

Mirai-console 插件教程

My First Plugin - kotlin 教程

Mar 14 2020

前言

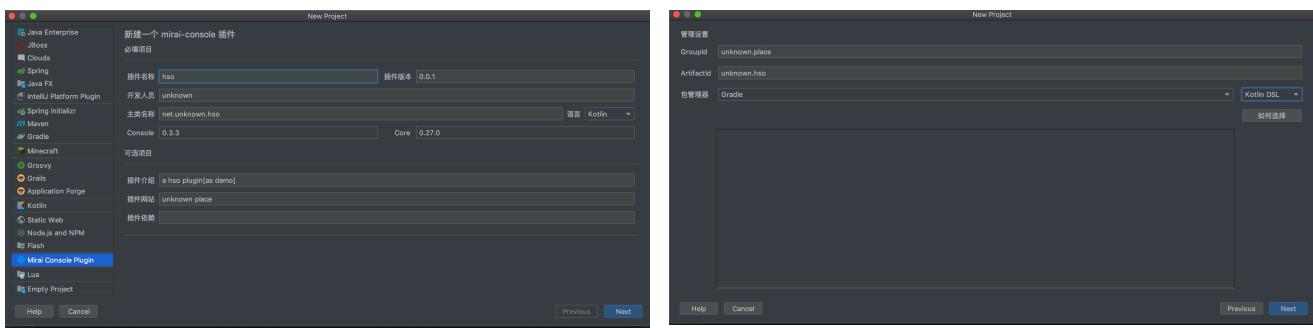
本篇教程基于你已经阅读了Mirai-console 插件开发中的如何上手, 将着重于如何实际的写出第一个有一定功能性的插件, 阅读本文不需要有很了解kotlin

文章所开发的是一个群内涩图插件, 仅供学习参考, 公开使用请注意所在国法律与社会价值观

本文共涉及两个插件项目, 完整版工程项目源码均可在github中找到

新建项目

根据上文所说, 在idea中添加插件, 启动一个插件项目, 我们给插件起名为hso, 并将开发者名设置为unknown



[Console/Core]版本可能在你阅读本文时已有更新, 请不要在意

The image shows a screenshot of the IntelliJ IDEA code editor with the project 'hso' open. The code editor displays the 'hso.kt' file, which contains the following Kotlin code:

```
1 package net.unknown
2
3 import ...
4
5 object hso : PluginBase() {
6
7     lateinit var images: Config
8     lateinit var normal: List<ConfigSection>
9     lateinit var r18: List<ConfigSection>
10
11     override fun onLoad() {
12         super.onLoad()
13     }
14
15     override fun onEnable() {
16         super.onEnable()
17     }
18 }
```

(hso作为object名应该是Hso, 这是一个不规范的例子)

在插件项目建立完成后, 你也应当会看到类似的初始代码

`onLoad()` `onEnable()` 是插件开发中的两个核心方法, 也是插件生命周期中重要的部分, 简单的说, 当一个插件被加载, `onLoad`方法会被先调用, 全部插件的`onLoad`方法被调用后, `onEnable`的方法会被依次调用

`onLoad()` 正如方法名, 是插件在加载时应做的逻辑, 例如

- 1: 加载本地资源, 测试本地资源是否正常(或更之前)
- 2: 加载配置文件(或更之前)
- 3: 初始化一些变量

但以下逻辑不推荐或不能做:

- 1: 检查依赖插件是否加载
- 2: 注册指令
- 3: 注册事件监听

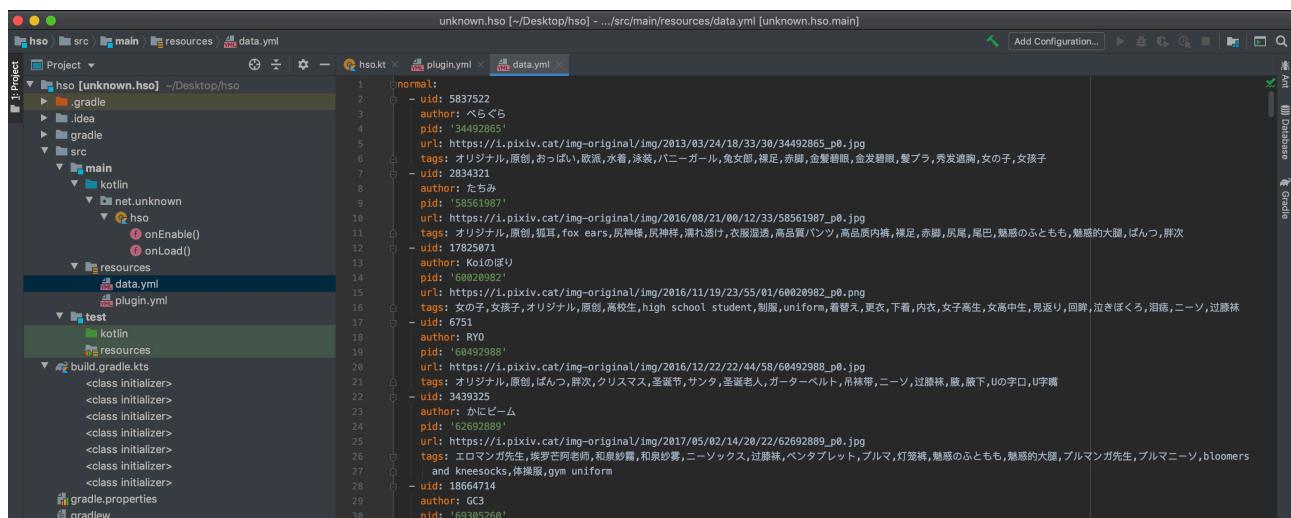
原因是因为你的插件在被调用`onLoad()`时, 其他插件还没有进入正式的生命周期

`onEnable()` 是插件启动时应做的逻辑, 如

- 1: 注册指令
- 2: 注册事件监听
- 3: 启动插件中的循环任务, 协程, 或`worker`

两个方法均不应该出现长时间的堵塞, 短时间的阻塞, 如通过`http`检查最新版本是允许并推荐的, 对于这种情况, 您应当在`onLoad()`时异步请求, `onEnable`的时候等待结果, 已寻求最短的阻塞时间。

对于一个涩图插件, 图源是十分重要的, 本文采用的是 <https://api.lolicon.app/#/setu> 的涩图api, 但鉴于该站不是很稳定且过量的重复请求极为浪费, 我们以极慢的速度获取了该站的1000余张涩图数据, 将其保存在一个YAML文件, 并根据分级储存为normal/r18两档, 并将文件放到/resources/下



The screenshot shows an IDE interface with the following details:

- Project Structure:** The project is named "hso [unknown.hso]". It contains a "src" directory with "main" and "test" sub-directories. "main" has "kotlin" and "resources" sub-directories. "resources" contains "data.yml" and "plugin.yml". "test" contains "resources".
- Code Editor:** The "data.yml" file is open. It is a YAML file listing various images with their details (uid, author, pid, url, tags). The first few entries are:

```
1: normal:
  - uid: 5837522
    author: ベジぐら
    pid: '34492865'
    url: https://i.pixiv.cat/img-original/img/2013/03/24/18/33/30/34492865_p0.jpg
    tags: オリジナル,原创,おっぱい,欧派,水着,泳装,バニーガール,兔女郎,裸足,赤脚,金髮碧眼,金发碧眼,秀发遮胸,胸,女孩子
  - uid: 2834321
    author: たちみ
    pid: '58561987'
    url: https://i.pixiv.cat/img-original/img/2016/08/21/00/12/33/58561987_p0.jpg
    tags: オリジナル,原创,猫耳,fox ears,原神様,虎神様,瀧れ透け,衣服湿透,高品质/パンツ,高品质内裤,裸足,赤脚,尻尾,尾巴,魅惑のふともも,魅惑的大腿,パンつ,胖次
  - uid: 17825871
    author: Ko!のぼり
    pid: '60026982'
    url: https://i.pixiv.cat/img-original/img/2016/11/19/23/55/01/60026982_p0.png
    tags: 女の子,女孩子,オリジナル,原创,高校生,high school student,制服,uniform,着替え,更衣,下着,内衣,女子高中生,女高中生,見返り,回眸,泣きぼくろ,泪痣,ニーソ,过膝袜
```

- Build Configuration:** The "build.gradle.kts" file is shown in the bottom left, indicating the use of Kotlin for build scripts.

配置文件是插件中的重要部分

Console为插件提供了方便的配置文件API用于格式化读写文件, 插件支持了yaml, json, toml, ini(properties)等文件格式

配置文件总的来说分为两种, 可读文件与读写文件

可读的配置文件是你放在/resources/中的文件, 这类文件读取出来后只可读 不可保存

可写的配置文件是你生成在_ROOT_/_plugins/插件名/ 下的配置文件, 这类文件插件可读可写, 且用户可读可写。

如果要读取一个resources中的配置文件, *PluginBase*中的`getResourcesConfig(fileName)`提供了支持, 如果要读取(如不存在则创建)一个可读可写的配置文件, *PluginBase*中的`loadConfig(fileName)`提供了支持

*Config*是线程安全的

我们在onLoad方法中将/resources/中的涩图数据读取出来并进行储存到内存中, 并进行简单的数据统计与错误处理

```
import ...

object hso : PluginBase() {

    lateinit var images: Config
    lateinit var normal: List<ConfigSection>
    lateinit var r18: List<ConfigSection>

    override fun onLoad() {
        logger.info("loading local image data")
        try {
            images = getResourcesConfig(fileName: "data.yml")
        } catch (e: Exception) {
            e.printStackTrace()
            logger.info("无法加载本地图片")
        }
        logger.info("本地图片版本" + images.getString(key: "version"))

        r18 = images.getConfigSectionList(key: "R18")
        normal = images.getConfigSectionList(key: "normal")
        logger.info("Normal * " + normal.size)
        logger.info("R18 * " + r18.size)
    }

    override fun onEnable() {
        super.onEnable()
    }
}
```

Config 与 ConfigSection

*Config*和*ConfigSection*均为键值对应的数据结构, 他的数据结构类似于*Map<String, Any!>*, 其中键一定是*String*请牢记, *Config*与*ConfigSection*中提供了高效的获取方式, 将*Any!*转为其他数据格式, 如*getLong()* *getDouble()* *getLongList()* *getStringList()*等, 但这不足以满足复杂的需求, 因此出现了*ConfigSection*

*ConfigSection*也是一个*Map<String, Any!>*, 且可以作为一个*Any!* 被放入其他的*Config*/*ConfigSection*, 这样套娃, 你就可以解决99%的场景需求

以这个插件为例, *data.yml*(涩图数据文件), 就是一个使用了*Config/ConfigSection*的场景, 整个文件读取为*Config*后, 只有两个键, *normal*和*r18*, 而它们对应的则各是一个*ConfigSection List*, 因此满足了这个复杂场景的需要

在处理完成涩图数据后, 我们来对编写插件的配置文件(即用户可以自定义的东西)

```
1.1 object hsq : PluginBase() {  
1.2       
1.3     lateinit var images: Config  
1.4     lateinit var normal: List<ConfigSection>  
1.5     lateinit var r18: List<ConfigSection>  
1.6       
1.7     val config = loadConfig(fileName: "setting.yml")  
1.8       
1.9     val Normal_Image_Trigger by config.withDefaultWriteSave { "色图" }  
2.0     val R18_Image_Trigger by config.withDefaultWriteSave { "不够色" }  
2.1     val Image_Resize_Max_Width_Height by config.withDefaultWriteSave { 800 }  
2.2       
2.3     override fun onLoad() {  
2.4         logger.info("loading local image data")  
2.5         try {  
2.6             images = getResourcesConfig(fileName: "data.yml")  
2.7         } catch (e: Exception) {  
2.8             e.printStackTrace()  
2.9             logger.info("无法加载本地图片")  
3.0         }  
3.1         logger.info("本地图片版本" + images.getString(key: "version"))  
3.2           
3.3         r18 = images.getConfigSectionList(key: "R18")  
3.4         normal = images.getConfigSectionList(key: "normal")  
3.5         logger.info("Normal * " + normal.size)  
3.6         logger.info("R18 * " + r18.size)  
3.7     }  
3.8       
3.9     override fun onEnable() {  
4.0         super.onEnable()  
4.1     }  
4.2       
4.3 }
```

首先在18行, 我们加载了一个config, 这个“setting.yml”会被创建在用户目录/plugins/hso(插件名)/下, 以供用户使用, 接着我们定义了三个简单类型的变量, 他们使用了Config提供的代理模式

Config 代理模式

Config 提供了kotlin的特性-代理(by)功能, 可以轻松的连接配置文件与代码

Var aValue by config.withDefaultWriteSave{0}的作用简单来说, 当aValue被使用时, 他会在config中寻找键为aValue(变量名)的数据, 如果没有, 则返回0, 并在配置文件写入aValue=0并保存, 当你给aValue赋值的时候, 配置文件中的数据会更改, 但不会保存

因此变量命名就变的比较重要, 这里不推荐使用驼峰命名, 推荐使用大写首字母与下划线的方式, 毕竟是所有用户都看得懂驼峰的

与withDefaultWriteSave类似的还有

with	如果没有找到对应数据, 就会error
withDefault{}	如果没有找到对应数据, 就会返回default数值
withDefaultWrite{}	如果没有找到对应数据, 只会写入不会保存

Config代理模式仅支持基础数据格式, 如Int/Long/Double 或LongList DoubleList等

不支持ConfigSection/ ConfigSectionList

且读写list 不推荐使用代理

```
3 import ...
11
12 object hso : PluginBase() {
13
14     lateinit var images: Config
15     lateinit var normal: List<ConfigSection>
16     lateinit var r18: List<ConfigSection>
17
18     val config = loadConfig( fileName: "setting.yml")
19
20     val Normal_Image_Trigger by config.withDefaultWriteSave { "色图" }
21     val R18_Image_Trigger by config.withDefaultWriteSave { "不够色" }
22     val Image_Resize_Max_Width_Height by config.withDefaultWriteSave { 800 }
23
24
25     val groupsAllowNormal = config.getLongList( key: "Allow_Normal_Image_Groups").toMutableList()
26     val groupsAllowR18 = config.getLongList( key: "Allow_R18_Image_Groups").toMutableList()
27
28     override fun onLoad() {
29         logger.info("loading local image data")
30         try {
31             images = getResourcesConfig( fileName: "data.yml")
32         } catch (e: Exception) {
33             e.printStackTrace()
34             logger.info("无法加载本地图片")
35         }
36         logger.info("本地图片版本" + images.getString( key: "version"))
37
38         r18 = images.getConfigSectionList( key: "R18")
39         normal = images.getConfigSectionList( key: "normal")
40         logger.info("Normal * " + normal.size)
41         logger.info("R18 * " + r18.size)
42     }
43
44     override fun onEnable() {
45         super.onEnable()
46     }
47 }
```

紧接着，我们读取出允许发送普通图片与r18图片的群组群号列表，这里没有使用by，因此在插件关闭时将它写回去是十分重要的

```
object hso : PluginBase() {

    lateinit var images: Config
    lateinit var normal: List<ConfigSection>
    lateinit var r18: List<ConfigSection>

    val config = loadConfig(fileName: "setting.yml")

    val Normal_Image_Trigger by config.withDefaultWriteSave { "色图" }
    val R18_Image_Trigger by config.withDefaultWriteSave { "不够色" }
    val Image_Resize_Max_Width_Height by config.withDefaultWriteSave { 800 }

    val groupsAllowNormal = config.getLongList(key: "Allow_Normal_Image_Groups").toMutableList()
    val groupsAllowR18 = config.getLongList(key: "Allow_R18_Image_Groups").toMutableList()

    override fun onDisable() {
        super.onDisable()
        logger.info("Saving data")
        config["Allow_Normal_Image_Groups"] = groupsAllowNormal
        config["Allow_R18_Image_Groups"] = groupsAllowR18
    }

    override fun onLoad() {
        logger.info("loading local image data")
        try {
            images = getResourcesConfig(fileName: "data.yml")
        } catch (e: Exception) {
            e.printStackTrace()
            logger.info("无法加载本地图片")
        }
        logger.info("本地图片版本" + images.getString(key: "version"))

        r18 = images.getConfigSectionList(key: "R18")
        normal = images.getConfigSectionList(key: "normal")
        logger.info("Normal * " + normal.size)
    }
}
```

`onDisable()` 与 `onLoad()` `onEnable()` 类似，是在插件要被关闭前调用的

如果这样写，你就会遇到一个问题，因为你没有保存文件，写了白写

```
26
27    override fun onDisable() {
28        super.onDisable()
29        logger.info("Saving data")
30        config["Allow_Normal_Image_Groups"] = groupsAllowNormal
31        config["Allow_R18_Image_Groups"] = groupsAllowR18
32        config.save()
33    }
34
```

`Config.save()`方法将`config`中的数据写入文件中，这个方法实际是阻塞的，但为了方便开发，它不是

紧接着, 我们来写图片的获取, 即从pixiv下载图片, 下载图片不需要出墙
 总的来说, 使用Jsoup获取图片数据, 并进行需要的压缩, 然后使用mirai的uploadAsImage方法上传到腾讯服务器, 最终返回mirai的Image

对于mirai api的教学不在本文范围之内, 请在mirai wiki学习, 那里更加齐全和详细

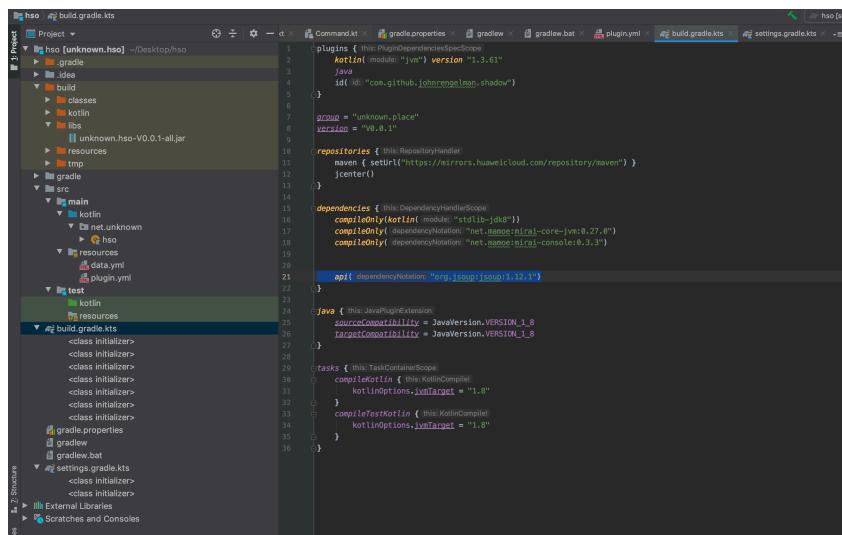
```

121     private suspend fun getImage(
122         contact: Contact,
123         url: String,
124         pid: String,
125         maxWidthOrHeight: Int,
126         anti: Boolean
127     ): Image {
128         val bodyStream = withTimeoutOrNull( timeMillis: 20 * 1000 ) { this: CoroutineScope
129             withContext(Dispatchers.IO) { this: CoroutineScope
130                 Jsoup
131                     .connect(url)
132                     .followRedirects( followRedirects: true )
133                     .timeout( millis: 180_000 )
134                     .ignoreContentType( ignoreContentType: true )
135                     .userAgent( userAgent: "Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_6_7; ja-jp) AppleWebKit/533.20.25 (KHTML, like Gecko) Version/5.0.4 Safari/533.20.27" )
136                     .referrer( referrer: "https://www.pixiv.net/member_illust.php?mode=medium&illust_id=$pid" )
137                     .ignoreHttpErrors( ignoreHttpErrors: true )
138                     .maxBodySize( bytes: 100000000 )
139                     .execute().also { check( value: it.statusCode() == 200 ) { "Failed to download image" } }
140             }
141         }?.bodyStream() ?: error("Failed to download image")
142         if (maxWidthOrHeight < 1) {
143             return bodyStream.uploadAsImage(contact)
144         }
145         val image = withContext(Dispatchers.IO) { this: CoroutineScope
146             ImageIO.read(bodyStream)
147         }
148         if (Image_Resize_Max_Width_Height <= 1200 || image.width.coerceAtLeast(image.height) <= maxWidthOrHeight) {//send master file
149             return image.upload(contact)
150         }
151         val rate = (maxWidthOrHeight.toFloat() / image.width.coerceAtLeast(image.height))
152         val newWidth = (image.width * rate).toInt()
153         val newHeight = (image.height * rate).toInt()
154         return withContext(Dispatchers.IO) { this: CoroutineScope
155             val dimg = BufferedImage(newWidth, newHeight, image.type)
156             if (anti) {
157                 dimg.setRGB( x: 1, y: 1, rgb: 0xFFFFFFFF )
158             }
159             val g = dimg.createGraphics()
160             g.setRenderingHint(RenderingHints.KEY_INTERPOLATION, RenderingHints.VALUE_INTERPOLATION_BILINEAR)
161             g.drawImage(image, 0, 0, newWidth, newHeight, 0, 0, image.width, image.height, null)
162             g.dispose()
163             dimg
164         }.upload(contact)
165     }

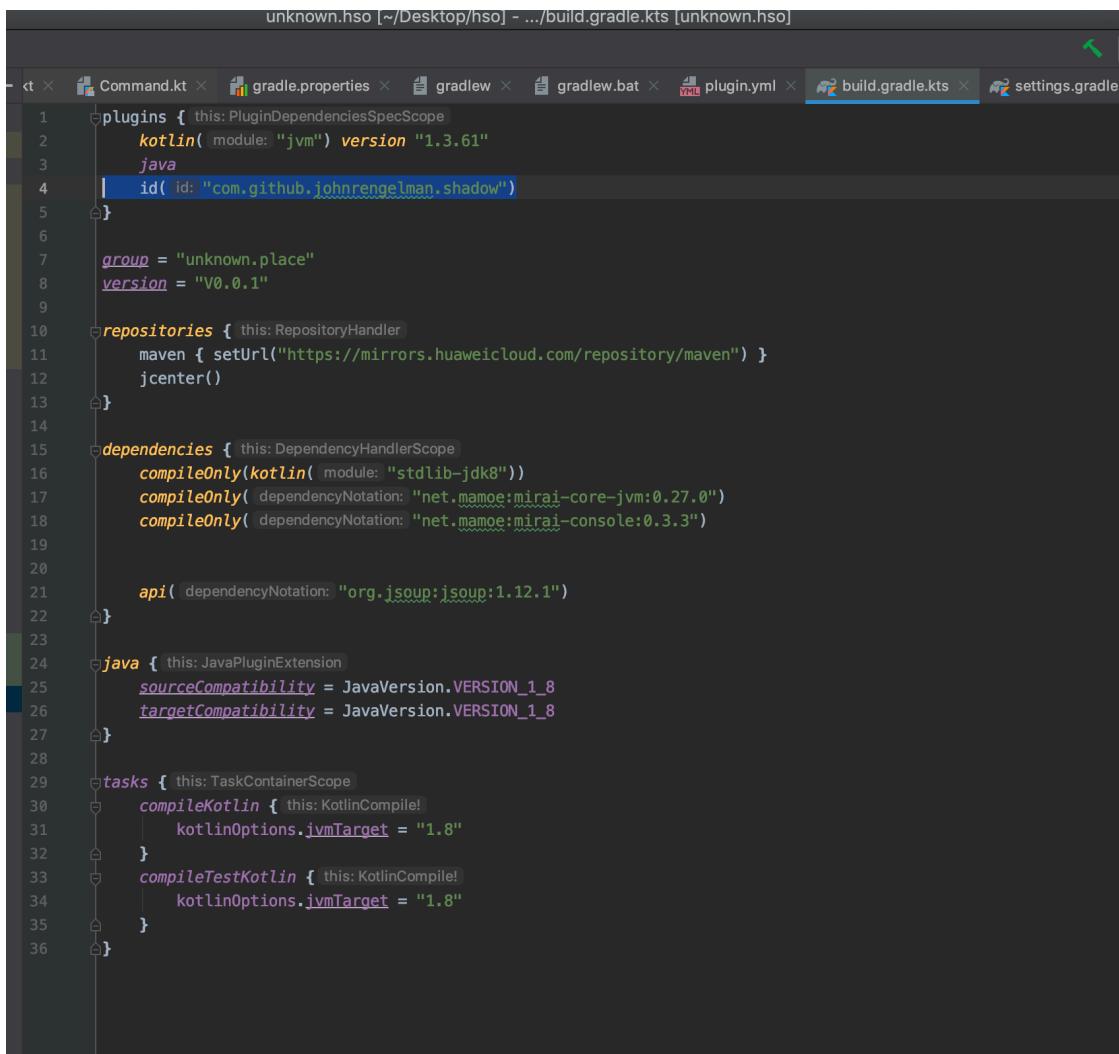
```

但是由于使用了Jsoup作为依赖, 我们需要将Jsoup加入到最终打包的jar中, **请注意依赖Jsoup和依赖其他插件是两个不同的概念, 在plugin.yml的depends中写jsoup是不明智的行为**

首先添加依赖

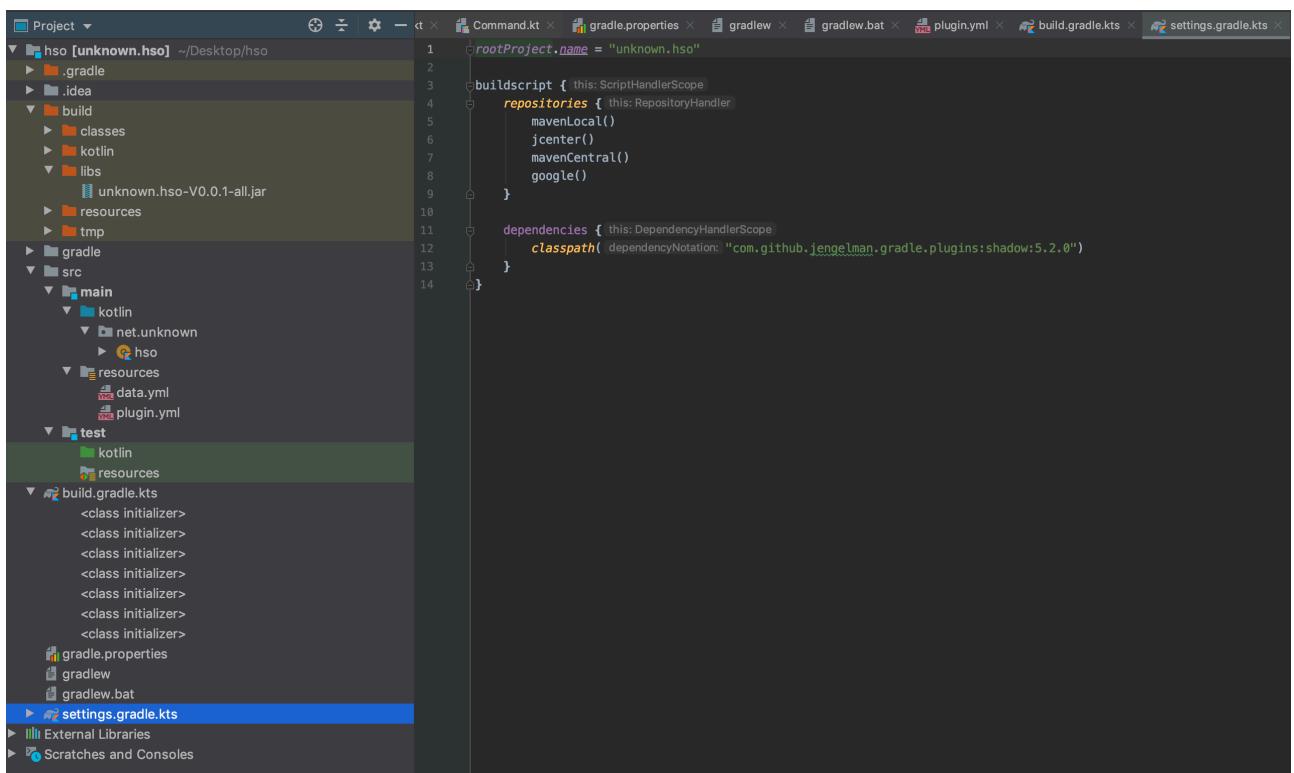


使用shadowJar



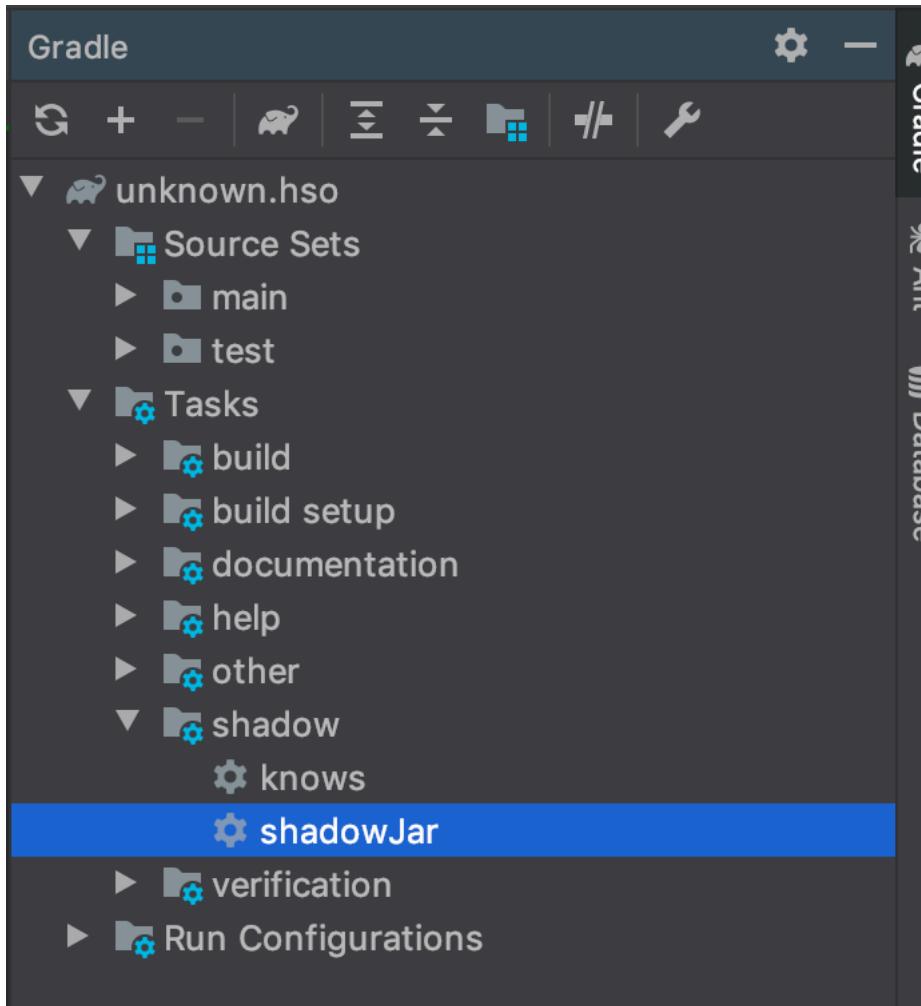
```
unknown.hso [~/Desktop/hso] - .../build.gradle.kts [unknown.hso]
1 plugins { this: PluginDependenciesSpecScope
2     kotlin(module: "jvm") version "1.3.61"
3     java
4     id(id: "com.github.johnrengelman.shadow")
5 }
6
7 group = "unknown.place"
8 version = "V0.0.1"
9
10 repositories { this: RepositoryHandler
11     maven { setUrl("https://mirrors.huaweicloud.com/repository/maven") }
12     jcenter()
13 }
14
15 dependencies { this: DependencyHandlerScope
16     compileOnly(kotlin(module: "stdlib-jdk8"))
17     compileOnly(dependencyNotation: "net.mamoe:mirai-core-jvm:0.27.0")
18     compileOnly(dependencyNotation: "net.mamoe:mirai-console:0.3.3")
19
20     api(dependencyNotation: "org.jsoup:jsoup:1.12.1")
21 }
22
23 java { this: JavaPluginExtension
24     sourceCompatibility = JavaVersion.VERSION_1_8
25     targetCompatibility = JavaVersion.VERSION_1_8
26 }
27
28 tasks { this: TaskContainerScope
29     compileKotlin { this: KotlinCompile!
30         kotlinOptions.jvmTarget = "1.8"
31     }
32     compileTestKotlin { this: KotlinCompile!
33         kotlinOptions.jvmTarget = "1.8"
34     }
35 }
36 }
```

添加仓库



```
Project
unknown.hso ~/Desktop/hso
1 rootProject.name = "unknown.hso"
2
3 buildscript { this: ScriptHandlerScope
4     repositories { this: RepositoryHandler
5         mavenLocal()
6         jcenter()
7         mavenCentral()
8         google()
9     }
10
11 dependencies { this: DependencyHandlerScope
12     classpath(dependencyNotation: "com.github.jengelman.gradle.plugins:shadow:5.2.0")
13 }
14 }
```

这样, 我们就可以使用shadowjar来进行打包, 最终的插件jar包中将包含jsoup



完成图片获取后, 我们将图片发送方法写好, 图片发送是异步执行的, 以防堵塞

```
97     private fun sendImage(contact: Contact, configSection: ConfigSection) {
98         launch { this: CoroutineScope {
99             try {
100                 logger.info("正在推送图片")
101                 getImage(
102                     contact,
103                     if(Image_Resize_Max_Width_Height > 1200){
104                         configSection.getString( key: "url")
105                     }else{
106                         with(configSection.getString( key: "url").replace( oldValue: "img-original", newValue: "img-master")){
107                             val index = this.lastIndexOf( string: ".")
108                             this.substring(0, index) + "_master1200.jpg"//it has to be jpg ^with
109                         }
110                     },
111                     configSection.getString( key: "pid"),
112                     Image_Resize_Max_Width_Height,
113                     Anti_Detect
114                     ).plus(configSection.getString( key: "tags")).sendTo(contact)
115                 } catch (e: Exception) {
116                     contact.sendMessage( plain: e.message ?: "unknown error")
117                 }
118             }
119         }
120     }
```

最终，我们在onEnable中增加群消息的监听，并判断是否要发送图片

```
80     override fun onEnable() {
81         logger.info("hso plugin enabled")
82         subscribeGroupMessages { this: MessageSubscribersBuilder<GroupMessage>
83             (contains(Normal_Image_Trigger)) { this: GroupMessage
84                 if (groupsAllowNormal.contains(this.group.id)) {
85                     sendImage(subject, normal.random())
86                 }
87             }
88             (contains(R18_Image_Trigger)) { this: GroupMessage
89                 if (groupsAllowR18.contains(this.group.id)) {
90                     sendImage(subject, r18.random())
91                 }
92             }
93         } ^subscribeGroupMessages
94     }
```

至此，涩图插件基本功能就完成了，但是我们希望他变得更适用，因此我们要引入指令这项功能

```
168     private fun registerCommands() {
169         registerCommand { this: CommandBuilder
170             name = "hso"
171             alias = listOf("setu")
172             description = "hso plugin management"
173             usage = "[/hso enable] 允许本群发送普通图片\n" +
174                 "[/hso enable r18]允许本群发送r18图片\n" +
175                 "[/hso size x] 将发送的图片大小变为x\n" +
176                 "[/hso anti] 开启反过滤和遁\n" +
177                 "对于色图插件触发色图的词汇请在配置文件更改"
178         onCommand { this: CommandSender
179             val operatingGroup = if (this is ContactCommandSender && this.contact is Group) {
180                 this.contact.id
181             } else {
182                 0
183             }
184             if (it.isEmpty()) {
185                 return@onCommand false
186             }
187             when (it[0]) {
188                 "enable" -> {
189                     if (operatingGroup == 0L) {
190                         return@onCommand false
191                     }
192                     if (it.size == 2 && it[1] == "r18") {
193                         groupsAllowR18.add(operatingGroup)
194                         this.sendMessage("以允许" + operatingGroup + "发送R18的图")
195                     } else {
196                         groupsAllowNormal.add(operatingGroup)
197                         this.sendMessage("以允许" + operatingGroup + "发送R普通的图")
198                     }
199                 }
200                 "size" -> {
201                     if (it.size < 2) {
202                         return@onCommand false
203                     }
204                     val to = try {
205                         it[1].toInt()
206                     } catch (e: Throwable) {
207                         this.sendMessage(it[1] + "无法被转为数字")
208                         return@onCommand false
209                     }
210                     Image_Resize_Max_Width_Height = to
211                     this.sendMessage("更新完成")
212                     this.sendMessage("更新完成")
213                 }
214                 "anti" -> {
215                     Anti_Detect = true
216                     this.sendMessage("更新完成")
217                 }
218                 else -> {
219                     return@onCommand false
220                 }
221             }
222         } return@onCommand true
223     }
```

并在onEnable的时候调用这个方法

```
80     override fun onEnable() {
81         registerCommands()
82         logger.info("hso plugin enabled")
83         subscribeGroupMessages { this: MessageSubscribersBuilder<GroupMessage>
84             (contains(Normal_Image_Trigger)) { this: GroupMessage
85                 if (groupsAllowNormal.contains(this.group.id)) {
86                     sendImage(subject, normal.random())
87                 }
88             }
89             (contains(R18_Image_Trigger)) { this: GroupMessage
90                 if (groupsAllowR18.contains(this.group.id)) {
91                     sendImage(subject, r18.random())
92                 }
93             }
94         } ^subscribeGroupMessages
95     }
```

指令(Command)系统

指令是mirai-console中核心的权限系统之一, mirai-console中的manager(bot主人)系统是独立于mirai-core所存在的, 而一条指令则是 bot主人在任何bot在的地方说的一句以/开头的话或是在console后台输入的指令

指令的存在使得bot的管理变得更加简单

指令结构

/commandName args[0] agrs[1] args[2] args[3]

指令发送者

ContactCommandSender Manager在qq中使用指令

ConsoleCommandSender 后台使用指令

使用sendMessage()会立即发送一句话(如果是qq则会回复, 如果是后台则会打印),
appendMessage()写入一些文字, 会在全部处理结束后一起发送给使用者

指令注册

插件可以注册属于自己的指令, kotlin中应使用PluginBase方法registerCommands{}进行注册
registerCommand {

```
name = "hso"(commandName指令名)
alias = listOf("setu") (command别名)
description = "hso插件总指令"(指令介绍)
usage = "[/hso normal] 允许本群发送普通图片\n" (指令使用方法)
onCommand = { 最高级监听器, return true代表执行顺利, false代表不顺利
    在这个{}里, this为CommandSender it为args[]
}
```

指令的生命周期

当一个指令被使用, 他会首先交给注册该指令的插件处理, 如果该插件返回true, 则代表指令正常, 会再交给所有插件的onCommand()方法监听, 如果返回false, 则不会给其他插件监听的机会, 且会给使用者usage进行帮助

```
81  override fun onCommand(command: Command, sender: CommandSender, args: List<String>) {  
82      if(command.name == "login"){  
83          }  
84      }  
85  }  
86 }
```

任意插件都可以监听“login”指令, 如果登录成功

至此, 涩图插件就可以运行了, 完整源码可以在github中找到

但现在写出的成品只是一个可以加载的插件, 但绝对谈不上是一个好用的插件, 作为一名插件开发者, 你不应该满足于此。

- 1: 图片发送速度极慢
- 2: 图片有色块, 失真现象
- 3: 代码混乱, 不够清晰

针对第一个问题, 我们将图片转为ExternalImage并与数据一起使用kotlin的Channel进行缓存, 这样我们在需要发送图片的时候不需要漫长的等待图片下载

针对第二个问题, 我们将图片使用文件储存, 并将自动压缩关闭

针对第三个问题, 我们将插件分为三个文件, 各司其职

The screenshot shows the IntelliJ IDEA interface with the project 'HealthPro' open. The left sidebar displays the project structure, including 'src/main/kotlin' and 'src/test/kotlin' directories containing various Kotlin files like 'ImageFetcher.kt', 'ImageWriter.kt', and 'ImageReader.kt'. The right side shows the code editor with several tabs open, including 'ImageFetcher.kt', 'ImageWriter.kt', and 'ImageReader.kt'. The code is written in Kotlin and deals with image processing, file I/O, and network requests. The code editor has syntax highlighting and code completion features.

```
1 override fun onCommand(command: Command, sender: CommandSender, args: List<String>) {  
2     if(command.name == "login"){  
3         }  
4     }  
5 }  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122  
123  
124  
125  
126  
127  
128  
129  
130  
131  
132  
133  
134  
135  
136  
137  
138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193  
194  
195  
196  
197  
198  
199  
200  
201  
202  
203  
204  
205  
206  
207  
208  
209  
210  
211  
212  
213  
214  
215  
216  
217  
218  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282  
283  
284  
285  
286  
287  
288  
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316  
317  
318  
319  
320  
321  
322  
323  
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377  
378  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404  
405  
406  
407  
408  
409  
410  
411  
412  
413  
414  
415  
416  
417  
418  
419  
420  
421  
422  
423  
424  
425  
426  
427  
428  
429  
430  
431  
432  
433  
434  
435  
436  
437  
438  
439  
440  
441  
442  
443  
444  
445  
446  
447  
448  
449  
450  
451  
452  
453  
454  
455  
456  
457  
458  
459  
460  
461  
462  
463  
464  
465  
466  
467  
468  
469  
470  
471  
472  
473  
474  
475  
476  
477  
478  
479  
480  
481  
482  
483  
484  
485  
486  
487  
488  
489  
490  
491  
492  
493  
494  
495  
496  
497  
498  
499  
500  
501  
502  
503  
504  
505  
506  
507  
508  
509  
510  
511  
512  
513  
514  
515  
516  
517  
518  
519  
520  
521  
522  
523  
524  
525  
526  
527  
528  
529  
530  
531  
532  
533  
534  
535  
536  
537  
538  
539  
540  
541  
542  
543  
544  
545  
546  
547  
548  
549  
550  
551  
552  
553  
554  
555  
556  
557  
558  
559  
560  
561  
562  
563  
564  
565  
566  
567  
568  
569  
570  
571  
572  
573  
574  
575  
576  
577  
578  
579  
580  
581  
582  
583  
584  
585  
586  
587  
588  
589  
590  
591  
592  
593  
594  
595  
596  
597  
598  
599  
600  
601  
602  
603  
604  
605  
606  
607  
608  
609  
610  
611  
612  
613  
614  
615  
616  
617  
618  
619  
620  
621  
622  
623  
624  
625  
626  
627  
628  
629  
630  
631  
632  
633  
634  
635  
636  
637  
638  
639  
640  
641  
642  
643  
644  
645  
646  
647  
648  
649  
650  
651  
652  
653  
654  
655  
656  
657  
658  
659  
660  
661  
662  
663  
664  
665  
666  
667  
668  
669  
670  
671  
672  
673  
674  
675  
676  
677  
678  
679  
680  
681  
682  
683  
684  
685  
686  
687  
688  
689  
690  
691  
692  
693  
694  
695  
696  
697  
698  
699  
700  
701  
702  
703  
704  
705  
706  
707  
708  
709  
710  
711  
712  
713  
714  
715  
716  
717  
718  
719  
720  
721  
722  
723  
724  
725  
726  
727  
728  
729  
730  
731  
732  
733  
734  
735  
736  
737  
738  
739  
740  
741  
742  
743  
744  
745  
746  
747  
748  
749  
750  
751  
752  
753  
754  
755  
756  
757  
758  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768  
769  
770  
771  
772  
773  
774  
775  
776  
777  
778  
779  
779  
780  
781  
782  
783  
784  
785  
786  
787  
788  
789  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809  
809  
810  
811  
812  
813  
814  
815  
816  
817  
818  
819  
819  
820  
821  
822  
823  
824  
825  
826  
827  
828  
829  
829  
830  
831  
832  
833  
834  
835  
836  
837  
838  
839  
839  
840  
841  
842  
843  
844  
845  
846  
847  
848  
849  
849  
850  
851  
852  
853  
854  
855  
856  
857  
858  
859  
859  
860  
861  
862  
863  
864  
865  
866  
867  
868  
869  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879  
879  
880  
881  
882  
883  
884  
885  
886  
887  
888  
889  
889  
890  
891  
892  
893  
894  
895  
896  
897  
898  
899  
900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
909  
910  
911  
912  
913  
914  
915  
916  
917  
918  
919  
919  
920  
921  
922  
923  
924  
925  
926  
927  
928  
929  
929  
930  
931  
932  
933  
934  
935  
936  
937  
938  
939  
939  
940  
941  
942  
943  
944  
945  
946  
947  
948  
949  
949  
950  
951  
952  
953  
954  
955  
956  
957  
958  
959  
959  
960  
961  
962  
963  
964  
965  
966  
967  
968  
969  
969  
970  
971  
972  
973  
974  
975  
976  
977  
978  
979  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
1000  
1001  
1002  
1003  
1004  
1005  
1006  
1007  
1008  
1009  
1009  
1010  
1011  
1012  
1013  
1014  
1015  
1016  
1017  
1018  
1019  
1019  
1020  
1021  
1022  
1023  
1024  
1025  
1026  
1027  
1028  
1029  
1029  
1030  
1031  
1032  
1033  
1034  
1035  
1036  
1037  
1038  
1039  
1039  
1040  
1041  
1042  
1043  
1044  
1045  
1046  
1047  
1048  
1049  
1049  
1050  
1051  
1052  
1053  
1054  
1055  
1056  
1057  
1058  
1059  
1059  
1060  
1061  
1062  
1063  
1064  
1065  
1066  
1067  
1068  
1069  
1069  
1070  
1071  
1072  
1073  
1074  
1075  
1076  
1077  
1078  
1079  
1079  
1080  
1081  
1082  
1083  
1084  
1085  
1086  
1087  
1088  
1089  
1089  
1090  
1091  
1092  
1093  
1094  
1095  
1096  
1097  
1098  
1099  
1099  
1100  
1101  
1102  
1103  
1104  
1105  
1106  
1107  
1108  
1109  
1109  
1110  
1111  
1112  
1113  
1114  
1115  
1116  
1117  
1118  
1119  
1119  
1120  
1121  
1122  
1123  
1124  
1125  
1126  
1127  
1128  
1129  
1129  
1130  
1131  
1132  
1133  
1134  
1135  
1136  
1137  
1138  
1139  
1139  
1140  
1141  
1142  
1143  
1144  
1145  
1146  
1147  
1148  
1149  
1149  
1150  
1151  
1152  
1153  
1154  
1155  
1156  
1157  
1158  
1159  
1159  
1160  
1161  
1162  
1163  
1164  
1165  
1166  
1167  
1168  
1169  
1169  
1170  
1171  
1172  
1173  
1174  
1175  
1176  
1177  
1178  
1179  
1179  
1180  
1181  
1182  
1183  
1184  
1185  
1186  
1187  
1188  
1189  
1189  
1190  
1191  
1192  
1193  
1194  
1195  
1196  
1197  
1198  
1199  
1199  
1200  
1201  
1202  
1203  
1204  
1205  
1206  
1207  
1208  
1209  
1209  
1210  
1211  
1212  
1213  
1214  
1215  
1216  
1217  
1218  
1219  
1219  
1220  
1221  
1222  
1223  
1224  
1225  
1226  
1227  
1228  
1229  
1229  
1230  
1231  
1232  
1233  
1234  
1235  
1236  
1237  
1238  
1239  
1239  
1240  
1241  
1242  
1243  
1244  
1245  
1246  
1247  
1248  
1249  
1249  
1250  
1251  
1252  
1253  
1254  
1255  
1256  
1257  
1258  
1259  
1259  
1260  
1261  
1262  
1263  
1264  
1265  
1266  
1267  
1268  
1269  
1269  
1270  
1271  
1272  
1273  
1274  
1275  
1276  
1277  
1278  
1279  
1279  
1280  
1281  
1282  
1283  
1284  
1285  
1286  
1287  
1288  
1289  
1289  
1290  
1291  
1292  
1293  
1294  
1295  
1296  
1297  
1298  
1299  
1299  
1300  
1301  
1302  
1303  
1304  
1305  
1306  
1307  
1308  
1309  
1309  
1310  
1311  
1312  
1313  
1314  
1315  
1316  
1317  
1318  
1319  
1319  
1320  
1321  
1322  
1323  
1324  
1325  
1326  
1327  
1328  
1329  
1329  
1330  
1331  
1332  
1333  
1334  
1335  
1336  
1337  
1338  
1339  
1339  
1340  
1341  
1342  
1343  
1344  
1345  
1346  
1347  
1348  
1349  
1349  
1350  
1351  
1352  
1353  
1354  
1355  
1356  
1357  
1358  
1359  
1359  
1360  
1361  
1362  
1363  
1364  
1365  
1366  
1367  
1368  
1369  
1369  
1370  
1371  
1372  
1373  
1374  
1375  
1376  
1377  
1378  
1379  
1379  
1380  
1381  
1382  
1383  
1384  
1385  
1386  
1387  
1388  
1389  
1389  
1390  
1391  
1392  
1393  
1394  
1395  
1396  
1397  
1398  
1399  
1399  
1400  
1401  
1402  
1403  
1404  
1405  
1406  
1407  
1408  
1409  
1409  
1410  
1411  
1412  
1413  
1414  
1415  
1416  
1417  
1418  
1419  
1419  
1420  
1421  
1422  
1423  
1424  
1425  
1426  
1427  
1428  
1429  
1429  
1430  
1431  
1432  
1433  
1434  
1435  
1436  
1437  
1438  
1439  
1439  
1440  
1441  
1442  
1443  
1444  
1445  
1446  
1447  
1448  
1449  
1449  
1450  
1451  
1452  
1453  
1454  
1455  
1456  
1457  
1458  
1459  
1459  
1460  
1461  
1462  
1463  
1464  
1465  
1466  
1467  
1468  
1469  
1469  
1470  
1471  
1472  
1473  
1474  
1475  
1476  
1477  
1478  
1479  
1479  
1480  
1481  
1482  
1483  
1484  
1485  
1486  
1487  
1488  
1489  
1489  
1490  
1491  
1492  
1493  
1494  
1495  
1496  
1497  
1498  
1499  
1499  
1500  
1501  
1502  
1503  
1504  
1505  
1506  
1507  
1508  
1509  
1509  
1510  
1511  
1512  
1513  
1514  
1515  
1516  
1517  
1518  
1519  
1519  
1520  
1521  
1522  
1523  
1524  
1525  
1526  
1527  
1528  
1529  
1529  
1530  
1531  
1532  
1533  
1534  
1535  
1536  
1537  
1538  
1539  
1539  
1540  
1541  
1542  
1543  
1544  
1545  
1546  
1547  
1548  
1549  
1549  
1550  
1551  
1552  
1553  
1554  
1555  
1556  
1557  
1558  
1559  
1559  
1560  
1561  
1562  
1563  
1564  
1565  
1566  
1567  
1568  
1569  
1569  
1570  
1571  
1572  
1573  
1574  
1575  
1576  
1577  
1578  
1579  
1579  
1580  
1581  
1582  
1583  
1584  
1585  
1586  
1587  
1588  
1589  
1589  
1590  
1591  
1592  
1593  
1594  
1595  
1596  
1597  
1598  
1599  
1599  
1600  
1601  
1602  
1603  
1604  
1605  
1606  
1607  
1608  
1609  
1609  
1610  
1611  
1612  
1613  
1614  
1615  
1616  
1617  
1618  
1619  
1619  
1620  
1621  
1622  
1623  
1624  
1625  
1626  
1627  
1628  
1629  
1629  
1630  
1631  
1632  
1633  
1634  
1635  
1636  
1637  
1638  
1639  
1639  
1640  
1641  
1642  
1643  
1644  
1645  
1646  
1647  
1648  
1649  
1649  
1650  
1651  
1652  
1653  
1654  
1655  
1656  
1657  
1658  
1659  
1659  
1660  
1661  
1662  
1663  
1664  
1665  
1666  
1667  
1668  
1669  
1669  
1670  
1671  
1672  
1673  
1674  
1675  
1676  
1677  
1678  
1679  
1679  
1680  
1681  
1682  
1683  
1684  
1685  
1686  
1687  
1688  
1689  
1689  
1690  
1691  
1692  
1693  
1694  
1695  
1696  
1697  
1698  
1699  
1699  
1700  
1701  
1702  
1703  
1704  
1705  
1706  
1707  
1708  
1709  
1709  
1710  
1711  
1712  
1713  
1714  
1715  
1716  
1717  
1718  
1719  
1719  
1720  
1721  
1722  
1723  
1724  
1725  
1726  
1727  
1728  
1729  
1729  
1730  
1731  
1732  
1733  
1734  
1735  
1736  
1737  
1738  
1739  
1739  
1740  
1741  
1742  
1743  
1744  
1745  
1746  
1747  
1748  
1749  
1749  
1750  
1751  
1752  
1753  
1754  
1755  
1756  
1757  
1758  
1759  
1759  
1760  
1761  
1762  
1763  
1764  
1765  
1766  
1767  
1768  
1769  
1769  
1770  
1771  
1772  
1773  
1774  
1775  
1776  
1777  
1778  
1779  
1779  
1780  
1781  
1782  
1783  
1784  
1785  
1786  
1787  
1788  
1789  
1789  
1790  
1791  
1792  
1793  
1794  
1795  
1796  
1797  
1798  
1799  
1799  
1800  
1801  
1802  
1803  
1804  
1805  
1806<br
```

```

HsoProPluginBase.kt (Left):
```kotlin
object HsoProPluginBase : PluginBase() {
 private val config = loadConfig(fileName: "setting.yml")

 val Normal_Image_Trigger by config.withDefaultWriteSave("正常图")
 val R18_Image_Trigger by config.withDefaultWriteSave("不带色")
 var Image_Resize_Max_Width_Height by config.withDefaultWriteSave(1200)
 var Anti_Detect by config.withDefaultWriteSave(false)
 var WorkerSize by config.withDefaultWriteSave(2)
 var messageTemplate by config.withDefaultWriteSave("{image}\n{tags}\n?url")

 val groupsAllowNormal by lazy {
 config.setIfAbsent("Allow_Normal_Image_Groups", listOf())
 }
 val groupsAllowR18 by lazy {
 config.setIfAbsent("Allow_R18_Image_Groups", listOf())
 }

 override fun onLoad() {
 try {
 images = getResourcesConfig(fileName: "data.yml")
 } catch (e: Exception) {
 e.printStackTrace()
 }
 }

 override fun onEnable() {
 logger.info("启动插件")
 val version = images.getConfigSectionInt("version", key: "version")
 val normal = images.getConfigSectionList(key: "normal")
 logger.info("Normal = " + normal.size)
 logger.info("R18 = " + R18.size)
 normalImageProvider = ImageFetcher()
 }
}
```

HsoCommands.kt (Right):
```kotlin
logger.info("Normal.size = " + normal.size)
logger.info("R18.size = " + R18.size)
normalImageProvider = ImageFetcher()
normalImageProvider = ImageFetcher(
 r18CacheSize, workerSize
)

registerCommands()

subscribedGroupMessages (this: MessageSubscribersBuilder<GroupMessage>)
 .contains(Normal_Image_Trigger) { this.groupMessage {
 if (group.id == contains(this.group.id)) {
 launch {
 normalImageProvider.send(subject)
 }
 }
 }
}

contains(R18_Image_Trigger) { this: GroupMessage
 if (groupsAllowR18.contains(this.group.id)) {
 launch {
 r18ImageProvider.send(subject)
 }
 }
} "subscribeGroupMessages"
logger.info("Hso插件已加载")
}

override fun onDisable() {
 logger.info("卸载插件")
 config["Allow_Normal_Image_Groups"] = groupsAllowNormal
 config["Allow_R18_Image_Groups"] = groupsAllowR18
 config.save()
}

override fun onEnable() {
 logger.info("启动插件")
 normalImageProvider = ImageFetcher()
}
```

```

我们的PluginBase则只负责插件的生命周期, 数据管理

```

HsoCommands.kt (Left):
```kotlin
internal fun registerCommands() {
 registerCommand(ContactCommand::class) {
 alias = "f10000"
 desc = "发送普通色图"
 usage = "f10000 [群号] [暗号]"
 operate = "普通色图"
 permission = "普通权限"
 note = "更多权限请见权限配置文件"
 }

 command<GroupCommandSender> {
 alias = "f10001"
 desc = "发送R18色图"
 usage = "f10001 [群号] [暗号]"
 operate = "R18色图"
 permission = "R18权限"
 note = "更多权限请见权限配置文件"
 }

 onCommand<GroupCommandSender> {
 val group = it.operatingGroup ?: if (it is ContactCommandSender) it.contact?.let { Group(it) } else null
 if (group == null) {
 return@registerCommand false
 }
 when (it.type) {
 ContactCommandType.OperatingGroup -> {
 if (it.isOwner) {
 return@registerCommand false
 }
 if (it.isSuperior) {
 return@registerCommand false
 }
 if (it.isOwnerOrSuperior) {
 if (it.superior == null) {
 return@registerCommand false
 }
 HsoProPluginBase.groupsAllowR18.addOperatingGroup(it.superior)
 it.superior.sendMessage("成功添加操作群组")
 } else {
 return@registerCommand false
 }
 }
 else -> {
 HsoProPluginBase.groupsAllowR18.addOperatingGroup(it)
 it.sendMessage("成功添加操作群组")
 }
 }
 return@registerCommand true
 }
}
```

HsoCommands.kt (Right):
```kotlin
registerCommands()
registerCommand<...>()
```

```

而HsoCommands负责Command的注册与逻辑

至此, 涩图插件的第二版写完, 解决了第一版的问题, 带来了很好的用户体验。

(完整源码可以在github找到)

而这还不是完, 最后一步则是书写使用说明, 就想这样(这里只展示了插件配置文件的帮助, 但你应该顺便提供指令帮助等...)

| | |
|--|---|
| <pre> Allow_Normal_Image_Groups: - 655057127 Allow_R18_Image_Groups: [] Normal_Image_Trigger: 色图 R18_Image_Trigger: 不够色 CacheSize: 10 WorkerSize: 2 Image_Resize_Max_Width_Height: 1200 Anti_Detect: false messageTemplate: - {image} {tags} {url} </pre> | <pre> ####允许普通色图的群号 由插件自动生成 也可以自己改 ####允许R18色图的群号 由插件自动生成 也可以自己改 ####使Bot发普通色图的暗号 ####使Bot发R18色图的暗号 ####缓存的色图数量, 缓存并非在内存中, 增加不会导致内存暴涨, 但推荐够用就好 ####用于缓存色图的工作线程数量, 增大数值会提高缓存速度, 增加宽带负载 ####防止图片被检测 会增加内存负载 ####发送出去的图片整体消息, 使用{image}{tags}{url}三个模版物进行替换 </pre> |
|--|---|

完

Citations

mirai-console:<https://github.com/mamoe/mirai-console>
mirai: <https://github.com/mamoe/mirai>

文章作者： mamoe. NaturalHG