亞東科技大學
Asia Eastern University of Science and Technology

電子工程系

學生專題研究報告書

Discord輔助管理機器人

專 題 生:許家銘(108104322)

指導教授:李烱三

中華民國一一二年 一 月 五 日

目錄

摘要

1、 研究動機

使用網路上開放的機器人後，我覺得為了一兩個小功能要多拉一隻機器人進群組不但不方便功能也容易重複。因此想製作出一個 Discord 的機器人，滿足自己的需求:方便、好管理。

2、 研究構想

製作伺服器日誌紀錄，當觸發事件時將訊息發送到特定頻道，並且排除非伺服器、非伺服器成員、機器人、指令訊息。

使用指令管理伺服器，用鍵對值的方式儲存指令以提高效率。添加指令前綴用於防呆，並且在程式內對訊息內容進行處理。

使用資料庫儲存資料，紀錄伺服器功能設定。

器材設備

（1）硬體規格

CPU：Intel Core i7-10700F

RAM：16G

作業系統：Windows 10 家用版

(2)開發語言選擇

網路上的Discord API Lib大約有20種，可以使用C#、Java、Ruby、PHP、JavaScript、Python…語言編寫。其中Discord.js更新最快，因此我選擇使用 JavaScript 撰寫 Discord機器人。由於專案不大因此我採用SQLite3這個輕量簡易的SQL資料庫引擎寫入及讀取資料庫。

(3)軟體環境

Node.js

SQLite3

Discord

Visual studio code

DB Browser for SQLite

(4) 軟體介紹

Node.js:

採用 Google 開發的 V8 執行程式碼，使用事件驅動、非阻塞和非同步輸入輸出模型等技術來提高效能，優化應用程式的傳輸量和規模。通常用於資料密集的即時應用程式。

Discord：

當初的是為了社群而設計的免費網路即時通話軟體與數位發行平台，主要針對遊戲玩家、教育人士及商業人士，Discord 的伺服器以主題分成不同的頻道，讓使用者之間可以在軟體的聊天頻道互相傳遞資訊、圖片、音訊等，這軟體可以在 Windows、Microsoft、MacOS、Android、ios、Linux 和網頁上執行。

Visual studio code：

由微軟開發且跨平台的免費原始碼編輯器。該軟體支援語法突顯、程式碼自動補全、程式碼重構功能，並且內建了命令列工具和 Git 版本控制系統。使用者可以更改佈景主題和鍵盤捷徑實現個人化設定，也可以透過內建的擴充元件程式商店安裝擴充元件以加強軟體功能。

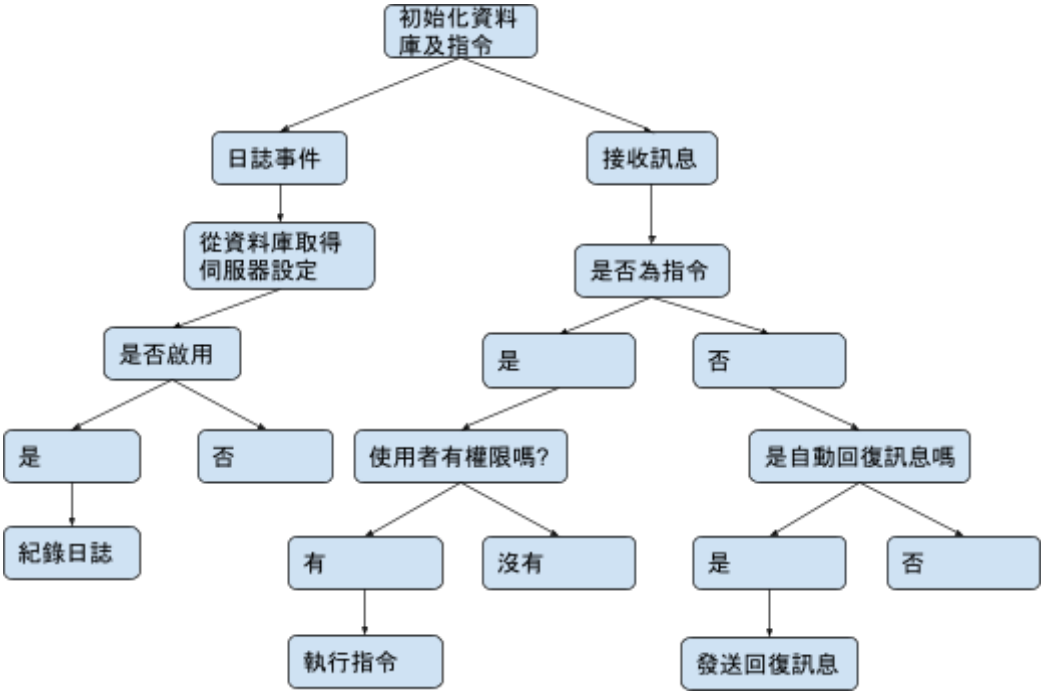
DB Browser for SQLite：

由於SQLite不像MySQL有phpMyAdmin的工具可使用，因此需要DB Browser for SQLite以查看資料庫方便除錯，也可以輕鬆的建立資料庫與欄位，設定欄位屬性，能瀏覽資料庫中的資料，支援Windows與MAC等平台。

原理方法

利用Discord.js 程式庫提供的函數生成HTTP標頭讓用戶端和伺服器透過要求或回應傳遞資料。

程式流程圖



計畫進度

項目\週數	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34	36
學習Discord.js API																		
學習資料庫系統																		
模組化																		
使用鍵對值儲存指令																		
使用JSDoc註解																		
使用物件導向程式																		
做報告																		

研究步驟

1、 測試

在撰寫第一個功能之前，要先了解聊天機器人是如何運作，Discord.js API 中，已經將可能發生的事件都已經打包好了，只需要登入機器人密鑰接者調用事件並且加上設定的條件(指令)就可以讓機器人說話了。

程式 & 測試結果:

```
const { Client } = require('discord.js');
const client = new Client({ ···
});

client.on("messageCreate", async (message) => {
    if (message.author.bot) return
    if (message.content === "ping")
        message.reply({ content: 'pong' })
})
```

2、 調用事件

```
// 刪除訊息&更新訊息
.on('messageDelete', message => {
    if (message.author.bot) return
    var ch = guildlogch(message.guild, 'message')
    if (!ch) return
    ch.send({
        embeds: [new EmbedBuilder()
            .setDescription(`${message.author} ${message.channel}`)
            .setFields({ name: `已刪除`, value: message.content })
            .setTimestamp(message.createdAt)]
    })
    if (message.attachments)
        message.attachments.forEach(x => ch.send({ content: x.url }))
}).on('messageUpdate', (old, message) => {
    if (message.author.bot) return
    if (old.content == message.content) return
    var ch = guildlogch(message.guild, 'message')
    if (!ch) return
    ch.send({
        embeds: [new EmbedBuilder()
            .setDescription(`${message.author} ${message.channel}`)
            .addFields({ name: `原訊息:`, value: old.content })
            .addFields({ name: `編輯後:`, value: message.content })
            .setTimestamp(message.createdAt)]
    })
})
```

```
// 加入伺服器&離開伺服器
.on('guildMemberAdd', member => {
        db.prepare(`INSERT OR IGNORE INTO user (userid, name)
    VALUES ('${member.id}', '${member.user.tag}')`).run()
    guildlogch(member.guild, 'servermember')?.send({
        embeds: [
            new EmbedBuilder().setDescription(member.user + ' 加入伺服器')]
    })
}).on('guildMemberRemove', member => {
    guildlogch(member.guild, 'servermember')?.send({
        embeds: [
            new EmbedBuilder().setDescription(member.user + ' 離開伺服器')]
    })
})
```

```
// 語音狀態
.on('voiceStateUpdate', async (old, stage) => {
    //#region 自動語音房…

    // 語音狀態更新
    var ch = guildlogch(stage.guild, 'voicestage')
    if (ch) {
        var eb = new EmbedBuilder()
        if (old.channel && gc?.avch && old.channelId != gc?.avch)
            eb.addFields({ name: `離開`, value: `${old.channel}` })
        if (stage.channel && gc?.avch && stage.channelId != gc?.avch)
            eb.addFields({ name: `加入`, value: `${stage.channel}` })
        if (eb.data.fields?.at(0))
            ch.send({ embeds: [eb.setDescription(`${stage.member.user}`)] })
    }
})
```

3、 物件導向及模組化

在一開始，我對於物件導向程式和模組化不是很有蓋念，也覺得很麻煩，一個檔案裏面全部都是if 後來發現越來越攏長如果要修改功能不但不容易找到在哪一行、變數名也容易重複，因此開始學習模組化和物件導向及使用JsDoc建立模型。

使用require()將json物件導入專案，在專案中例如:指令前綴,機器人的密鑰...可能會更動的參數直接存成json，方便以後修改，要將專案開源分享時，也能方便檢查密鑰等等...個人設定是否有清除 。剛開始想簡化程式的時候是使用json物件來儲存資料，但是當資料越來越多時，在讀取檔案的時候就會越來越慢，所以需要用到資料庫。而SQLite套件可以在JS中輕鬆儲存與讀取資料庫內的資料。

導入json物件

config.json
```
{
    "OwnerId": "",
    "Prefix": "=",
    "Token": ""
}
```

bot.js
```
const { Prefix, OwnerId, Token } = require('./config.json');
```

用鍵對值的方式儲存指令

```
/** @type {Collection<string,string[]>} */
const Modules = new Collection()
/** @type {Collection<string,ICommand>} */
const Commands = new Collection()
/** @type {Collection<string,string>} */
const Aliases = new Collection()
```

用JsDoc註解建立模板

```
/**
 * @typedef {"OwnerOnly"|PermissionsString} CmdPermissions
 * @typedef {(context:Message, args:string[]) => any} run
 *
 * @typedef ICommand
 * @property {string} module
 * @property {string[]} aliases
 * @property {string} description
 * @property {string[]} example
 * @property {CmdPermissions} permissions
 * @property {run} exec
 */
```

建立類別

```
class Command {
    /** @type {ICommand} */ data = {}
    /** @param {string | undefined} modname */
    constructor(modname) { this.modname = modname?.trim()[0] ? modname : 'other' }
    /** @param {string[]} alias */
    setName(...alias) {
        this.data.aliases = alias
        alias.forEach(x => Aliases.set(x, alias[0]))
        return this
    }
    /** @param {string[]} value */
    setDescription(...value) { this.data.description = value; return this }
    /** @param {string[]} value */
    setExample(...value) { this.data.example = value; return this }
    /** @param {CmdPermissions} value */
    setPremissions(value) { this.data.permissions = value; return this }
    /** @param {run} exec  */
    setexec(exec) {
        this.data.exec = exec
        this.data.module = this.modname
        this.data.description ??= ""
        this.data.example ??= [""]
        this.data.permissions ??= "SendMessages"
        Commands.set(this.data.aliases[0], this.data)
        Modules.set(this.modname, (Modules.get(this.modname) ?? [])
            .concat(this.data.aliases[0]))
        this.data = {}
        return this
    }
}
```

導出物件

```
module.exports = { client, db, Modules, Commands, Aliases, Command }
```

設計指令

```
.setName('autovoicechannel', 'avch')
.setDescription('自動語音頻道')
.setExample(`autovoicechannel 語音頻道`,`autovoicechannel`)
.setPremissions('Administrator')
.setexec(async (message, args) => {
    if (!message.inGuild()) return
    var ch = message.mentions.channels.first() || message.guild.channels.cache.get(args[1]) || message.channel
    if(!ch.isVoiceBased()) return
    db.prepare(`update guildconfig set autovoicechannel = '${ch.id}' where guildid = '${message.guildId}'`).run()
    message.channel.send({ embeds: [new EmbedBuilder().setDescription(`當使用者加入 ${ch} 後 將會自動一個語音頻道`)] })
})
```

Commands > Administrator > JS logger.js > ...

```
const { EmbedBuilder } = require("discord.js")
const { Command, db } = require("../../Builder")

new Command('Administrator')
    .setName('logset')
    .setDescription(`loggerlist: message\nvoicestage\nservermember`)
    .setExample('logset message', 'logset all [channelid]')
    .setPremissions('Administrator')
    .setexec(async (message, args) => {
        if (!args[1]) return
        args[2] ??= message.mentions.channels.first()?.id || message.channelId
        var loglist = ['message', 'voicestage', 'servermember']
        if (args[1] === 'all') {
            db.prepare(`update guildconfig
                        set message = '${args[2]}',
                            voicestage = '${args[2]}',
                            servermember = '${args[2]}'
                        where guildid = '${message.guildId}'`).run()
            message.channel.send({
                embeds: [new EmbedBuilder()
                    .setDescription(`將 ${loglist.join(', ')} 記錄到 <#${args[2]}>`)]
            })
        }
        else if (loglist.includes(args[1])) {
            db.prepare(`update guildconfig
                        set ${args[1]} = '${args[2]}'
                        where guildid = '${message.guildId}'`).run()
            message.channel.send({
                embeds: [new EmbedBuilder()
                    .setDescription(`將 ${args[1]} 記錄到 <#${args[2]}>`)]
            })
        }
    })
    .setName('logremove', 'logrm')
    .setDescription(`loggerlist: message\nvoicestage\nservermember`)
    .setExample('logrm all', 'logremove message')
    .setPremissions('Administrator')
    .setexec(async (message, args) => {
        if (!args[1]) return
        var loglist = ['message', 'voicestage', 'servermember']
        args[2] ??= message.mentions.channels.first()?.id || message.channelId
        if (args[1] === 'all') args[1] = loglist.map(x => `${x} = "${args[2]}" `).join(',')
        else if (loglist.includes(args[1])) args[1] += ` = ""`
        db.prepare(`update guildconfig set ${args[1]} where guildid = '${message.guildId}'`).run()
        message.channel.send({ embeds: [new EmbedBuilder().setDescription(`將 ${loglist} 移除`)] })
    })
```

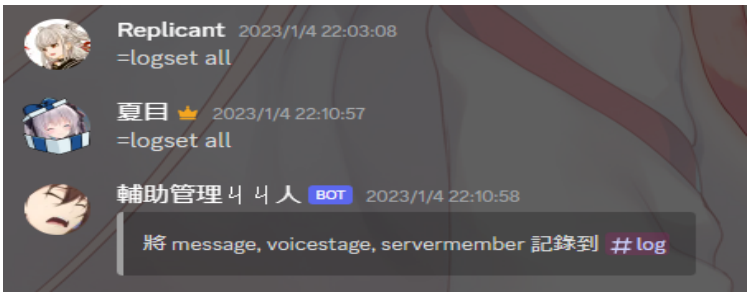
將主程式中掃描資料夾內的檔案並導入

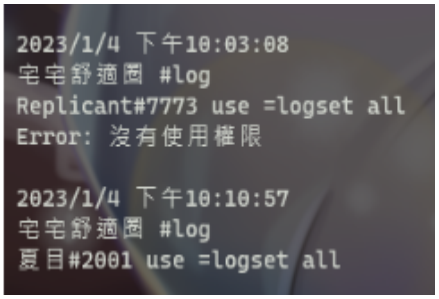
```
client.on("ready", () => {
    //#region  init db···
    fs.readdirSync('./Commands').forEach(x =>
        fs.readdirSync(`./Commands/${x}`).forEach(file =>
            require(`${process.cwd()}\\Commands\\${x}\\${file}`)))

    console.log(`\n${client.user.tag} 已登入` +
        `\nGUILD(${client.guilds.cache.size}) -> [P:${Commands.size}]`)

})
```

成果展示

1、 指令執行

```
}).on("messageCreate", async (message) => {
    //#region test reply…

    const { content, createdAt, member, guild, channel, author } = message

    try {
        if (execute()) return
        //#region 自動回覆訊息…
    } catch (error) { console.log(error) }

    //#region Cmd-execute
    function execute() {
        const args = content.trim().split(' ')
        const prefix = [Prefix, `${client.user}`].find(x => args[0].startsWith(x))
        if (!prefix) return false
        args[0] = Aliases.get(args[0].toLowerCase().slice(prefix.length))
        const cmd = Commands.get(args[0])
        var logstr = `\n${createdAt.toLocaleString()}` +
            `${guild.name} #${channel.name} ${author.tag} use ${content}`
        if (!cmd) {
            channel.send('找不到指令')
            return true
        }
        var result = cmd.permissions == "OwnerOnly"
            ? author.id == OwnerId
            : member?.permissions.has(cmd.permissions)

        if (result) cmd.exec(message, args)
        else logstr += `\nError: 沒有使用權限`
        console.log(logstr)
        return true
    }
    //#endregion
})
```

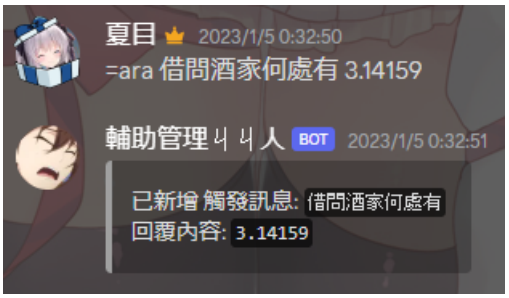


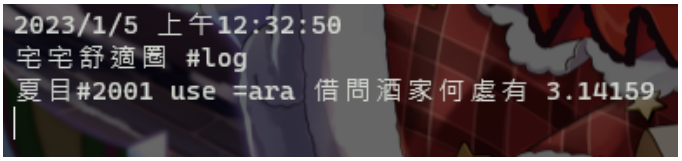


	guildname	message	voicestage	servermember
	過濾	過濾	過濾	過濾
8	宅宅舒適圈	1054425118446714971	1054425118446714971	1054425118446714971

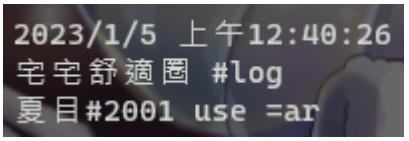
2、 自動回覆訊息

```
//#region 自動回覆訊息
if (!db.prepare(`select channelid from autoreactchannel
                 where channelid = '${channel.id}'`).get()?.channelid) return
var autoreactlist = db.prepare(`select * from autoreact where guildid = ${guild.id} or guildid = 0`).all()
    .filter(x => content.includes(x.target))
if (autoreactlist.length > 0) {
    console.log(`\n${createdAt.toLocaleString()}` +
    `${guild.name} #${channel.name} ${author.tag} send ${content}`)
    channel.send({ content: autoreactlist[Math.floor(Math.random() * autoreactlist.length)].text })
}
//#endregion
```

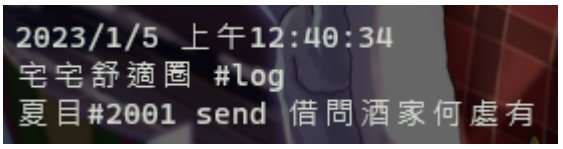


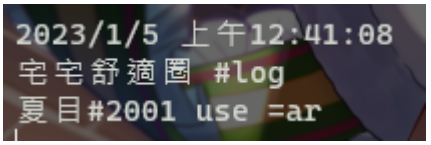


id	guildid	text	target
…	過濾	過濾	過濾
1	626251441127948288	3.14159	借問酒家何處有



id	guildid	channelid
…	過濾	過濾
1	626251441127948288	1054425118446714971





id	guildid	channelid
…	過濾	過濾

3、 自動語音房

```
//#region 自動語音房
var gc = db.prepare(`select autovoicechannel as avch
                     from guildconfig where guildid = '${stage.guild.id}'`).get()
if (gc?.avch && stage.channel && stage.channelId == gc.avch) {
    var avch = await stage.channel.clone({ name: stage.member.user.username + "'s Room" })
    stage.setChannel(avch)
    db.prepare(`INSERT OR IGNORE INTO autovoicechannel
                (guildid, channelid) VALUES ('${stage.guild.id}', '${avch.id}')`).run()
}
if (!stage.channel)
    db.prepare(`select channelid as id from autovoicechannel`).all().forEach(x => {
        var dech = stage.guild.channels.cache.get(x.id)
        if (!dech) return db.prepare(`DELETE from autovoicechannel
                                      where channelid = '${x.id}'`).run()
        if (!dech.members?.first()) {
            db.prepare(`DELETE from autovoicechannel where channelid = '${x.id}'`).run()
            dech.delete()
        }
    })
//#endregion
```

4、 日誌事件

結論

優點:當有需要時，可以隨時加入新功能，也方便管理。

缺點:若是需要24小時在線上需要租伺服器或是電腦維持24小時不關機。

上個學期是我真正意義上第一次接觸程式從一開始只能看著教學一步一步做，到現在學會模組化和物件導向程式。在這將近一年的時間，我發現程式沒有想像中的難，也了解到自己要學的還有很多，而網路上也有許多資源可以用來精進自己，雖然程式語言有很多種，但是語法也都差不多，只要有了基本概念很容易就能觸類旁通。