⣶⣴⣤⣤⠉⠑⣦⣌⠙⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿
⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⠀⣾⣿⣦⣄⣅⣌⣄⣤⣤⣤⣤⣤⣤⣤⣤⣤⣤⣤⣤⣤⣤⣤⣶⣾⣿⣿⣁⣘⡙⠻⠿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿
⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⠿⢃⣤⣨⣿⡿⡿⠟⠟⠛⠻⠚⠛⠛⠛⠛⠛⠛⠋⡋⡋⠛⠛⠉⠉⠼⠿⠿⠿⠿⠿⠷⢶⢤⣈⠙⠿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿
⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⠿⢋⡡⠞⠛⣉⣉⣤⣤⣶⣾⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣷⣶⣶⣶⣶⣶⣶⣶⣦⣦⣦⣦⣶⣾⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿
⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣟⢁⣀⣤⣴⣾⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿
```





```
";
    println!("{}", body);
}
```

できました。これでaiを実行すると、aiが表示されます。

```
$ cargo build
$ ./target/debug/ai
```

config

config

blueskyの認証系のコードを追加します。

具体的には`~/.config/ai/config.toml`に情報を書いておくと、
`~/.config/ai/token.toml`に認証情報を置くコマンドオプションを作成します。

~/.config/ai/config.toml

```
handle = "yui.syui.ai"
pass = "xxx"
host = "bsky.social"
```

Cargo.toml

```
[package]
name = "ai"
version = "0.1.0"
edition = "2021"

[dependencies]
seahorse = "*"
reqwest = { version = "*", features = ["blocking", "json"] }
tokio = { version = "1", features = ["full"] }
serde_derive = "1.0"
serde_json = "1.0"
serde = "*"
config = { git = "https://github.com/mehcode/config-rs",
branch = "master" }
shellexpand = "*"
toml = "*"
```

src/data.rs

```
use config::{Config, ConfigError, File};
use serde_derive::{Deserialize, Serialize};

#[derive(Debug, Deserialize)]
#[allow(unused)]
pub struct Data {
    pub host: String,
    pub pass: String,
    pub handle: String,
}

#[derive(Serialize, Deserialize)]
#[allow(non_snake_case)]
pub struct Token {
    pub did: String,
    pub accessJwt: String,
    pub refreshJwt: String,
    pub handle: String,
}

#[derive(Serialize, Deserialize)]
#[allow(non_snake_case)]
pub struct Tokens {
    pub did: String,
    pub access: String,
    pub refresh: String,
```

```
    pub handle: String,
}

impl Data {
    pub fn new() -> Result<Self, ConfigError> {
        let d = shellexpand::tilde("~") +
"/.config/ai/config.toml";
        let s = Config::builder()
            .add_source(File::with_name(&d))

.add_source(config::Environment::with_prefix("APP"))
            .build()?;
        s.try_deserialize()
    }
}
```

src/main.rs

```
pub mod data;
use seahorse::{App, Context, Command};
use std::env;
use std::fs;
use std::io::Write;
use std::collections::HashMap;

use data::Data as Datas;
use crate::data::Token;
use crate::data::Tokens;

fn main() {
    let args: Vec<String> = env::args().collect();
    let app = App::new(env!("CARGO_PKG_NAME"))
        //.action(c_ascii_art)
        .command(
                Command::new("bluesky")
                .alias("b")
                .action(c_list_records),
                )
        .command(
                Command::new("login")
                .alias("l")
                .action(c_access_token),
                )

        ;
        app.run(args);
}

#[tokio::main]
async fn list_records() -> reqwest::Result<()> {
    let client = reqwest::Client::new();
    let handle= "support.bsky.team";
    let col = "app.bsky.feed.post";
    let body =
client.get("https://bsky.social/xrpc/com.atproto.repo.listRe
cords")
        .query(&[("repo", &handle),("collection", &col),
("limit", &"1"),("revert", &"true")])
        .send()
        .await?
        .text()
        .await?;
    println!("{}", body);
    Ok(())
}
```

```
fn c_list_records(_c: &Context) {
    list_records().unwrap();
}

#[tokio::main]
async fn access_token() -> reqwest::Result<()> {
    let file = "/.config/ai/token.toml";
    let mut f = shellexpand::tilde("~").to_string();
    f.push_str(&file);

    let data = Datas::new().unwrap();
    let data = Datas {
        host: data.host,
        handle: data.handle,
        pass: data.pass,
    };
    let url = "https://".to_owned() + &data.host +
&"/xrpc/com.atproto.server.createSession";

    let mut map = HashMap::new();
    map.insert("identifier", &data.handle);
    map.insert("password", &data.pass);

    let client = reqwest::Client::new();
    let res = client
        .post(url)
        .json(&map)
        .send()
        .await?
        .text()
        .await?;

    let json: Token = serde_json::from_str(&res).unwrap();
    let tokens = Tokens {
        did: json.did.to_string(),
        access: json.accessJwt.to_string(),
        refresh: json.refreshJwt.to_string(),
        handle: json.handle.to_string(),
    };

    let toml = toml::to_string(&tokens).unwrap();
    let mut f = fs::File::create(f.clone()).unwrap();
    f.write_all(&toml.as_bytes()).unwrap();

    Ok(())
}

fn c_access_token(_c: &Context) {
    access_token().unwrap();
}
```

mention

mention

いよいよ、blueskyにpostするコマンドを作成します。正確にはmentionです。

今度は、新しいファイルを作成し、そのファイルをsrc/main.rsで読み込む方式で書いてみます。

Cargo.toml

```
[package]
name = "ai"
version = "0.1.0"
edition = "2021"

[dependencies]
seahorse = "*"
reqwest = { version = "*", features = ["blocking", "json"] }
tokio = { version = "1", features = ["full"] }
serde_derive = "1.0"
serde_json = "1.0"
serde = "*"
config = { git = "https://github.com/mehcode/config-rs",
branch = "master" }
shellexpand = "*"
toml = "*"
iso8601-timestamp = "0.2.10"
```

src/data.rs

```
use config::{Config, ConfigError, File};
use serde_derive::{Deserialize, Serialize};

#[derive(Debug, Deserialize)]
#[allow(unused)]
pub struct Data {
    pub host: String,
    pub pass: String,
    pub handle: String,
}

#[derive(Serialize, Deserialize)]
#[allow(non_snake_case)]
pub struct Token {
    pub did: String,
    pub accessJwt: String,
    pub refreshJwt: String,
    pub handle: String,
}

#[derive(Serialize, Deserialize)]
#[allow(non_snake_case)]
pub struct Tokens {
    pub did: String,
    pub access: String,
    pub refresh: String,
    pub handle: String,
}

#[derive(Serialize, Deserialize)]
#[allow(non_snake_case)]
pub struct Labels {
```

```
}

#[derive(Serialize, Deserialize)]
#[allow(non_snake_case)]
pub struct Declaration {
    pub actorType: String,
    pub cid: String,
}

#[derive(Serialize, Deserialize)]
#[allow(non_snake_case)]
pub struct Viewer {
    pub muted: bool,
}

#[derive(Serialize, Deserialize)]
#[allow(non_snake_case)]
pub struct Profile {
    pub did: String,
    pub handle: String,
    pub followsCount: Option<i32>,
    pub followersCount: Option<i32>,
    pub postsCount: i32,
    pub indexedAt: Option<String>,
    pub avatar: Option<String>,
    pub banner: Option<String>,
    pub displayName: Option<String>,
    pub description: Option<String>,
    pub viewer: Viewer,
    pub labels: Labels,
}

impl Data {
    pub fn new() -> Result<Self, ConfigError> {
        let d = shellexpand::tilde("~") +
"/.config/ai/config.toml";
        let s = Config::builder()
            .add_source(File::with_name(&d))

.add_source(config::Environment::with_prefix("APP"))
            .build()?;
        s.try_deserialize()
    }
}

impl Tokens {
    pub fn new() -> Result<Self, ConfigError> {
        let d = shellexpand::tilde("~") +
"/.config/ai/token.toml";
        let s = Config::builder()
            .add_source(File::with_name(&d))

.add_source(config::Environment::with_prefix("APP"))
            .build()?;
        s.try_deserialize()
    }
}

pub fn token_toml(s: &str) -> String {
    let s = String::from(s);
    let tokens = Tokens::new().unwrap();
    let tokens = Tokens {
        did: tokens.did,
        access: tokens.access,
        refresh: tokens.refresh,
        handle: tokens.handle,
```

```
    };
    match &*s {
        "did" => tokens.did,
        "access" => tokens.access,
        "refresh" => tokens.refresh,
        "handle" => tokens.handle,
        _ => s,
    }
}
```

src/profile.rs

```
extern crate reqwest;
use crate::token_toml;

pub async fn get_request(handle: String) -> String {

    let token = token_toml(&"access");
    let url =
"https://bsky.social/xrpc/app.bsky.actor.getProfile".to_owne
d() + &"?actor=" + &handle;
    let client = reqwest::Client::new();
    let res = client
        .get(url)
        .header("Authorization", "Bearer ".to_owned() +
&token)
        .send()
        .await
        .unwrap()
        .text()
        .await
        .unwrap();

    return res
}
```

src/mention.rs

```
extern crate reqwest;
use crate::token_toml;
use serde_json::json;
use iso8601_timestamp::Timestamp;

pub async fn post_request(text: String, at: String, udid:
String, s: i32, e: i32) -> String {

    let token = token_toml(&"access");
    let did = token_toml(&"did");
    let handle = token_toml(&"handle");

    let url =
"https://bsky.social/xrpc/com.atproto.repo.createRecord";
    let col = "app.bsky.feed.post".to_string();

    let d = Timestamp::now_utc();
    let d = d.to_string();

    let post = Some(json!({
        "did": did.to_string(),
        "repo": handle.to_string(),
        "collection": col.to_string(),
        "record": {
            "text": at.to_string() + &" ".to_string() +
&text.to_string(),
            "$type": "app.bsky.feed.post",
            "createdAt": d.to_string(),
```

```
            "facets": [
            {
                "$type": "app.bsky.richtext.facet",
                "index": {
                    "byteEnd": e,
                    "byteStart": s
                },"features": [
                {
                    "did": udid.to_string(),
                    "$type":
"app.bsky.richtext.facet#mention"
                }
                ]
            }
            ]
        },
    }));

    let client = reqwest::Client::new();
    let res = client
        .post(url)
        .json(&post)
        .header("Authorization", "Bearer ".to_owned() +
&token)
        .send()
        .await
        .unwrap()
        .text()
        .await
        .unwrap();

    return res
}
```

src/main.rs

```
pub mod data;
pub mod mention;
pub mod profile;

use seahorse::{App, Command, Context, Flag, FlagType};
use std::env;
use std::fs;
use std::io::Write;
use std::collections::HashMap;

use data::Data as Datas;
use crate::data::Token;
use crate::data::Tokens;
use crate::data::Profile;
use crate::data::token_toml;

fn main() {
    let args: Vec<String> = env::args().collect();
    let app = App::new(env!("CARGO_PKG_NAME"))
        //.action(c_ascii_art)
        .command(
            Command::new("bluesky")
            .alias("b")
            .action(c_list_records),
            )
        .command(
            Command::new("login")
            .alias("l")
            .action(c_access_token),
            )
```

```
        .command(
            Command::new("profile")
            .alias("p")
            .action(c_profile),
            )
        .command(
            Command::new("mention")
            .alias("m")
            .action(c_mention)
            .flag(
                Flag::new("post", FlagType::String)
                .description("post flag\n\t\t\t$ ai m
syui.bsky.social -p text")
                .alias("p"),
                )
            )

        ;
    app.run(args);
}

#[tokio::main]
async fn list_records() -> reqwest::Result<()> {
    let client = reqwest::Client::new();
    let handle= "support.bsky.team";
    let col = "app.bsky.feed.post";
    let body =
client.get("https://bsky.social/xrpc/com.atproto.repo.listRe
cords")
        .query(&[("repo", &handle),("collection", &col),
("limit", &"1"),("revert", &"true")])
        .send()
        .await?
        .text()
        .await?;
    println!("{}", body);
    Ok(())
}

fn c_list_records(_c: &Context) {
    list_records().unwrap();
}

#[tokio::main]
async fn access_token() -> reqwest::Result<()> {
    let file = "/.config/ai/token.toml";
    let mut f = shellexpand::tilde("~").to_string();
    f.push_str(&file);

    let data = Datas::new().unwrap();
    let data = Datas {
        host: data.host,
        handle: data.handle,
        pass: data.pass,
    };
    let url = "https://".to_owned() + &data.host +
&"/xrpc/com.atproto.server.createSession";

    let mut map = HashMap::new();
    map.insert("identifier", &data.handle);
    map.insert("password", &data.pass);
    let client = reqwest::Client::new();
    let res = client
        .post(url)
        .json(&map)
        .send()
```

```
            .await?
            .text()
            .await?;
        let json: Token = serde_json::from_str(&res).unwrap();
        let tokens = Tokens {
            did: json.did.to_string(),
            access: json.accessJwt.to_string(),
            refresh: json.refreshJwt.to_string(),
            handle: json.handle.to_string(),
        };
        let toml = toml::to_string(&tokens).unwrap();
        let mut f = fs::File::create(f.clone()).unwrap();
        f.write_all(&toml.as_bytes()).unwrap();

        Ok(())
    }

    fn c_access_token(_c: &Context) {
        access_token().unwrap();
    }

    fn profile(c: &Context) {
        let m = c.args[0].to_string();
        let h = async {
            let str = profile::get_request(m.to_string()).await;
            println!("{}",str);
        };
        let res =
    tokio::runtime::Runtime::new().unwrap().block_on(h);
        return res
    }

    fn c_profile(c: &Context) {
        access_token().unwrap();
        profile(c);
    }

    fn mention(c: &Context) {
        let m = c.args[0].to_string();
        let h = async {
            let str = profile::get_request(m.to_string()).await;
            println!("{}",str);
            let profile: Profile =
    serde_json::from_str(&str).unwrap();
            let udid = profile.did;
            let handle = profile.handle;
            let at = "@".to_owned() + &handle;
            let e = at.chars().count();
            let s = 0;
            if let Ok(post) = c.string_flag("post") {
                let str =
    mention::post_request(post.to_string(), at.to_string(),
    udid.to_string(), s, e.try_into().unwrap()).await;
                println!("{}",str);
            }
        };
        let res =
    tokio::runtime::Runtime::new().unwrap().block_on(h);
        return res
    }

    fn c_mention(c: &Context) {
        access_token().unwrap();
        mention(c);
    }
```

今回、面倒なので`bsky.social`以外のhostには対応していません。主に`profile.rs`,`mention.rs`です。その辺は注意してください。

src/profile.rs

```
let url =
"https://bsky.social/xrpc/app.bsky.actor.getProfile".to_owne
d() + &"?actor=" + &handle;
```

base64

base64

次は、コマンドオプションで指定した文字をbase64に変換してmentionするコードを書きます。

これでプログラムの完成です。

まずはbase64のパッケージを追加します。

Cargo.toml

```
[package]
name = "ai"
version = "0.1.0"
edition = "2021"

[dependencies]
seahorse = "*"
reqwest = { version = "*", features = ["blocking", "json"] }
tokio = { version = "1", features = ["full"] }
serde_derive = "1.0"
serde_json = "1.0"
serde = "*"
config = { git = "https://github.com/mehcode/config-rs",
branch = "master" }
shellexpand = "*"
toml = "*"
iso8601-timestamp = "0.2.10"
base64 = "*"
```

そして、src/main.rsのmentionのところにdidをbase64に変換するコードを書いていきます。

これらはサブオプションに設定します。

要点をまとめるとこんな感じです。

example

```
.command(
        Command::new("mention")
        .alias("m")
        .action(c_mention)
        .flag(
            Flag::new("base", FlagType::String)
            .description("base flag\n\t\t\t$ ai m
syui.bsky.social -p text -b 123")
            .alias("b"),
            )
        .flag(
            Flag::new("egg", FlagType::Bool)
            .description("egg flag\n\t\t\t$ ai m
syui.bsky.social -e")
            .alias("e"),
            )

let did = token_toml(&"did");
let body = "/egg ".to_owned() + &encode(did.as_bytes());
```

-bで変換する文字列を指定できるようにします。必ず-b "foo bar"というようにダブルクオーテーションで囲ってください。-eでdidを取ってきて自動変換してmentionするようにします。

```
# 指定してた文字列をbase64にしてmentionする
$ ai m yui.syui.ai -b "did:plc:4hqjfn7m6n5hno3doamuhgef"
@yui.syui.ai /egg
ZGlkOnBsYzo0aHFqZm43bTZuNWhubzNkb2FtdWhnZWY=

# 自分のdidをbase64にしてmentionする
$ ai m yui.syui.ai -e
@yui.syui.ai /egg
ZGlkOnBsYzo0aHFqZm43bTZuNWhubzNkb2FtdWhnZWY=
```

では、全部のコードを書いていきます。

src/main.rs

```
pub mod data;
pub mod mention;
pub mod profile;
//pub mod ascii;

use seahorse::{App, Command, Context, Flag, FlagType};
use std::env;
use std::fs;
use std::io::Write;
use std::collections::HashMap;

use data::Data as Datas;
use crate::data::Token;
use crate::data::Tokens;
use crate::data::Profile;
use crate::data::token_toml;
//use crate::ascii::c_ascii;

extern crate base64;
use base64::encode;

fn main() {
    let args: Vec<String> = env::args().collect();
    let app = App::new(env!("CARGO_PKG_NAME"))
        //.action(c_ascii_art)
        .command(
            Command::new("bluesky")
            .alias("b")
            .action(c_list_records),
            )
        .command(
            Command::new("login")
            .alias("l")
            .action(c_access_token),
            )
        .command(
            Command::new("profile")
            .alias("p")
            .action(c_profile),
            )
        .command(
            Command::new("mention")
            .alias("m")
            .action(c_mention)
            .flag(
                Flag::new("post", FlagType::String)
                .description("post flag\n\t\t\t$ ai m
```

```
syui.bsky.social -p text")
                .alias("p"),
                )
            .flag(
                Flag::new("base", FlagType::String)
                .description("base flag\n\t\t\t$ ai m
syui.bsky.social -p text -b 123")
                .alias("b"),
                )
            .flag(
                Flag::new("egg", FlagType::Bool)
                .description("egg flag\n\t\t\t$ ai m
syui.bsky.social -e")
                .alias("e"),
                )
            )

        ;
    app.run(args);
}

#[tokio::main]
async fn list_records() -> reqwest::Result<()> {
    let client = reqwest::Client::new();
    let handle= "support.bsky.team";
    let col = "app.bsky.feed.post";
    let body =
client.get("https://bsky.social/xrpc/com.atproto.repo.listRe
cords")
        .query(&[("repo", &handle),("collection", &col),
("limit", &"1"),("revert", &"true")])
        .send()
        .await?
        .text()
        .await?;
    println!("{}", body);
    Ok(())
}

fn c_list_records(_c: &Context) {
    list_records().unwrap();
}

#[tokio::main]
async fn access_token() -> reqwest::Result<()> {
    let file = "/.config/ai/token.toml";
    let mut f = shellexpand::tilde("~").to_string();
    f.push_str(&file);

    let data = Datas::new().unwrap();
    let data = Datas {
        host: data.host,
        handle: data.handle,
        pass: data.pass,
    };
    let url = "https://".to_owned() + &data.host +
&"/xrpc/com.atproto.server.createSession";

    let mut map = HashMap::new();
    map.insert("identifier", &data.handle);
    map.insert("password", &data.pass);
    let client = reqwest::Client::new();
    let res = client
        .post(url)
        .json(&map)
        .send()
```

```
            .await?
            .text()
            .await?;
        let json: Token = serde_json::from_str(&res).unwrap();
        let tokens = Tokens {
            did: json.did.to_string(),
            access: json.accessJwt.to_string(),
            refresh: json.refreshJwt.to_string(),
            handle: json.handle.to_string(),
        };
        let toml = toml::to_string(&tokens).unwrap();
        let mut f = fs::File::create(f.clone()).unwrap();
        f.write_all(&toml.as_bytes()).unwrap();

        Ok(())
    }

    fn c_access_token(_c: &Context) {
        access_token().unwrap();
    }

    fn profile(c: &Context) {
        let m = c.args[0].to_string();
        let h = async {
            let str = profile::get_request(m.to_string()).await;
            println!("{}",str);
        };
        let res =
    tokio::runtime::Runtime::new().unwrap().block_on(h);
        return res
    }

    fn c_profile(c: &Context) {
        access_token().unwrap();
        profile(c);
    }

    fn mention(c: &Context) {
        let m = c.args[0].to_string();
        let h = async {
            let str = profile::get_request(m.to_string()).await;
            let profile: Profile =
    serde_json::from_str(&str).unwrap();
            let udid = profile.did;
            let handle = profile.handle;
            let at = "@".to_owned() + &handle;
            let e = at.chars().count();
            let s = 0;
            if let Ok(base) = c.string_flag("base") {
                let body = "/egg ".to_owned() +
    &encode(base.as_bytes());
                let str =
    mention::post_request(body.to_string(), at.to_string(),
    udid.to_string(), s, e.try_into().unwrap()).await;
                println!("{}",str);
            }
            if let Ok(post) = c.string_flag("post") {

                let str =
    mention::post_request(post.to_string(), at.to_string(),
    udid.to_string(), s, e.try_into().unwrap()).await;
                println!("{}",str);
            }
            if c.bool_flag("egg") {
                let did = token_toml(&"did");
                let body = "/egg ".to_owned() +
```

```
&encode(did.as_bytes());
            println!("{}", body);
            let str =
mention::post_request(body.to_string(), at.to_string(),
udid.to_string(), s, e.try_into().unwrap()).await;
            println!("{}",str);
        }
    };
    let res =
tokio::runtime::Runtime::new().unwrap().block_on(h);
    return res
}

fn c_mention(c: &Context) {
    access_token().unwrap();
    mention(c);
}

//fn c_ascii_art(_c: &Context) {
//    c_ascii();
//}
```

```
cargo build
```

できました。

これでmentionを`yui.syui.ai`に指定して、`-e`のオプションを使うと、自分のdidをbase64に自動変換して送ってくれます。

```
./target/debug/ai m yui.syui.ai -e
```

しかし、これではコマンドが実行しづらい。

このコマンドをどこにいても実行できるよう、binary、つまり、`cargo build`すると`./target/debug/ai`を`$PATH`に置いてみます。

linux

```
$ echo $PATH|tr : '\n'
/usr/bin
/usr/local/bin

$ sudo cp -rf ./target/debug/ai /usr/local/bin/
$ ai -h

Name:
        ai
Flags:
        -h, --help : Show help
Commands:
        b, bluesky :
        l, login   :
        p, profile :
        m, mention :
```

windows

```
$ENV:Path.Split(";")
C:\Users\syui\scoop\apps\rust\current\bin

cp ~/scoop/rust/current/bin/
ai -h
```

こんな感じで`rust`で自分のコマンドを作って遊んでみましょう。

end

end

最後に文章でも書いて終わりにしたいと思います。

続けることは尊い

最初から何でもできるということはありません。

できなくても、わからなくても、続けることで、人は成長します。

ただし、続けることは簡単ではありません。

「1年続いたよ。やったね」

「.....」

もしかしたら、誰も何も言ってくれないかもしれません。

「2年続いたよ。がんばったんだ」

「.....」

誰も褒めてくれないかもしれません。

「3年続いたよ。大変だった」

「.....」

「...5年続いたよ。つらいことも、悲しいことも、あったよ」

「.....」

それでも、あなたは、続けることができますか?

続けることができなくてもいい。

ただ、それでも続けられることは尊い。

もしよかったら、頑張ってみてください。

この文章が少しでも勇気を与えられることを願って。