コーディングを効率化する
コミュニケーションツールの提案

関西学院大学理工学部
情報科学科 西谷研究室
27014581
オムリ ユーセフ

2020 年 3 月

概 要

大学では，レクチャーやアサイメントなどでコーディングをする際にはプログラミングスキルが一番重要である．しかし，会社では一人でコーディングを行う事などほとんどない．それよりも，"チームワーク"や"共同作業"が重要となる．そして，それらを行う上でで必要不可欠になる要素が"コミュニケーション"だ．

本研究を進める前に，西谷研究室では，コミュニケーションツールとしてメールを使い，Jekyll と GitHub を融合して学習物や成果物を公開･共有していた．しかし，この成果物や学習物，コミュニケーションは決してリアルタイムのものと言えない．そこで，VSCode の Live Share と Discord を利用した新たな開発環境を提案した．

Live Share を利用する事により，リアルタイムかつ遠隔で共同作業できる．また，コミュニケーションツールに Discord を採用した事により，メールより早く，頻繁にリアルタイムでコミュニケートする事が可能となった．それに加え，Discord に Bot を導入する事により，学習時間をいつでも記録できる．

一方で，Discord の導入によるコミュニケーションの加速によって，"けじめの曖昧性"や"雑さ"といったデメリットが出てくる．しかし，これらの問題は，各個人が意識して利用することで解消できるのでは無いかと考える．それよりも，会話の頻度を増やし，より多くの情報交換を重要視すべきである．

目 次

第 1 章　序論 **3**
1.1　研究の背景 3
1.2　研究の目標 3

第 2 章　手法 **4**
2.1　テスト駆動開発 (TDD) 4
2.2　GitHub と Jekyll 4
2.3　Visual Studio Code 5
2.4　Discord 6
2.4.1　サーバー内のロールについて 8
2.4.2　Discord 内でのサーバーについて 10

第 3 章　メールから Discord への移行 **12**

第 4 章　Discord の Bot について **13**
4.1　Bot の作成・導入方法 13
4.2　Bot の持つ機能 14
4.3　Bot の設計 (学習時間記録) 15
4.4　Bot のシステムの振る舞い 17

第 5 章　試験導入 **19**

第 6 章　考察 **21**

第 7 章　総括 **22**

図 目 次

2.1 Live Share の画面. 6
2.2 メッセージのピン留め機能. 7
2.3 各メンバーのロール. 9
2.4 Discord のサーバについて. 10

4.1 Bot の情報画面. 13
4.2 Bot の URL. 14
4.3 Bot の構造. 15
4.4 Bot のシステム. 17

5.1 テストの流れ. 19
5.2 Bot の受け答え. 20
5.3 Result.text の中身. 20

第1章　序論

1.1　研究の背景

西谷研の卒業研究においては，コードを書くことが要求される．私の場合は，個人が使うメモソフト my_help から情報を抽出して，研究室全体で共有するためのシステム構築が当初の開発目標であった．そこで，my_help のテキストフォーマットである Org から静的 web サイト Jekyll への変換ツールの作成を試みた．開発の中でも特に，Org-mode 上での日付，画像，強調，アンダーライン，テーブル表記の変換を優先して行った．

中間発表の直後，教授からテスト駆動開発 (TDD) を紹介された．そのレクチャを通して，私は，ただひたすらにコーデイングをするのではなく，開発環境つまりは土台を整え，その上で作業をすることが重要であることに気が付いた．さらに，先輩や，教授の手を借りながら開発を進めていったが，双方の日程が合わない事が多々あった．私は，もっと手軽で円滑にコミュニケーションを取りながら，離れていても双方の都合の良い時にディスカッションやコーデイングができる様な開発環境があればいいのではないかと考えた．

1.2　研究の目標

先述した環境に必要な条件をまとめると下記になる．

- 遠隔でも作業やコーディングができる
- 音声通話やチャットでリアルタイムコミュニケーションが取れる

それらを満たすのが Visual Studio Code の Live Share 機能と Discord を組み合わせた使い方だと考えた．元々，Jekyll で Blog として公開・共有する目的の一つに，作業を"記録" として残すという要素があった．そのため，遠隔での作業時間や，家での作業時間を記録できるソフトを開発したいと考えた．また，その記録データをファイルに書き込み，過去の作業時間や日付を見直せるようなデータファイルを作りたいと考えた．

以上のことから，先ずは西谷研究室内で環境を整備することを目標とし，そのために必要な項目を下記とした．

- 自身の開発環境の整備 (VSCode, Discord)
- Discord に記録用の Bot を導入
- VSCode の初期セットアップファイルの作成

第2章　手法

2.1　テスト駆動開発 (TDD)

レクチャーを受けたテスト駆動開発 (以下 TDD と記す) とは，その名の通りテストをしながら開発を行っていくものである．TDD の目指すゴールは「動作するきれいなコード」である．文字通り，まず初めに「動作する」ものを作成し，次に「きれい」にしていく．順序にすると下記になる．

- テストを書く
- コンパイラを通す
- テストを実行し，エラーをみる
- エラーを直し，テストが動く様にする
- 重複を無くし，きれいにする

この開発方法を使うことにより，段階的にプログラムをテストすることができ，スムーズな開発が可能となる．

TDD はアーキテクチャ駆動とは全く逆の発想である [1]．すなわち，アーキテクチャ駆動では，上位者がアーキテクチャをデザインして，上意下達あるいは waterfall 型で下請けに仕様を伝えて，そこでコーディングが進行する．一方，TDD はボトムアップ型である．コーディングをしながら全体のシステムを考えていく必要がある．そうするとコミュニケーションの質が既存のものとは違ってくる．より早く，より頻繁にディスカッションをする必要が出てくる．

2.2　GitHub と Jekyll

このような開発者のコーディングの連携を効率的に進める一つのプラットフォームとして GitHub がある．管理システムに Git を使い，レポジトリを特定の人と共有したり，公開したりできる．GitHub を利用することにより，研究室の仲間内で学習物，成果物の共有が容易となる．また，issue や pull request によって問題の共有や，連携を促進する工夫がなされている．さらに，GutHub には GitHub Pages というホスティングサービス [2] があり，Web サーバーを自分で用意するとなく Web サイトを無料で作成・公開できる．

Github で効率的に Web サイトを作成するために，Jekyll を利用する．Jekyll とはプレーンテキストから静的な Web サイトを生成するツールである．GitHub と Jekyll は連携が

可能であるため，Jekyll で構築したデータを簡単に github.io で公開できる [3]．これにより，学習物を Web ページという見易い形で共有・閲覧できる．

2.3 Visual Studio Code

先述の通り，先進的な開発環境が提供されているが，さらに TDD に馴染んだ開発環境を構築する必要がある．

Visual Studio Code(以下 VSCode) はマイクロソフト社が開発したソースコードエディタである．electron を使って構築されているため，windows, mac, および linux 版でほぼ同じルックスで提供されている．私が以前に使っていた Emacs というエディタと比較してみると表 2.1のようになる

表 2.1: Emacs と VSCode の比較.

	Emacs	VSCode
ビジュアル	CUI だけ	CUI と GUI の融合
操作性	キーバインドを覚える必要がある	キーバインドはカスタマイズ可
カスタマイズ性	色などは変えられる	色に加え，アイコンなども変更可
拡張性	lisp による内部拡張が基本	css による内部拡張に加え，外部拡張も可

Emacs ではビジュアルとして，キャラクター表示だけであるのに対して，VSCode はグラフィックスとも融合した表示を提供している．それがそのままインターフェースに結びつき，Emacs がキーバインドだけの操作であるのに対して，VSCode はマウスを使ったポインティングが可能である．操作に慣れてくると編集は，key bind が圧倒的に高速であるが，初心者ではどのように操作していいかわからず途方にくれることが多い．従って，VSCode は初心者から熟練者までをカバーしている．そのほか，カスタマイズ性や拡張性にも VSCode が優れている．

上記に挙げた以外の良い点としては下記になる

- Live Share によるペアプロができる
- ファイルのツリーを GUI や CUI で表示できる
- ターミナルを表示したまま作業ができる (非表示も可)
- コピーとペーストが簡単
- カーソルを自由に使える

VSCode の機能の一つに Live Share がある．この機能は，リアルタイムでコードの共同編集を可能にしてくれる [4].

実際に Live Share を利用している時の画面は図 2.1のようになる．黄色のマークの箇所が相手が作業をしているという印になる．これにより，離れていても遠隔で作業ができ，素早く複数人でコーディングを開始することができる．

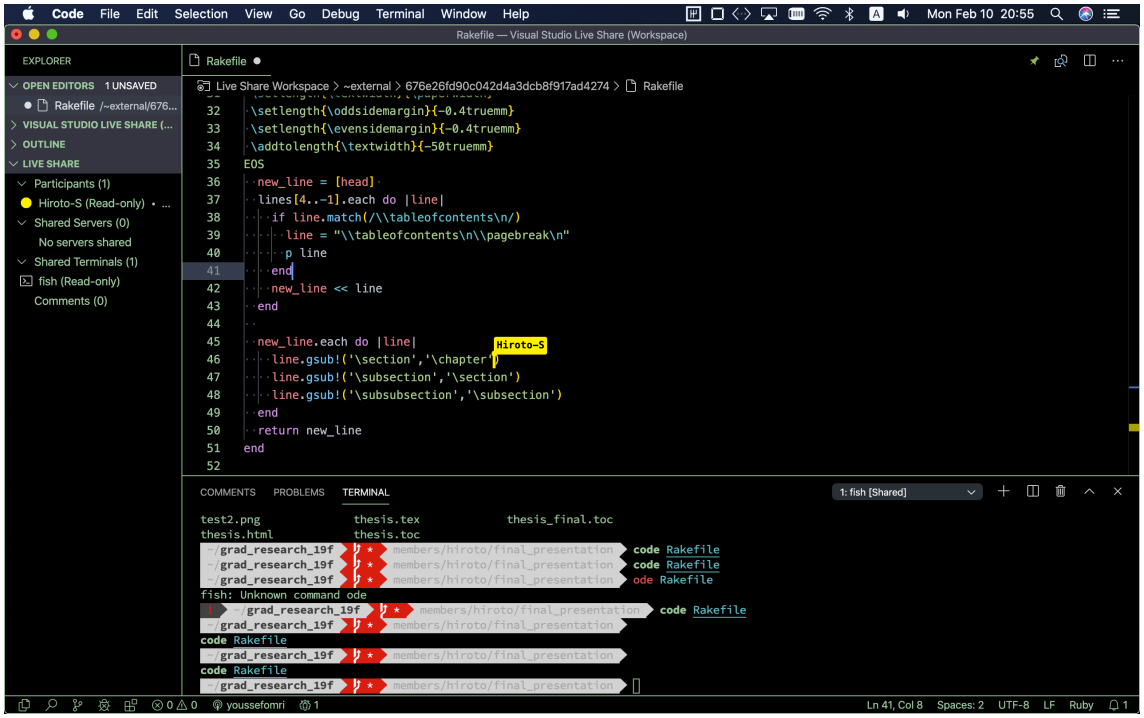

図 2.1: Live Share の画面.

2.4 Discord

遠隔で複数人で作業をしている時に必要不可欠な機能が先述した通り，早く，頻繁なコミュニケーションである．コミュニケーションにもいくつかの方法がある．例えば，通話やメール，チャットなどがある．その中でも，通話はメールやチャットでコミュニケートするより遥かに早い．それらを可能にするコミュニケーションツールの 1 つに Discord がある．Discord は Windows、macOS、Linux、Android、iOS、Web ブラウザで動作するソフトウェアである．他の有名なツールとしては，Skype, Slack, Line などがあるので，なぜ Discord を選んだかの比較を以下に記す．

- Discord vs Skype
  - Skype は他人と通話をする時に必ず呼び出しを必要とする．また，全体的な動作が重たい．
  - それに対し Discord は，呼び出しを必要とせず用意されたボイスチャンネルに入るだけで他人との通話が可能となる．
- Discord vs Slack
  - Slack はワークスペースごとにアカウントを作り，コミュニケーションを取るのでどちらかと言えばビジネス向けのツールである．
  - それに対し Discord は，アカウント一つで全てのチャンネルやサーバーに入れるので Line のような感覚で 1 対全体，もしくは 1 対 1 のコミュケーションが取れる．加えて，Slack で有料で提供しているサービスを全て無料で利用できる．

- Discord vs Line
  - Line はオンラインやオフラインといったステータス表示がない．Line の場合，複数人で話すと会話は上へ流れ，見逃すとスクロールで戻らなければならない．また，友達を会話に入れるには，一度フレンド登録をして直接招待しなければならない．どちらかというとプライベート向け．
  - それに対し Discord は，オンライン，オフラインに加え，様々なカスタムステータス（自分でステータスを入力できる) 機能がある．複数人で話しても，未読の位置から画面が始まる．また，基本複数人での会話を想定しているので，アカウントさえあればフレンド登録をしていなくとも，サーバーに居れば会話を始められる．

その他の Discord の利点をまとめて紹介する．

- サーバーやチャンネルの操作や閲覧権限をロールごとに割り当てられるので，セキュリティー面も抜群に良い．
- Bot を導入することにより様々な役目を自動でこなしてくれる．
- 特定のキーワードや，日時指定で検索をかけられる．
- ユーザーネームは登録したものとは別に，ニックネームとしてサーバー別に変更可能
- 各個人に対してのボリュームコントロールがある．
- "push to talk"により，ボタンを押している時だけ喋れる．
- メッセージの pin 留め機能があり，重要なメッセージを図 2.2のように即座に見れる．

図 2.2: メッセージのピン留め機能．

次に Discrod 内の操作性について記述する．

2.4.1　サーバー内のロールについて

Discord は各メンバーにロールを割り振ることがきる．ロールと辞書を引くと,「役目」や「役割」と出てくるが，実際に Discord のロールも同じようなものである．サーバーを立てた者は管理者となり，サーバーを管理する．また，メンバー内の人にサーバーの管理権限を委ねる事もできる．権限には下記のようないくつもの種類がある．

- チャンネルの管理権限
  - 作成/削除/編集
- メンバーの管理権限
  - ニックネームの変更
  - 招待/キック/BAN
  - メンバーの移動
- ロールの管理権限
  - 作成/削除/編集
- テキスト管理権限
  - メッセージの管理
  - ファイルの添付
  - 履歴の閲覧
  - 絵文字の管理
- 音声の管理権限
  - 接続
  - 発言
  - ミュート/スピーカーのミュート

また，ロールごとにメッセージを送る事もできるので，1 つのロールを 1 つのグループの様に使う事もできる．

西谷研究室での各メンバーのロールは図 2.3のようになっている．

左の列は設定の項目である．真ん中の列は各メンバーのニックネームとそれぞれの UserID が記されている．右の列がロールを示しており，色によってグループが違う．以下に色別のグループを記す．

- 赤色：先生
- 青色：物理組

図 2.3: 各メンバーのロール.

- 緑色：プログラミング組
- 紫色：大学院生
- オレンジ色：サーバーの管理者
- 灰色：BOT

ロールは複数割り当てる事ができ，omri と書かれたニックネームの人は管理者とプログラミング組が割り振られている.
通常はロールを割り振られると以下の項目の権限がデフォルトで割り振られる.

- 招待を作成
- 個人のニックネームを変更
- テキストチャンネル
  - 閲覧
  - メッセージ送信
  - 埋め込みリンク送信
  - ファイルを添付

- 履歴を読む
- 絵文字の使用
- リアクションの追加

- ボイスチャンネル
  - 接続
  - 発言
  - 音声検出
  - Go Live(配信)

上記以外の，サーバーに関しての権限やメンバーに関しての権限は割り振られない．そのため，無駄な手間を必要とせず，メンバーが参加すると，すぐに通話やチャットを楽しめる．

2.4.2 Discord 内でのサーバーについて

Discord のサーバーは個人でサーバーを用意することなく簡単に立てることができることが魅力の一つである．Discord にログインをすると図 2.4のようにサーバーを新規作成するか，参加するかが出てくる．

図 2.4: Discord のサーバについて．

サーバーを立てるには左側の新規作成を押し，サーバー名と地域を入力/選択して完了する．他の誰かが立てたサーバーに入るには右側の参加を押し招待 URL を入力して完了する．

第3章　メールからDiscordへの移行

我々の研究室でDiscordを導入するまでは，メールを使い報告や連絡をしていた．研究室以外でのコミュニケーションではメールは馴染みがある．メールは礼節や相手との距離感を保てるので，研究室以外では適したツールだと考える．それに対し研究室内では，親しくなった研究室の仲間や教授にメールで堅い文章を使用し，余白までつけて習慣化した決まり文句を打つ必要はないと感じた．それよりも，素早く，的確に，伝えたい内容を送る方がよほど効率的だと考えた．

具体例として，学生の一人がメールを使い教授に連絡をとっても，教授が学生の通知をオンにしていないとすぐに内容を確認することは困難である．さらには，教授は研究室外の人たちとも多くのメールのやり取りをしており，積もりに積もったメールの山の中からゼミ生だけのメールを瞬時に探し出し返答するには，重要フォルダーに入れたり，お気に入りに入れたりと手間がかかる．しかしDiscordの場合には，基本的にメッセージのpushのみなので，メールのようにpullしてサーバーからメールを引っ張ってくる事なしにメッセージを受信し返答できる．ゆえに，研究室のような狭い領域のコミュニケーションであればDiscrodの方圧倒的に早い．

表3.1にメールとDiscordの違いをまとめた．

表 3.1: メールとDiscordの比較．

	メール	Discord
挨拶	毎回，決まり文句から始まる	決まり文句なし
対応速度	遅い	早い
対話形式	1対1 (メーリングリストは例外)	1対複数人
送受信形式	pullとpush	pushのみ

第4章　DiscordのBotについて

次にDiscordのBot機能について紹介する．DiscordにはBotというアプリケーションを導入することができる．Botは Discordの拡張機能であり，入れることにより Discordにさらなる価値を追加できる．

4.1　Botの作成・導入方法

先ず初めに，Discordの developer用サイト [4] で自分のアカウントにログインし Botを作成する．作成すると図 4.1のようになる．

作成した Botをサーバーへ導入する順序を下記に記す．

- 画面内の OAuth2へ移動する
- Botというボタンを押す
- Botのボタンが押されると図 4.2のように URLが出てくるのでそれをコピー．
- コピーした URLへ移動し，自分の立てたサーバーを選択し導入完了．

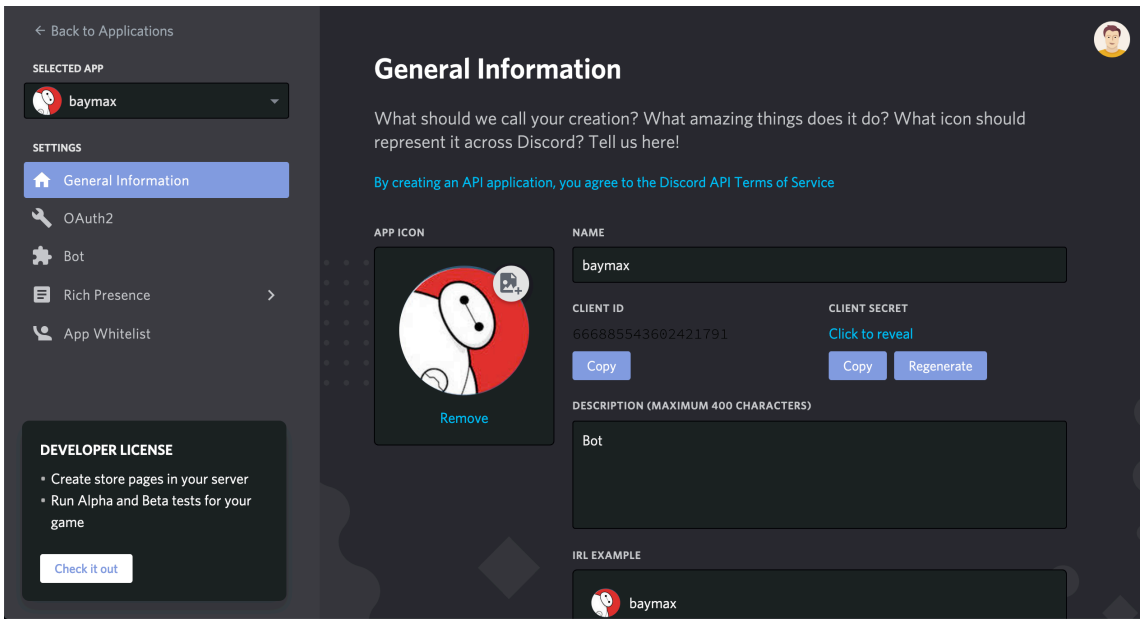

図 4.1: Botの情報画面．

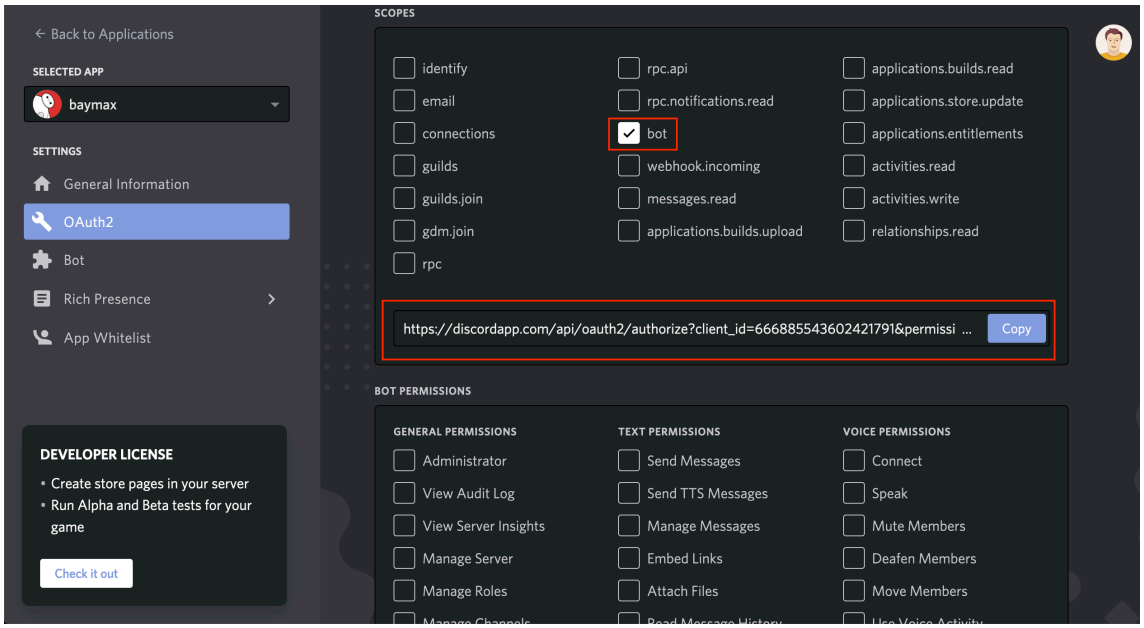

図 4.2: Bot の URL.

4.2 Bot の持つ機能

Discord に導入できる Bot は様々な機能を持っている．以下に機能の一部を記載する．

- ユーザーに関して
  - ニックネームをつける
  - 役職を割り振る
  - ユーザーのキックや BAN
- チャンネルに関して
  - サーバーに参加した日付の取得
  - メンバーリストの取得
  - メッセージを送信
  - メッセージのピン留め
  - メッセージの削除
  - ファイルの送信
  - 音楽を流す
  - チャンネル内のユーザーの移動
- サーバーに関して
  - 役職の管理

        * 追加/削除/編集
    - チャンネルの管理
        * 追加/削除/編集

などがある．

その中でも，メッセージ機能を利用し，ユーザーがアクションを起こすとそれに反応してくる機能．それから，Bot に学習時間の記録を取らせる機能を持たせたいと考えた．

4.3 Bot の設計 (学習時間記録)

続いて実際の Bot の設計について記述する．Discord の Bot は C, Pyhton, Java, Ruby など様々な言語の API が用意されている．西谷研究室で多くの使用経験がある Ruby 言語を用いて Bot を作成した．私が目標とする機能の一つである記録時間取得の構造は図 4.3である．

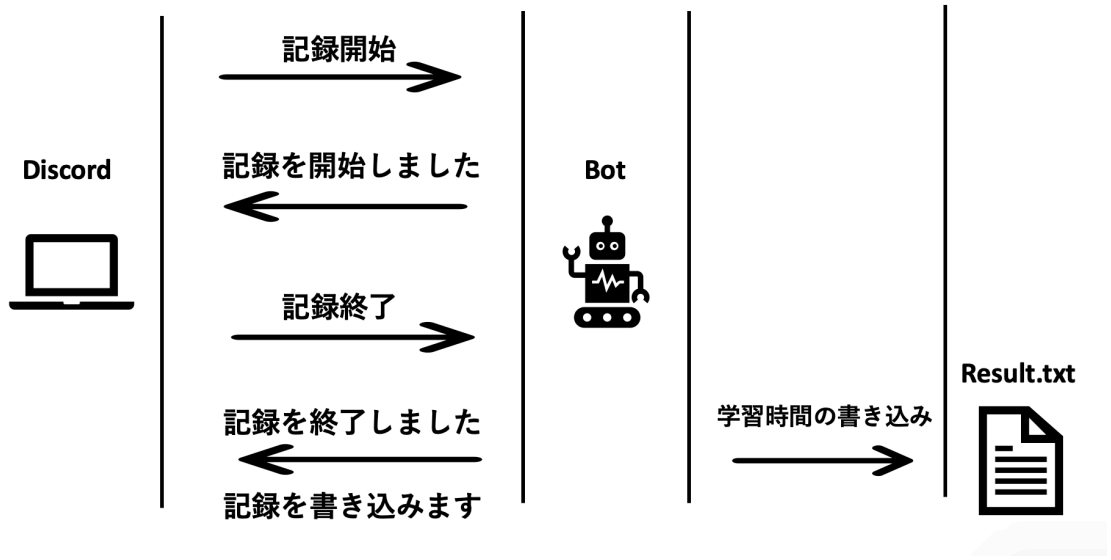

図 4.3: Bot の構造.

研究の目的でも示した通り，記録を取るだけではなく，記録を残す事をゴールとして設計している．

実際の Bot のプログラムは下記となる．

```
1
2  # -*- coding: utf-8 -*-
3  require 'discordrb'
4  require 'date'
5
6
```

```
 7  TOKEN = 'NjY2ODg1NTQzNjAyNDIxNzkx.Xh6yLg.WE-MyAxdqjdcai6-3JgNrDhJc1M'
 8  CLIENT_ID = '666885543602421791'
 9
10  bot = Discordrb::Commands::CommandBot.new token: TOKEN,
11  client_id: CLIENT_ID, prefix: "/"
12
13  bot.message(content:"こんにちは") do |event|
14    event.respond "#{event.user.display_name}さん，こんにちは！"
15  end
16
17  bot.message(content:"start") do |event|
18    $startTime = Time.now
19    event.respond "#{event.user.display_name}さん，記録を開始しました．"
20    event.respond "開始時刻は#{$startTime.hour}時#{$startTime.min}分です．"
21    $st = $startTime.to_i
22  end
23
24  bot.message(content:"fin") do |event|
25    $finTime = Time.now
26    event.respond "#{event.user.display_name}さん，記録を終了しました．"
27    event.respond "終了時刻は#{$finTime.hour}時#{$finTime.min}分です．"
28    $ft = $finTime.to_i
29  end
30
31  bot.message(content:"submit") do |event|
32
33    $resTime = Time.now
34    $rt = $resTime.strftime("%Y-%m-%d-%a %H:%M")
35    $res = $ft - $st
36
37    case
38    when $res < 60
39    then $output = "#{event.user.display_name}さんの学習時間は#{$res}秒です．"
40    when $res >= 60 && $res < 3600
41    then $output =  "#{event.user.display_name}さんの学習時間は#{($res)/60}分#{($res)%60}秒です．"
42    when $res >= 3600
43    then $output = "#{event.user.display_name}さんの学習時間は#{($res)/3600}時間#{($res % 3600)/60}分#{($res % 3600) % 60}秒です．"
44    end
```

```
45
46    File.open("result.text", "a") do |func|
47      func.puts("[#{$rt}]\t#{$output}\n")
48    end
49    event.send_message "お疲れ様です．記録の書き込みが完了しました．"
50  end
51
52  bot.run
```

プログラムの説明は以下の通りである．プログラムの１３行目から１５行目の間は，ユーザーが「こんにちは」とメッセージを送信すると Bot がユーザーのニックネームと共に「○○さん，こんにちは！」と返信するようになっている．

プログラムの１７行目から４４行目まではユーザーの学習時間を取得するようになっており，開始時間と終了時間を取り，その差分をとって学習時間とするようになっている．呼び出し方は「start」で記録開始,「fin」で記録を終了する．

46 行目から 50 行目までで書き込みを実行している．Discord のディレクトリ内に Result.txt を作成し，その中に日付と学習時間を追記するようになっている．

4.4 Bot のシステムの振る舞い

Discord の Bot は言わば，サーバのようなものである．そのため，Bot の管理権限を共有する事ができ，誰しもが必要に応じて Bot を起動できる．実際のシステムの動きを図 4.4に示す．

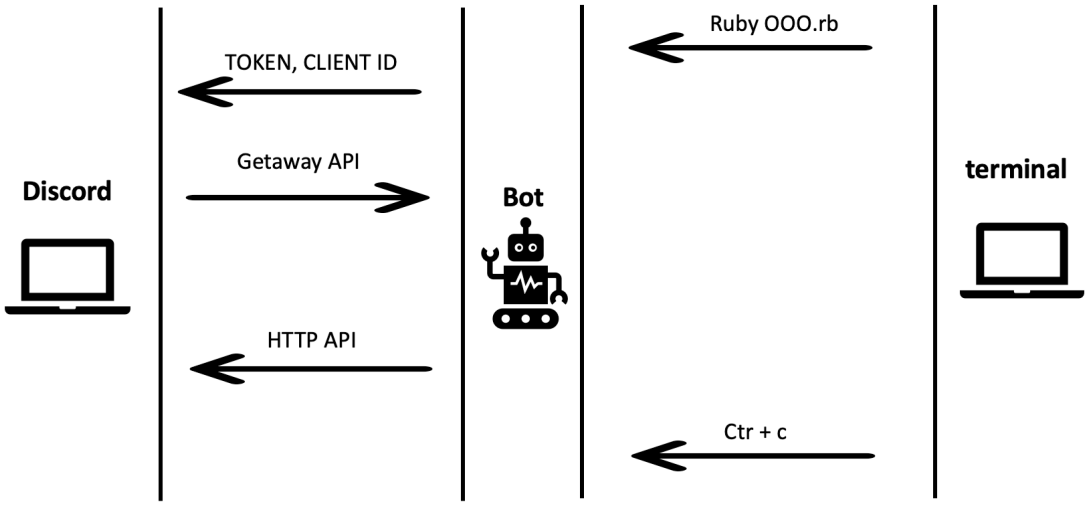

図 4.4: Bot のシステム．

Bot のプログラムはターミナルで ruby ファイルとして保存されている．Bot を起動するには"Ruby ○○.rb"で実行する．実行すると，プログラムに書かれた TOKEN と CLIENT_ID を元に Discord のサーバーを確定する．接続方法の Gateway API の中でも，WebSocket を利用し双方の通信を確立する．Websocket の利点としては，一度コネクションを確立すると，それ以降の通信をそのコネクション上で行うのでわざわざ，繋ぎ直さなくていい点である．そのためシステムの動作が軽くなる．Bot はターミナル上で"ctrl + c"と入力すると強制終了する．

第5章　試験導入

実際に研究室でDiscordを試験導入してみると，皆から使いやすいとの声を得た．教授からもメールでやりとりするよりも，送りたい人に@を付けるだけで送信できたり，今誰がオンラインで，誰がオフラインかを瞬時に認識できてやりとりが早いとの評価を得た．加えて，チャットメッセージ内で検索できたり，重要なメッセージを簡単にピン留めできるので後で見返しやすいとの意見もあった．

実際のやりとりは図5.1ようになる．

図 5.1: テストの流れ．

Discordを導入して良かった点は以下である．

- メールより見易い
- やりとりが早い
- 仲間との距離が近くなるので，メールより堅くなく気軽にできる
- メッセージを振り返りやすい

Botを動かすと画面上では図5.2のようになる．

メッセージ送信すると即座に返答が返ってくるので，打っていてとても気分が良い．また，開始時刻と終了時刻を表示することにより，自分がどれくらい勉強したかをおおまかでも瞬時に分かるのが特徴となる．

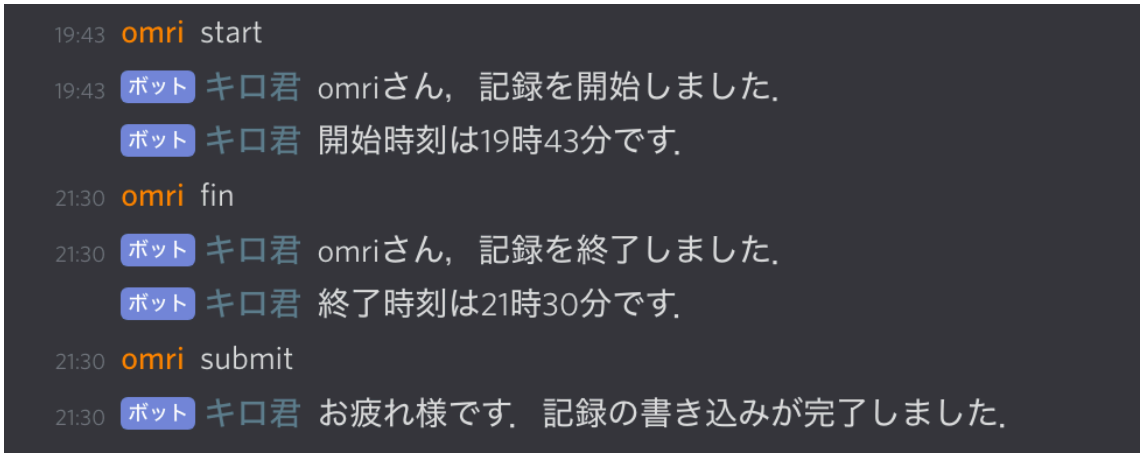

図 5.2: Bot の受け答え.

```
~/G/grad_thesis_19  master *  o/t/Discord  cat result.text
[2020-02-07-Fri 16:03]  omriさんの学習時間は9秒です.
[2020-02-11-Tue 16:02]  omriさんの学習時間は1時間16分26秒です.
~/G/grad_thesis_19  master *  o/t/Discord
```

図 5.3: Result.text の中身.

結果を集めたものは Bot ファイルの直下に Result.text として保存される．Result.text の中身は図 5.3のようになっている.

出力結果に関しては，日付と時刻が取れている．後に集計ができるともっと良くなるとのアドバイスを得た.

第6章　考察

Discordを使用することにより，メールでの手間が省けるということは一つの大きなメリットとなると考える．それに加えて，コーディング作業と同時にDiscordを動かしていても動作が軽いので，遠隔作業の妨げにならない．

DiscordにBotを導入することにより，学習時間の記録を残し，学んだ学習物や成果物をJekyllへあげる事により，"記憶"として残す事が可能となった．

Discordの導入はメリットの方が多いと思うが，デメリットも生じると考えている．それは，教員と生徒の間に"けじめの曖昧性"が出てくくる可能性があることだ．このデメリットは言ってしまえば個人の意識の問題ではあるが，以前まではメールで堅く文章を作成していたものをChat形式に変えることにより，会話のキャッチボールのスピードは上がる分，多少の"雑さ"が生じるのではないかと考える．

第7章　総括

本研究では，Discord と VSCode の Live Share を研究室に取り入れることにより，よりスムーズに学習を進めることができた．そして，Discord に Bot を導入する事により，ただのコミュニケーションツールに収まることなく，学習を効率化し，自己の学習過程を振り返ることのできるツールへと変身させることができた．実際に Bot を使用していても，メッセージ一つで記録の開始，終了，書き込みができ，ストレスのない環境を作ることができた．

今後の展望を以下に挙げる．

- 取得したデータを元に 1 週間，1 ヶ月と長期的な学習時間を分析し，自分がいつどれくらい学習しているかを可視化できるようなソフトウェアの開発．
- Discord のサーバー全体を管理する Bot を作成し，メンバーは参加するだけで自動的にロールを割り振り，お互いにディスカッションできるチャンネルを自動生成できるような機能を持たす．
- Discord を使用していての不便な部分のアンケートを取り，それを補うような機能を付けた Bot を作成する．
- 24 時間 Bot が動くように，西谷研究室で使用しているサーバーに Bot を入れる．

今回の研究で，今までの研究室の在り方を見直すきっかけとなった．今後の研究室内での連絡や，相談，質問のあり方を変え，今より早くスムーズに作業が進むような環境を整える事ができたと考える．この学習環境は，目まぐるしく情報が交差する社会の中で，お互いに情報を交換し合い，精査し合う基盤として役立てて頂きたい．そして，様々な地域から学びにくるユニバーサルな大学こそいち早く取り入れもらいたいと強く願う．

謝辞

本論文の作成にあたり，多くの方々のご協力とご指導を頂きました．

先ず，日々多くのご指導をしてくださいました西谷滋人教授に心より深く感謝いたします．教授のご指導のおかげで，研究全体の方向性を見出すことができました．そして，研究の進行に伴い，様々な助言や，知識の供給をして頂きました西谷研究室の同輩や，先輩方々に深く感謝いたします．本当にありがとうございました．

参考文献

[1] Kent Beck (著), 和田 卓人 (翻訳), テスト駆動開発, (オーム社, 2017/10/14).

[2] Jekyll-GitHubPages, http://jekyllrb-ja.github.io(20/01/05,accessed).

[3] GitHub Pages について, https://help.github.com/ja/github/working-with-github-pages/about-github-pages(19/12/09,accessed).

[4] Visual Studio Live Share, https://visualstudio.microsoft.com/ja/services/live-share/(19/12/13,accessed).

[5] Discord Developer Portal - API Docs for Bots and Developers, https://discordapp.com/developers/applications/(19/12/17,accessed).

[6] 簡単な Discord Bot の作り方 (初心者向け), https://note.com/bami55/n/ncc3a68652697(19/12/17,accessed).

付録

VSCodeのセットアップファイルは，~/Library/Application\ Support/code/User/settings.jsonに置かれている．

私がtuningした設定を添付しておく．参考にしてもらえれば幸いです．

```
{
    "editor.minimap.enabled": false,
    //"workbench.editor.swipeToNavigate": true,
    "editor.renderWhitespace": "boundary",
    "editor.renderIndentGuides": true,
    "editor.quickSuggestions": true,
    "editor.tabSize": 2,
    "editor.insertSpaces": true,
    "files.trimTrailingWhitespace": true,
    "workbench.statusBar.visible": true,
    "workbench.activityBar.visible": false,
    "workbench.editor.enablePreviewFromQuickOpen": false,
    "workbench.editor.enablePreview": false,
    "workbench.startupEditor": "none",
    "window.zoomLevel": 0,
    "terminal.integrated.fontFamily": "Source Code Pro for Powerline",
    "terminal.integrated.fontSize": 13,
    "editor.fontSize": 13,
    "workbench.colorTheme": "One Dark Pro",
    "files.encoding": "utf8",
    "extensions.autoUpdate": true,
    "editor.formatOnPaste": true,
    "editor.formatOnType": true,
    "window.restoreWindows": "one",
    "editor.renderLineHighlight": "gutter",
    "workbench.tree.indent": 5,
    "workbench.tree.renderIndentGuides": "none",
    "editor.folding": true,
    "editor.wordWrap": "on",
```

```
    "workbench.editor.tabSizing": "shrink",
    "workbench.colorCustomizations": //追加項目（外観）
    {
        "statusBar.background": "#000000", // ステータスバー背景色
        "statusBar.foreground": "#8FBE8F", // ステータスバー前景色
        "statusBar.border": "#8FBE8F", // ステータスバーの境界線
        "sideBar.background": "#000000", // サイドバーの背景色
        "sideBar.foreground": "#8FBE8F", // サイドバーの前景色
        "sideBar.border": "#8FBE8F", // サイドバーの境界線
        "activityBar.background": "#000000", // アクティビティバーの背景色
        "activityBar.foreground": "#8FBE8F", // アクティビティバーの前景色
        "activityBar.border": "#8FBE8F", // アクティビティバーの境界線
        "editor.background": "#000000", // エディタ背景色
        "scrollbarSlider.background": "#8FBE8F", // スクロールバーの色
        "scrollbarSlider.hoverBackground": "#8FBE8F", // 移動中のスクロール
バーの色
        "editorGroupHeader.tabsBackground": "#000000", //"タブの背景色",
        "tab.activeForeground": "#8FBE8F", //""アクティブなタブの文字色",
        "tab.border": "#000000",
        "tab.unfocusedActiveBorder": "#000000",
        "tab.activeBorder": "#8FBE8F",
        "titleBar.border": "#8FBE8F",
        "tab.inactiveBorder": "#8FBE8F",
        "tab.inactiveBackground": "#000000", //"非アクティブな色の背景色",
        "tab.activeBackground": "#000000", //"非アクティブな色の背景色",
        "tab.inactiveForeground": "#888888", //"非アクティブなタブの文字色",
        "terminal.border": "#8FBE8F",
        "panel.border": "#8FBE8F",
        "panelInput.border": "#8FBE8F",
        "panelTitle.activeBorder": "#8FBE8F",
        "menu.border": "#8FBE8F",
        "terminal.foreground": "#8FBE8F",
        "focusBorder": "#8FBE8F",
        "window.activeBorder": "#8FBE8F",
        "window.inactiveBorder": "#8FBE8F",
        "editorGroup.border": "#8FBE8F",
        "editorLineNumber.foreground": "#8FBE8F",
    },
    "workbench.iconTheme": "vs-minimal",
    "workbench.panel.defaultLocation": "bottom",
    "workbench.editor.showTabs": true,
```

```
    "editor.smoothScrolling": true,
    "editor.scrollbar.horizontal": "hidden",
    "editor.scrollbar.vertical": "hidden",
    "terminal.integrated.scrollbar.horizontal": "hidden",
    "terminal.integrated.scrollbar.vertical": "hidden",
    "terminal.integrated.experimentalRestore": true,
    "editor.fastScrollSensitivity": 1,
    "editor.scrollBeyondLastColumn": 1,
    "editor.scrollBeyondLastLine": false,
    "terminal.integrated.fontWeightBold": "bold",
    "terminal.integrated.cwd": "/Users/your_uername/",
    "workbench.list.horizontalScrolling": false,
    "workbench.list.verticalScrolling": false,
    "editor.cursorSmoothCaretAnimation": true,
    "editor.cursorBlinking": "solid",
    "terminal.integrated.cursorBlinking": false,
    "editor.formatOnSave": true,
}
```