PlantUMLによる設計。

目次

PlantUMLによる設計を簡単な例題をもとに説明します。

はじめに

大雑把な目標は、UMLでどんなことが表現できるか、実際に簡単なUMLを書きながら説明することです。
用語の内容やPlantUMLの詳細な文法にはあまり触れません。適宜、適当なURLを示すことで代替します。

さて、これから作るのは、いわば「なんちゃってツイッター」です。
要件は以下の通り。

1. ユーザは自由に自分のアカウントを作れます。
2. ユーザは自分のアカウントを削除できます。
3. ユーザは相手のアカウントがわかっていれば、フォローとして登録できます。
4. ユーザは、フォローが送信したメッセージを一覧表示できます。
5. ユーザはメッセージを送信できます（そのユーザをフォローとして登録した人はメッセージを閲覧できます）。
6. ユーザの中には管理者がいます。
7. 管理者は、一般ユーザを利用停止したり削除したりできます。
8. 管理者はユーザを管理者にしたり、管理者をユーザにもどしたりできます。

また、最初の管理者はデータベースに直接用意することにしましょう。
まったく実用にはほど遠いですが、これで良しとしましょう。してください。

開発環境

開発環境は、というか、UMLを書くためのマシンは、どうしてもWindowsが中心ならざるを得ません。
WindowsにおけるUMLの記述方法はいくつかあります。

- astahなどを使う。
- ドロー系ソフトを使う
  - Excel
  - PowerPoint
- PlantUMLを使う

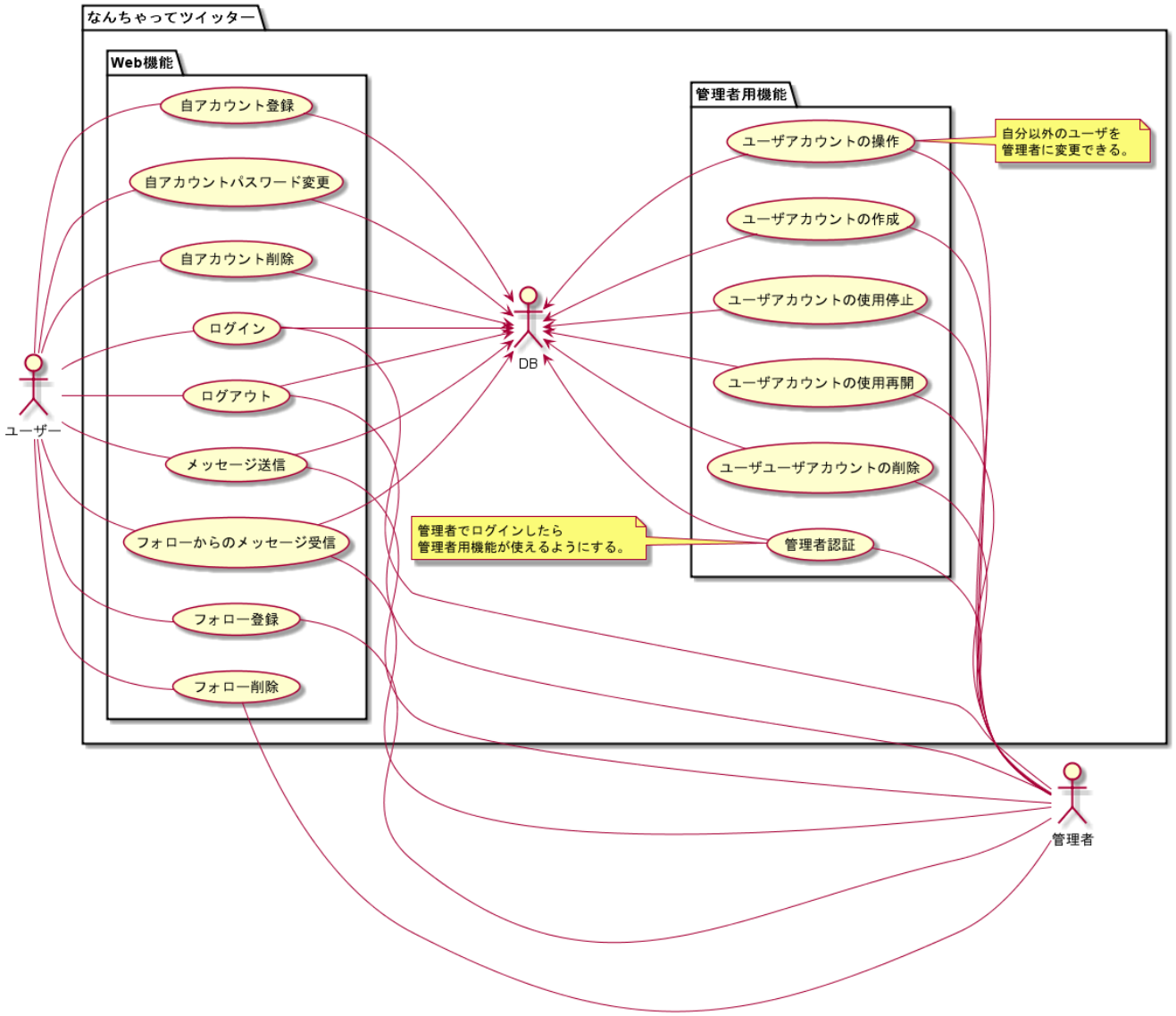
PlantUMLを使えるようにする。

WindowsでPlantUMLを使えるようにするには...

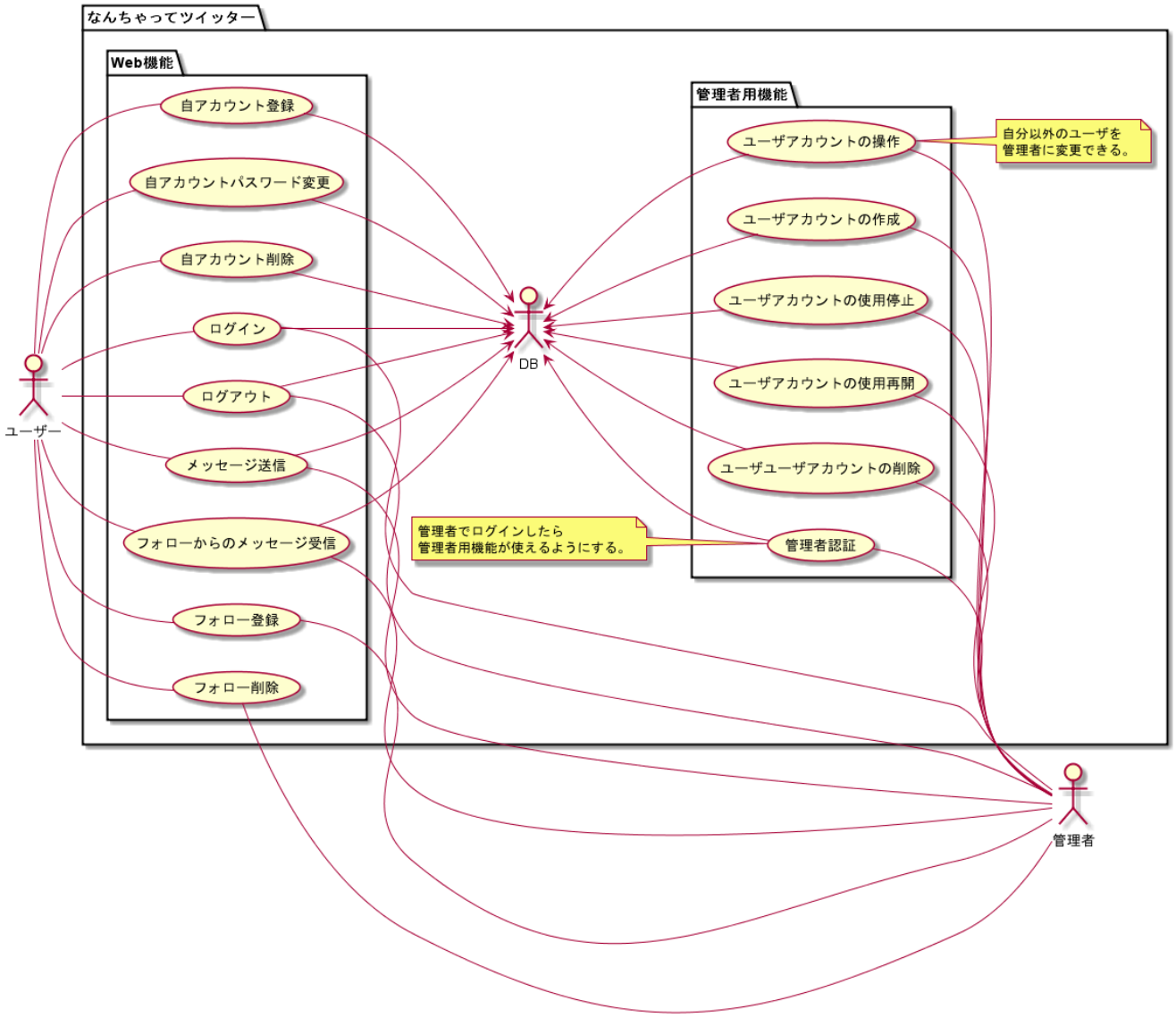
ユースケース

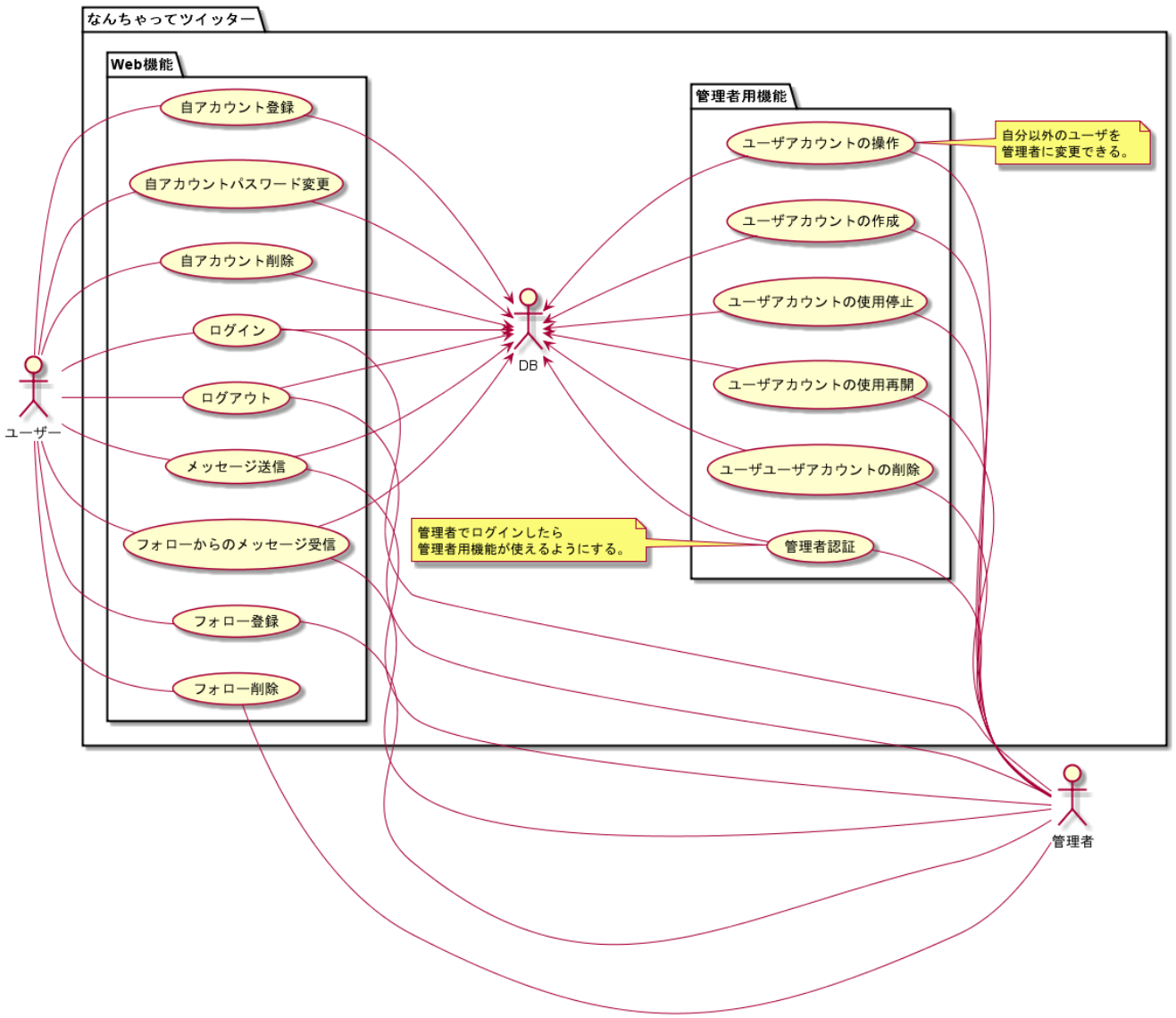
ユースケースとは、ひとことでいうと表現力豊かな箇条書きです。

ユースケースサンプル


なんちゃってツイッター
Web機能
自アカウント登録
自アカウントパスワード変更
自アカウント削除
ログイン
ログアウト
メッセージ送信
フォローからのメッセージ受信
フォロー登録
フォロー削除
管理者用機能
ユーザアカウントの操作
ユーザアカウントの作成
ユーザアカウントの使用停止
ユーザアカウントの使用再開
ユーザユーザアカウントの削除
管理者認証
自分以外のユーザを
管理者に変更できる。
管理者でログインしたら
管理者用機能が使えるようにする。
DB
ユーザー
管理者

ユースケースサンプル


なんちゃってツイッター
Web機能
自アカウント登録
自アカウントパスワード変更
自アカウント削除
ログイン
ログアウト
メッセージ送信
フォローからのメッセージ受信
フォロー登録
フォロー削除
管理者用機能
ユーザアカウントの操作
自分以外のユーザを
管理者に変更できる。
ユーザアカウントの作成
ユーザアカウントの使用停止
ユーザアカウントの使用再開
ユーザユーザアカウントの削除
管理者認証
管理者でログインしたら
管理者用機能が使えるようにする。
DB
ユーザー
管理者




```
@startuml
title ユースケースサンプル
left to right direction

actor :ユーザー :as user
actor :管理者: as admin

package "なんちゃってツイッター" {
  /'
    一般用の画面。
    PCまたは携帯端末のブラウザからアクセスする。
  '/
  package "Web機能" {
    user --(自アカウント登録)
    user --(ログイン)
    user --(自アカウントパスワード変更)
    user --(自アカウント削除)
    user --(フォロー登録)
```

```
    user --(フォロー削除)
    user --(メッセージ送信)
    user --(フォローからのメッセージ受信)
    user --(ログアウト)
  }

  /'
    管理者用の画面。
  '/
  package "管理者用機能" {
    (ログイン)--admin
    (ログアウト)--admin
    (管理者認証)--admin
    (ユーザアカウントの作成)--admin
    (ユーザアカウントの操作)--admin
    (ユーザアカウントの使用停止) -- admin
    (ユーザアカウントの使用再開) -- admin
    (ユーザユーザアカウントの削除) -- admin
    (フォロー登録) -- admin
    (フォロー削除) -- admin
    (メッセージ送信)--admin
    (フォローからのメッセージ受信)--admin
  }
  note right of ユーザアカウントの操作: 自分以外のユーザを\n管理者に変更できる。
  note left of 管理者認証: 管理者でログインしたら\n管理者用機能が使えるようにする。

  database :DB : as DB

  (自アカウント登録)->DB
  (自アカウントパスワード変更)->DB
  (ログイン)->DB
  (ログアウト)->DB
  (管理者認証)->DB
  (自アカウント削除)->DB
  (メッセージ送信)->DB
  (フォローからのメッセージ受信)->DB
  DB<-(ユーザアカウントの作成)
  DB<-(ユーザアカウントの使用停止)
  DB<-(ユーザアカウントの使用再開)
  DB<-(ユーザユーザアカウントの削除)
  DB<-(ユーザアカウントの操作)
}

@enduml
```

画面設計

画面設計は意外と工数のかかる作業です。
本来のUMLには画面設計にあたるドキュメントはありませんが、簡単な業務システムならば、PlantUMLで設計することが可能です。

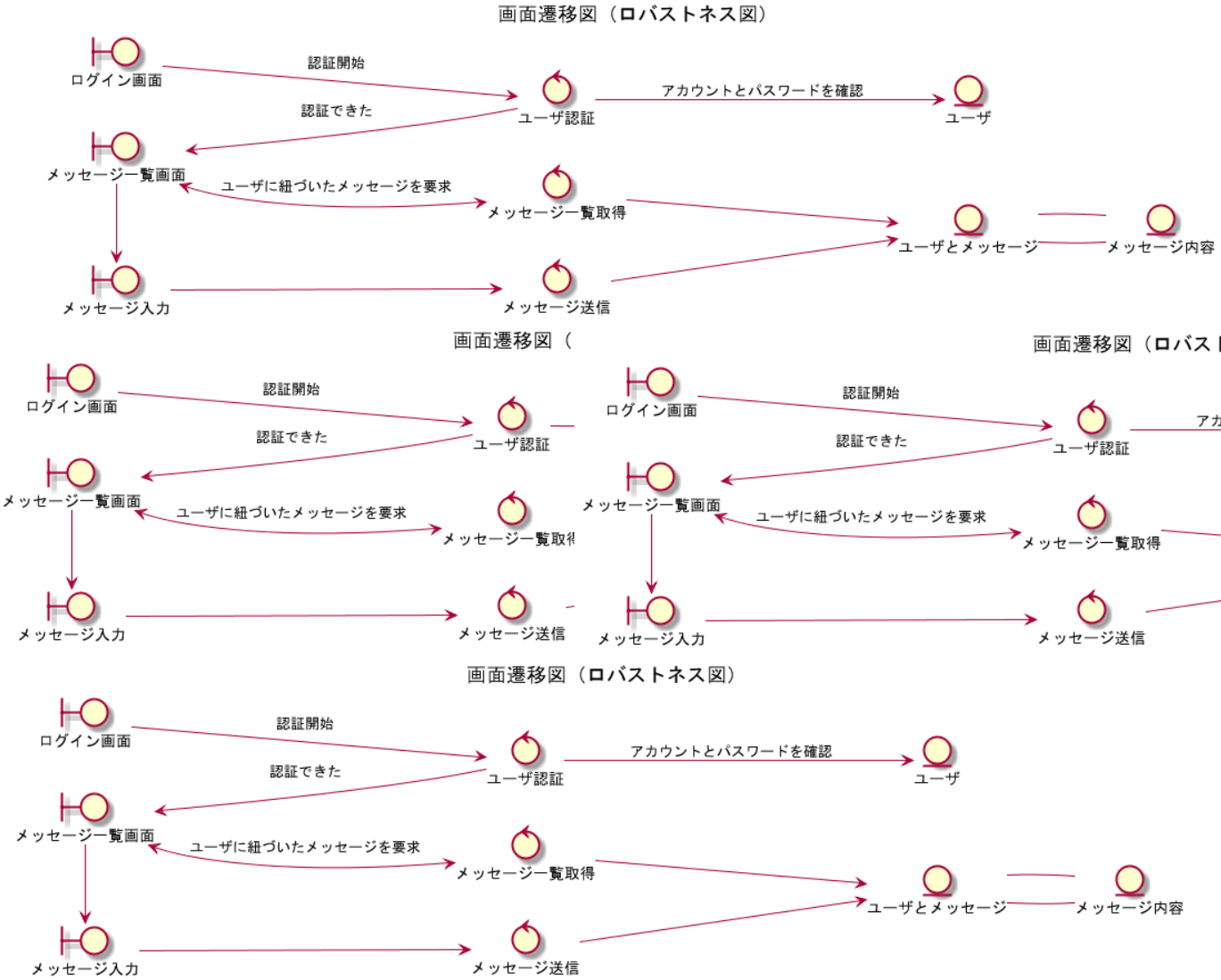
たとえば、以下のようなログイン画面は...

このようなテキストで表現できます。

```
@startuml
salt
{+
  Login    | "MyName   "
  Password | "****     "
  [Cancel] | [  OK   ]
}
@enduml
```

画面遷移

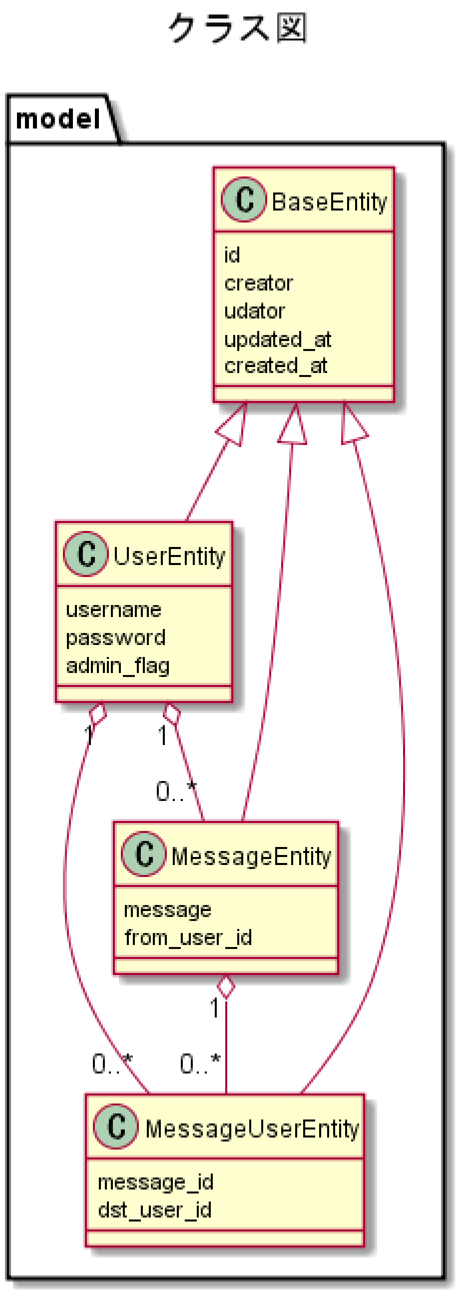
それぞれの見た目を画面設計で作った後、または作ると同時に、画面遷移も決めていかなけらばなりません。
これをUMLで表現する場合は、ロバストネス図というものを作ります。

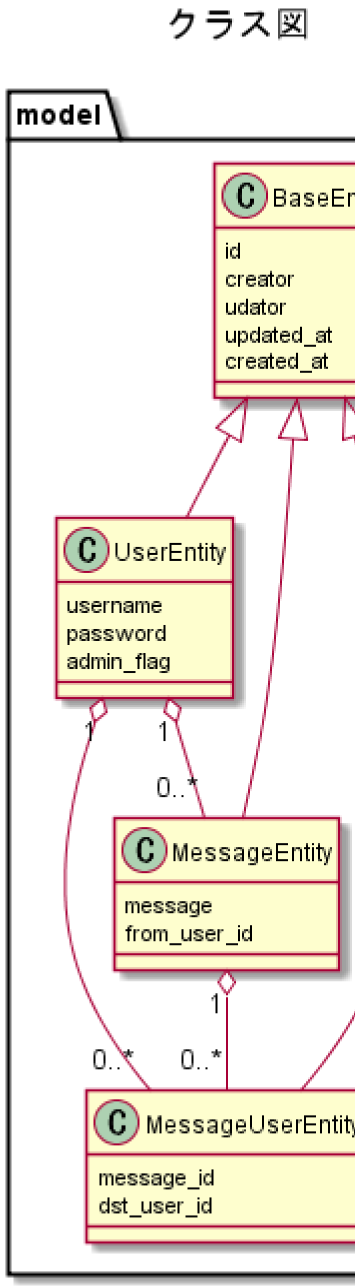


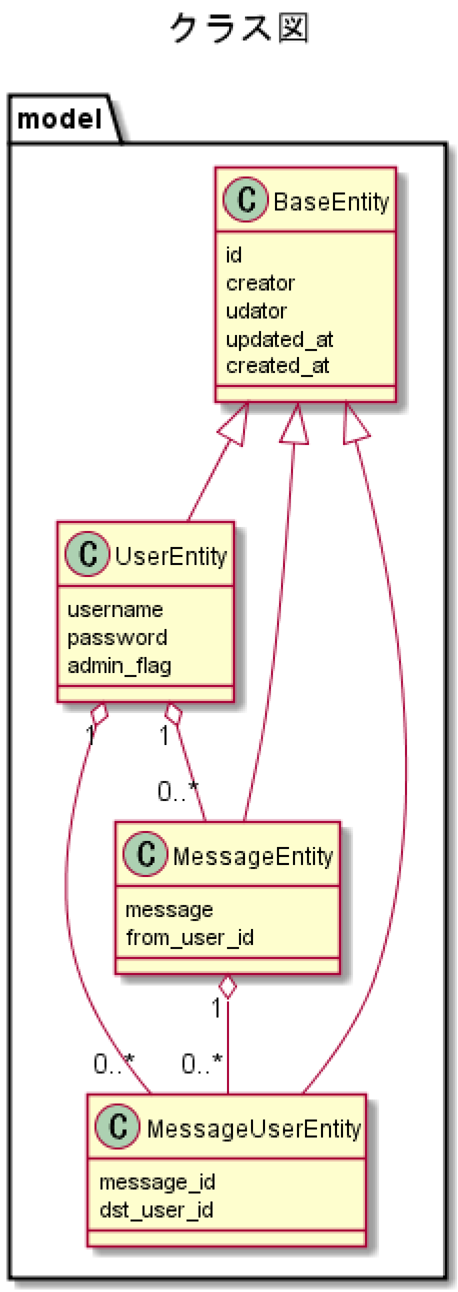
```
@startuml
title 画面遷移図（ロバストネス図）
left to right direction

boundary ログイン画面 as login
boundary メッセージ一覧画面 as messageTable
boundary メッセージ入力 as messageInput

control ユーザ認証 as userCredential
control メッセージ一覧取得 as messageGet
control メッセージ送信 as messageSend

entity ユーザ as users
entity メッセージ内容 as messages
entity ユーザとメッセージ as usermessages


login --> userCredential : 認証開始
userCredential --> users : アカウントとパスワードを確認
messageTable <-- userCredential : 認証できた
messageTable <--> messageGet : ユーザに紐づいたメッセージを要求
messageGet --> usermessages
usermessages -- messages

messageTable -> messageInput
messageInput --> messageSend
messageSend --> usermessages
usermessages -- messages

@enduml
```

クラス図

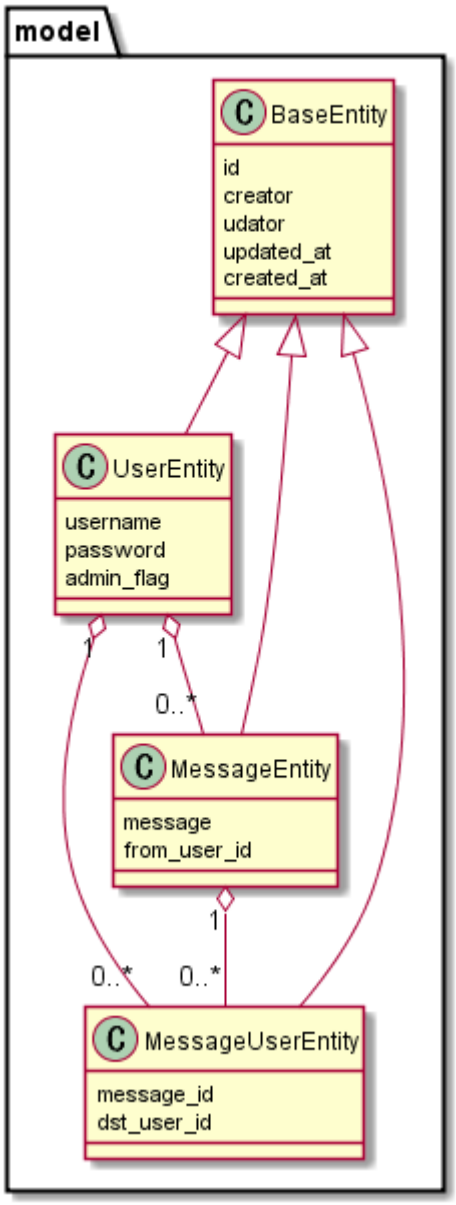
外側を大まかに決めたら、次は中身を決めていきます。


クラス図
model
BaseEntity
id
creator
udator
updated_at
created_at
UserEntity
username
password
admin_flag
MessageEntity
message
from_user_id
MessageUserEntity
message_id
dst_user_id
1
1
0..*
1
0..*
0..*


クラス図
model
id
creator
udator
updated_at
created_at
UserEntity
username
password
admin_flag
MessageEntity
message
from_user_id
message_id
dst_user_id
1
1
0..*
1
0..*
0..*


クラス図
model
BaseEntity
id
creator
udator
updated_at
created_at
UserEntity
username
password
admin_flag
MessageEntity
message
from_user_id
MessageUserEntity
message_id
dst_user_id
1
1
0..*
1
0..*
0..*

クラス図



終わりに

PlantUMLの最大の利点は、ドキュメント更新の速さです。
その秘密は、実は、柔軟性のなさにあります。繰り返します。柔軟性のなさです。
この線をあと1ドットずらしてくれ、ここのフォントを小さくしてくれなど、見た目にしか影響がなくロジックの表現はいっさい貢献しない部分に工数を割く必要はないのです。