第 6 章

マインクラフトのサーバー管理を楽で便利なものにしてみる話

74 回生 日野

6.1　誤植訂正

部誌サンプルのタイトルが「Mineraft のサーバー管理を楽で便利なものにしてみる話」になっていますが、正しくは「マインクラフトのサーバー管理を楽で便利なものにしてみる話」です。お詫びして訂正いたします。

6.2　初めに&自己紹介

初めまして。74 回の日野と申します。この度は第 72 回灘校文化祭・NPCA にお越しいただきありがとうございます。今回の部誌では元あるゲームを改造してみる、ということに関して部誌を書かせてもらいます。

改造するゲームは「マインクラフト」ってやつです。有名ですよね。中一の一年間これに溶かしました。成績が悲惨なことになるので熱中しすぎないようにしましょう。

で、本題の「どのように改造するのか」ですが、マインクラフトのサーバー管理に使える「プラグイン」というやつを作ってみます。(Mod 作成だと思った人、ごめんなさい。どうあがいてもまだ作れそうにありません。) 作るバージョンは 1.11.2 で行きます。

6.3　プラグインって？

いきなりプラグインの話をしても戸惑う人もいるかもしれないので、ブラウザを例にとって説明してみます。

この部誌を手に取ってくれる人はたぶん「Firefox」というウェブブラウザーについて知っていると思います。このブラウザを使う人は多くいます。じゃあなぜこんなに多くの需要があるのか。答えは簡単で、拡張性が高いからです。(拡張機能自体はどのブラウザにも多少は存在するのですが、一番高いのが Firefox です。) きっと拡張性がない Firefox は誰からも使われないでしょう。

で、どのように拡張できるのかというと、

- ダウンロードの管理
- マウスジェスチャーでショートカット
- Web ページから直接動画をダウンロードする

のように、様々です。たぶん僕が知らないだけで他にもいっぱいあるんでしょう。詳細はあんまり書きません。

そしてこの拡張機能を提供してくれるのがプラグインなのです。今は Firefox を例にとって言いましたが、このように拡張性が高いものはブラウザだけではありません。たとえば、プログラムを書く環境であったり、動画や画像の編集ソフトがあったりします。ちなみに今自分はこの部誌を Atom というエディタで書いてるんですが、これも拡張機能つけて便利にしています。そして、一部のゲームはこれらと同様に拡張性が高いのです (拡張性が高いほかのゲームに関してはグーグル先生に聞いてみてください。たぶんいっぱいある)。基本的にシングルプレイメインのゲームは拡張性高いです。

6.4　実際に作ってみる前の下準備

まずは作成に必要な下準備です。

開発環境の DL

個人的には Eclipce がおすすめです。ここから有志のみなさんが作ってくれた日本語化プラグインつきで Eclipce がインストールできます。(プラグインについては分かりますね？ 割といろいろなところで使われてます。) 特に問題がなければ最新バージョンをインストールするので OK です。

マインクラフトのアカウントを用意する

こればっかりはストアで買ってください。間違っても Windows10 版を買わないようにね。最近円に対応して 3000 円固定になりました。ここで購入できます。

マインクラフトのサーバーを用意する

ググってください。解説してるサイトはたくさんあります。解説してると量が膨大に...

6.5　「何もしない」プラグインを作ってみる

まずはプロジェクトを作成します。ファイル→新規→プロジェクトを選択してください。java プロジェクトを選択して、「次へ」をクリックします。プロジェクト名を設定して、完了をクリックしてください。ここではプロジェクト名を `FirstPlugin` にします。

次に、マインクラフトサーバーのプラグインを作成するのに必要な java のビルドパスを通します。次に、パッケージ・エクスプローラーの中の FirstPlugin を右クリックして、プロパティー→ Java のビルドパス→外部 jar の選択をクリックします。出てきたところで使用するバージョン (ここでは 1.11.2) の spigot サーバーを選択します。写真のようになってたら成功です。

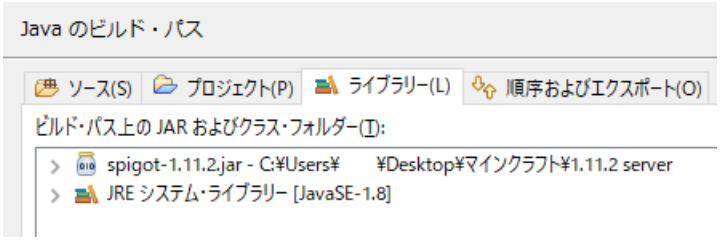

図 6.1: このように表示されていることを確認してください。

java のビルドパスを通せたら、パッケージを作成します。ツリーを開いて、scr のフォルダを右クリックして、新規作成→パッケージを選択してください。パッケージの名前を `first.plugin` にします。

次に、ソースコードを書く場所を作成します。さっき作ったパッケージを右クリックして、新規作成→クラスを選択します。名前はここでは `FirstPlugin` とします。ここで、スーパークラスの参照をクリックします。出てきたウィンドウのボックスに `JavaPlugin` と入力して、でてきたリストの中の `JavaPlugin-org.bukkit.plugin.java` を選択して OK をクリックします。

すると、画像のようにデフォルトでコードがいくつか生成されます。

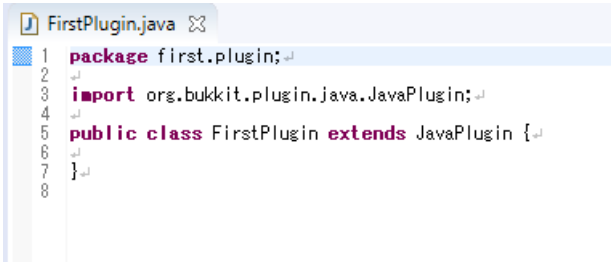

図 6.2: こんな感じです

次に、サーバー起動時と終了時の処理を実装します。ソース→メソッドのオーバーライド/実装を選択し、開いた画面から `onDisable()` と `onEnable()` を選択して、OK を押します。入力されたのがわかりましたか？

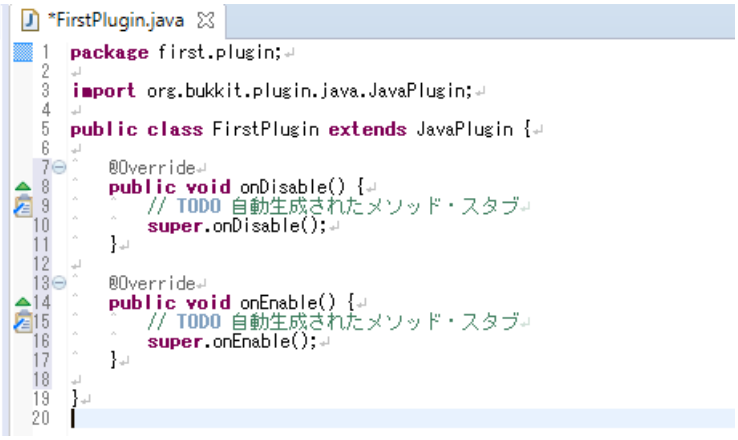

図 6.3: onDisable と onEnable を実装したところ

これで何も機能を持たないプラグインが完成しました。

はい。簡単でしょ？

ちなみに、追加された onDiable() と onEnable() はメソッドと呼ばれます。詳しくはググって。

ただこれだけでは動作しないので、プラグインの仕様書を作る必要があります。プロジェクトエクスプローラーの FirstPlugin を右クリックして、新規→ファイルを選択します。ファイル名に plugin.yml と入力して完了しましょう。テキストエディタが展開します。そうしたら、plugin.yml の中に

```
name: FirstPlugin
version: 1.0.0
main: first.plugin.FirstPlugin
```

と入力します。ここで間違えると動かないので注意しましょう。 plugin.yml の中身についてはあとで解説してます。

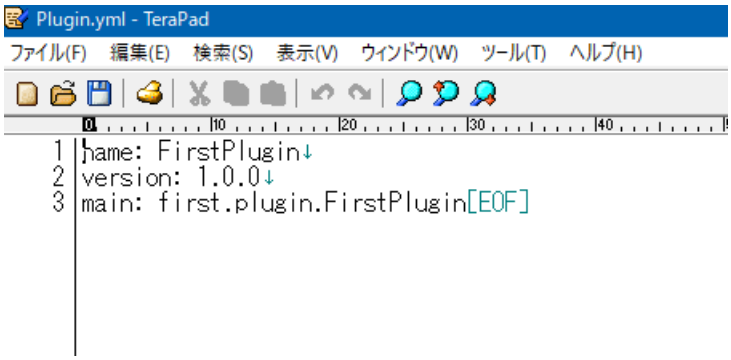

図 6.4: イメージ

こんな感じです。

次にプログラムをコンパイルします。FirstPlugin を右クリック→エクスポートを選択してください。JAR ファイルを選択して次へをクリックします。

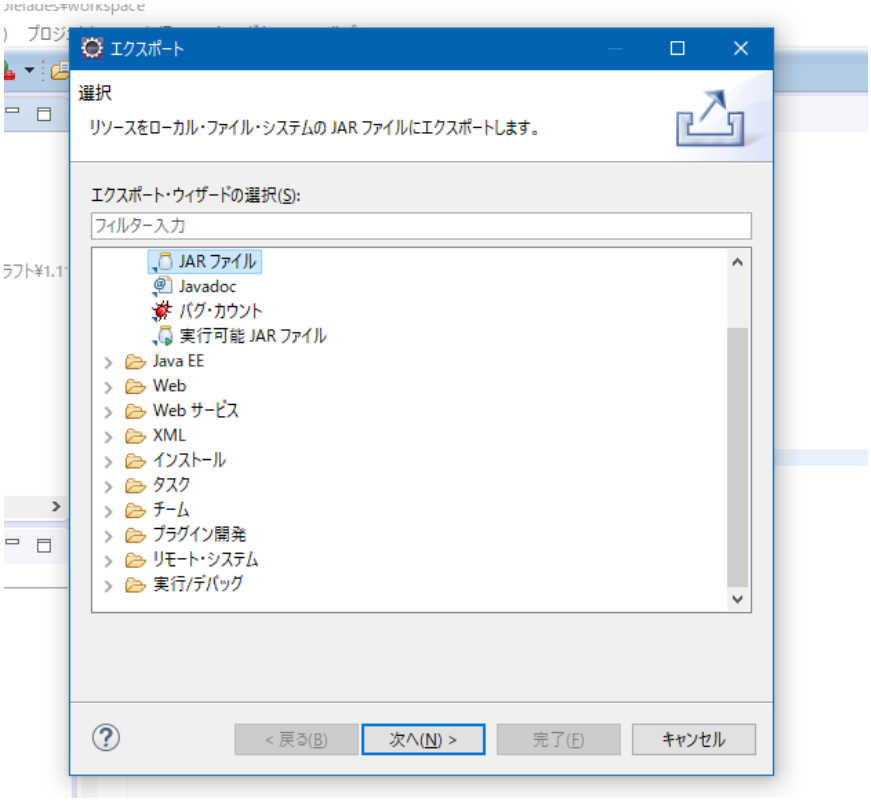

図 6.5: JAR ファイルを選択

次に、この画面で.classpath と.project からチェックを外して、plugin.yml にチェックを入れてください。エクスポートは用意しているサーバーの plugin フォルダに直接してしまいましょう。名前は FirstPlugin.jar にします。この時、エクスポート先に FirstPlugin.jar まで書くようにしてください。画像を参考にしてくださいね。

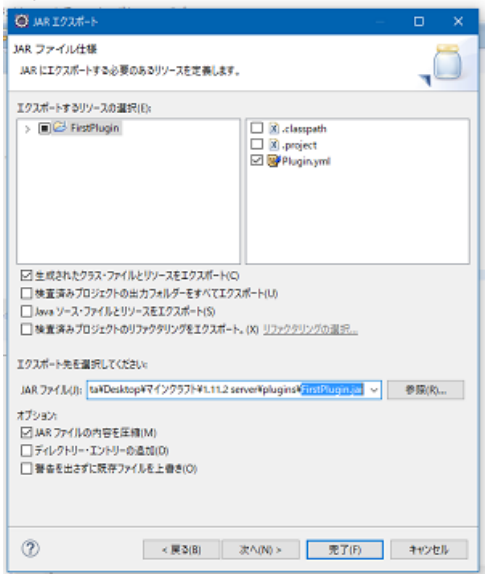

図 6.6: エクスポート設定

エクスポートできたらサーバーを起動して、pl コマンドを実行してみましょう。緑色で FirstPlugin と表示されていたら成功です!

6.6　コマンドを作る

マインクラフトをプレイしたことがある人なら、一度はマルチプレイのサーバーにつないだことがあると思います。その時に、サーバー独自のコマンドを使ったことはありませんか？ 実はそのコマンドたちも、プラグインによって実装されています。ここでは、コマンドを実装してみます。

クラスの実装

■コラム: クラスとは？

マインクラフトのプラグインを作ってく上で、クラスの概念は重要です。マインクラフトのプラグインでは、コマンドや動作ごとに行う処理一つ一つをそれぞれのクラスを作成して行っています。下の画像のようなイメージです。

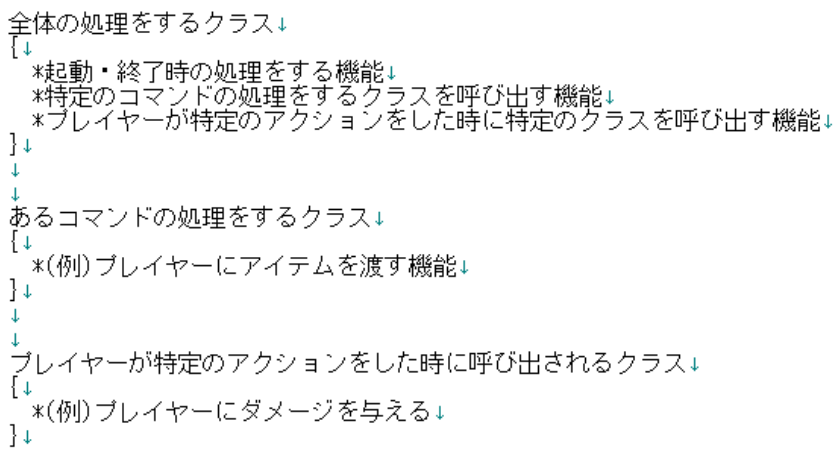

図 6.7: クラスのイメージ画像

クラスの説明が終わったところでクラスを実装してみます。まずは、onEnable メソッドに `getCommand("first").setExecutor(new FirstCommand());` と入力してください。これは、「first ってコマンドが入力されたら Firstcommand ってクラスで対応してね」という意味です。すると、`new FirstCommand` に赤線が引かれると思います。これは、「FirstCommand ってなんやねんワイ知らんわ」という意味です。なので、FirstCommand というクラスを作成する必要があります。その赤線にマウスを合わせて、クラスを作成を押してください。出てきた画面では設定を変えずに作成してください。これで、`first` というコマンドを処理する専用のクラスができました。

因みに、getCommand の setExcutor の部分を `this` に設定すると、getCommand を記述したクラスで処理することができます。が、クラスのインポートが必要になりますし、何より見た目が汚くなります。お勧めしません。this に設定する時には、`public class FirstPlugin extends JavaPlugin` を `public class FirstPlugin extends JavaPlugin implements CommandExecutor` に手動で変更してください。クラスを新規作成した時には自動で追加されるので問題ありません。

onCommand メソッドの役割

さて、クラスを生成することはできましたが、この自動で入力された onCommand とは何でしょうか？ その役割を見ていきましょう。ちなみにこれもメソッドですよ〜。

onCommand メソッドの引数

onCommand メソッドの引数を見てみると `sender,command,label,args[]` の 4 種類が存在します。一つずつ紹介していきます。

sender

コマンドを使った人の情報が入ります。使うには Player 型にキャストしなければいけません。

command

コマンドの情報が入ります。基本使わないかな？

label と args[]

`/test a b c` というコマンドがあったとします。この時、label に `a` が代入されて、args[0] に `b`、args[1] に `c` が代入されます。これでわかるよね？

因みに、クラス作成した直後は `arg0,arg1,arg2,arg3[]` になっています。上で紹介した `sender,command,label,args[]` のほうが簡単なので、書き換えてください。この部誌の中では `sender,command,label,args[]` で説明します。

```
package first.plugin;

import org.bukkit.command.Command;

public class FirstCommand implements CommandExecutor {

    @Override
    public boolean onCommand(CommandSender arg0, Command arg1, String arg2, String[] arg3) {
        return false;
    }

}
```

図 6.8: 作成したすぐのクラス。変数名をそれぞれ変更することを推奨します。

onCommand メソッドの返り値

onCommand メソッドの返り値は boolean 型です。true の時には特に何も起こりませんが、false の時にはコマンドを送った人に、あとで紹介する plugin.yml で設定したメッセージを送信します。

onCommand で何か処理してみよう

せっかくクラスとメソッドを作ったのに何もしないのはもったいないので、何か処理を実装してみましょう。ここでは、インベントリをダイヤブロックで満たしてみたいと思います。

ここで、コマンドの送信者について考えてみましょう。もし、このコマンドがコンソールから実行されていたらどうなると思いますか？ 当然コンソールにアイテムを渡すことができずに、処理に失敗してしまします。無理やり実行するとこのようにエラーが発生します。

```
>first
[11:02:01 WARN]: Unexpected exception while parsing console command "first"
org.bukkit.command.CommandException: Unhandled exception executing command 'first' in plugin FirstPlugin v1.0.0
        at org.bukkit.command.PluginCommand.execute(PluginCommand.java:46) ~[spigot-1.11.2.jar:git-Spigot-3fb9445-6e3cec8]
        at org.bukkit.command.SimpleCommandMap.dispatch(SimpleCommandMap.java:141) ~[spigot-1.11.2.jar:git-Spigot-3fb9445-6e3cec8]
        at org.bukkit.craftbukkit.v1_11_R1.CraftServer.dispatchCommand(CraftServer.java:690) ~[spigot-1.11.2.jar:git-Spigot-3fb9445-6e3cec8]
        at org.bukkit.craftbukkit.v1_11_R1.CraftServer.dispatchServerCommand(CraftServer.java:636) [spigot-1.11.2.jar:git-Spigot-3fb9445-6e3cec8]
        at net.minecraft.server.v1_11_R1.DedicatedServer.aM(DedicatedServer.java:437) [spigot-1.11.2.jar:git-Spigot-3fb9445-6e3cec8]
        at net.minecraft.server.v1_11_R1.DedicatedServer.D(DedicatedServer.java:400) [spigot-1.11.2.jar:git-Spigot-3fb9445-6e3cec8]
        at net.minecraft.server.v1_11_R1.MinecraftServer.C(MinecraftServer.java:678) [spigot-1.11.2.jar:git-Spigot-3fb9445-6e3cec8]
        at net.minecraft.server.v1_11_R1.MinecraftServer.run(MinecraftServer.java:576) [spigot-1.11.2.jar:git-Spigot-3fb9445-6e3cec8]
        at java.lang.Thread.run(Unknown Source) [?:1.8.0_102]
Caused by: java.lang.ClassCastException: org.bukkit.craftbukkit.v1_11_R1.command.ColouredConsoleSender cannot be cast to org.bukkit.entity.Player
        at first.plugin.FirstCommand.onCommand(FirstCommand.java:14) ~[?:?]
        at org.bukkit.command.PluginCommand.execute(PluginCommand.java:44) ~[spigot-1.11.2.jar:git-Spigot-3fb9445-6e3cec8]
        ... 8 more
```

図 6.9: console に無理やりアイテムを渡そうとしたらこうなる

なので、コマンドを実行した人にアイテムを渡す前に、コマンドを実行したのがコンソールかプレイヤーかを確認する必要があります。Java で、変数の型を比較するには`"instanceof"`を使用します。sender が Player 型の変数ではなかったときは sender がコンソールなので、return false で終了させてしまいましょう。実際に書くコードはとなります。

次に、コマンドを送信したプレイヤーの情報を取得して、Player 型にキャストします。`Player player` で、プレイヤー情報を保存する変数を作成できます。次に、プレイヤー型にキャストした sender の情報を player に代入します。`player = (Player)sender` でプレイヤー型にキャストすることができます。

ここで、あれ？ と思った人が多いと思います。なんたって、Player の下に赤線が引かれてますから。これは、「Player 型ってなんやねんワイ知らんぞ」とコンパイラが言っているのです。なので、Player 型を処理するパッケージをインポートする必要があります。Ctrl+Shift+O を押してください。赤線がなくなったら OK です。

次にプレイヤーに渡すアイテムを表す変数を作成します。。まず、ItemStack 型の変数 item を作成します。この ItemStack 型もコンパイラは知らないので、Ctrl+Shift+O で ItemStack 型をインポートしてください。この時、2 つの選択肢を提示されますが、`org.bukkit...` を選択してください。

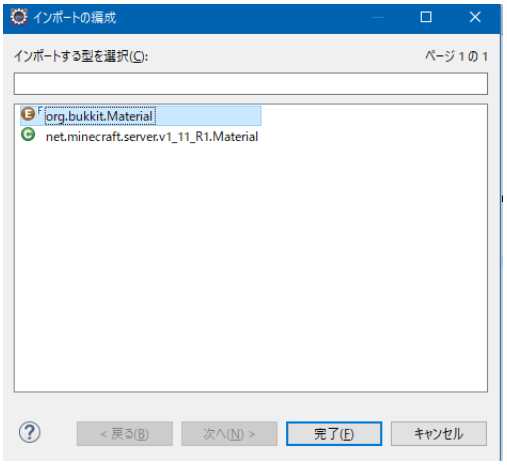

図 6.10: こっちを選択してください

次に、item にダイヤモンドというデータを代入します。アイテムデータは ItemStack を使用することで取得できます。あとこの new をくっつけることでインスタンスの生成をすることができます。インスタンスの生成に関してはググってください。実装すると、下のようになります。

ダイヤモンドのデータを持つ変数の作成

```
ItemStack item;
item = new ItemStack(Material.DIAMOND);
```

次に、コマンドの実行者のインベントリを取得します。。Inventory 型の変数 inv を作成して、player のインベントリを保存します。またいつものように赤線が引かれるのでインポートしてください。次に、送信者のインベントリデータを inv に代入します。player のインベントリデータは `player.getInventory()` で取得できます。

インベントリの取得

```
Inventory inv ;
inv = player.getInventory();
```

最後に、インベントリにアイテムを追加する処理です。インベントリにアイテムを追加する処理は `addItem()` と `setItem()` の 2 種類が存在します。addItem はプレイヤーのインベントリに場所を指定せずにアイテムを追加します。アイテムを投げられる、とするとイメージがわきやすいと思います。それに対して、setItem ではインベントリの指定した場所にアイテムを追加します。もとからあったアイテムは上書きされます。今回は、インベントリをダイヤブロックで満たすので、setItem を使用します。setItem の書式は `setItem(インベントリ番号, アイテム)` です。インベントリ番号は、インベントリスロット一つ一つに割り当てられています。割り当てのイメージは下の画像のような感じです。

図 6.11: 番号割り振りのイメージ　ダイヤブロックの数が番号を表す

今回は、0~35 のスロットにダイヤモンドをセットします。36 回同じ処理を書くのは面倒なので、For 文を使ってみましょう。
プレイヤーにアイテムを渡す処理は以下のようになります。

プレイヤーにアイテムを渡す

```
for(int i=0;i<36;i++){
    inv.setItem(i,item);
}
```

最後に、コマンドを実行したプレイヤーにメッセージを送信してみる機能を実装してみましょう。(プレイヤー型の変数).sendMessage("メッセージの内容") で、変数で指定されたプレイヤーにメッセージを送ることができます。player にキャストして代入したのなら、player.sendMessage("メッセージ内容") で送信できます。
さあ、これでコマンドが完成しました。

```
package first.plugin;

import org.bukkit.Material;
import org.bukkit.command.Command;
import org.bukkit.command.CommandExecutor;
import org.bukkit.command.CommandSender;
import org.bukkit.entity.Player;
import org.bukkit.inventory.Inventory;
import org.bukkit.inventory.ItemStack;

public class FirstCommand implements CommandExecutor {

    @Override
    public boolean onCommand(CommandSender sender, Command command, String label, String[] args) {
        if(!(sender instanceof Player)) return false;

        Player player=(Player)sender;

        ItemStack item=new ItemStack (Material.DIAMOND_BLOCK);

        Inventory inv;
        inv = player.getInventory();

        for(int i=0;i<36;i++){
            inv.setItem(i,item);
        }
        player.sendMessage("コマンドが実行されました");

        return true;
    }

}
```

図 6.12: ここまで

■コラム: 変数の初期化

上に示したコードでは変数を作成した後に代入していますが、作成するのと同時に初期化してもかまいません。例えば Player 型の変数を作成するときに `Player player = (Player)sender` のようにすることができます。これより後ではこの方法を使っている前提で説明します。====[/column]

コマンド作成時に必要な plugin.yml への記載

ただ、これだけだとコマンドは実行されません。Enable 時に plugin.yml を通じてコマンドの設定をサーバーに読み込ませる必要があります。

まず、plugin.yml を開きます。開いたら、次の画像のように入力してください。commands:の後が追加されています。空白の部分はすべて**半角スペース 2 つ**で開けてください。タブではないです。

```
name: FirstPlugin
version: 1.0.0
main: first.plugin.FirstPlugin
commands:
  first:
    description: 初めてのコマンド作り
    usage: /<command>
    default: true[EOF]
```

図 6.13: plugin.yml のイメージ

■コラム: plugin.yml の役目とそれぞれの役割

plugin.yml は、サーバーにプラグインの情報を伝える役目をしています。それぞれの設定のセットの役割を確認してみましょう。

サーバーの基本設定

name

必須。プラグインの名前です。pl コマンドを実行したときに表示されたり、plugin フォルダの下に生成されるフォルダ名に適用されたりします。jar ファイルの名前と揃えることが推奨。

version

必須。/versino <プラグイン名>が実行されたときに表示されます。バージョン管理番号です。任意の文字列が使用可能です。外部に公開したりするときには真面目に設定したほうがいいです。まあ個人で使う分には 0.0.0 とか 1.0.0 でいいでしょう。

main

必須。プラグインのメインクラスの場所を記述します。パッケージ名を含めて指定してください。

description

/version <プラグイン名>が実行されたときに表示されます。必須ではないですが、説明なのでつけておく価値はあります。

commands

プラグインによって追加されるコマンドの設定を記述します。

commands の設定

<コマンド名>

コマンド名を記述します。コマンドはスラッシュ (/) を省いた状態で記述してください。

description

コマンドの説明文です。help コマンドを実行したときに表示されます。

usage

onCommand メソッドが false を返した時に表示されます。<command>を使うと、実際にメッセージが表示されたときに<command>の部分が初めに設定したコマンド名に置き換わります。

default

権限を設定します。permission でも同じことをしてくれますが、こちらは permission の簡易版です。default の後に true(全員が使用可能),false(全員が使用不可能),op(OP のみ使用可能),notop(OP のみが使用不可能) を指定することができます。

permission の設定周りはここでは扱いません。探したらわかりやすいサイトがたくさん出てくると思います。

plugin.yml の編集が終わったら、前回と同じようにしてエクスポートしてください。上書きしてもらって構いません。

サーバーを起動して、pl コマンドを実行して、表示されることを確認してください。プラグインが表示されたら、マインクラフトを起動してサーバーにログインして、コマンドを実行してみてください。

図 6.14: 実行結果

インベントリがダイヤブロックで満たされたら成功です!

6.7　config ファイルを作ってみる

マインクラフトのプラグインを使ったことがある人は、コンフィグファイルを使ったことがあると思います。ここでは、コンフィグ周りの設定方法を解説したいと思います。

config.yml の作成

まず最初に元となるコンフィグを作成したいと思います。plugin.yml と同じようにして config.yml を作成してしまいましょう。中身はここでは画像のようにしておきます。

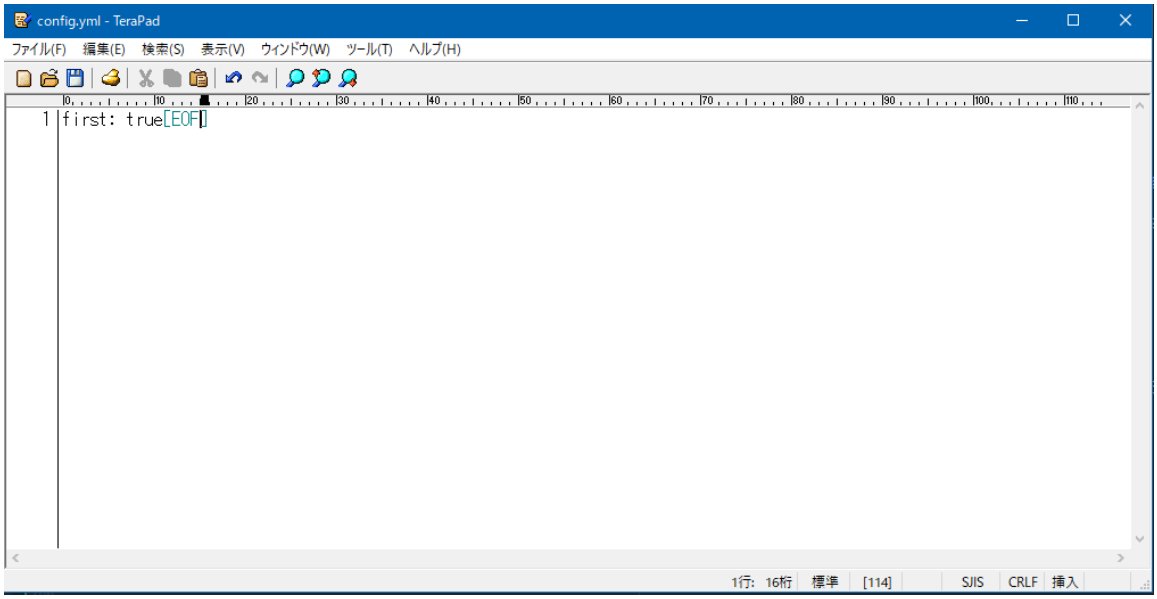

図 6.15: config.yml の中身はこのようにしておきます。

config.yml をサーバーフォルダに保存させる

config ファイルは起動時に読み込んでおきたいので、onEnable メソッドに記述します。

plugin/<プラグイン名>のフォルダに config.yml を保存させるには、`saveDefaultConfig();` を使います。この時、このクラスで処理することを明示するために this を付けて `this.saveDefaultConfig();` とします。

saveDefaultConfig() の働き方は、config.yml が存在しているか存在していないかでわけられます。もし config.yml が存在しないときには、plugin/<プラグイン名>の中に jar ファイルの中の config.yml と同じものを作成し、config.yml が存在するときには何もしません。つまり、上書きされることがない、ということです。

config データを取り込む

マインクラフトのプラグインには `FileConfiguration` という型が存在します。ここでは、config という名前の変数を作成します。御多分に洩れず、こんな型知らねーよと帰ってくるのでインポートしてやってください。この型に対して `getConfig()` の戻り値を代入すると config データを取得できます。2 つ合わせて `FileConfiguration config = getConfig()` となります。

次に、一つ一つの項目の取り込みです。例を使って説明します。getConfig で取り込んだ config ファイルの中に、<設定名>: <設定内容 (TRUE/FALSE)>というコンフィグファイルがあったとします。この時、<設定内容>を取得するには、`config.getBoolean("設定名");` を使うことで取得できます。この `getBoolean` の部分は、<設定内容>の方によって変更可能です。ここでは true と false の判別を付けるために `getBoolean()` を使用します。取得したコンフィグの内容を保存するために変数を一つ作りましょう。ここでは名前を `config_test` とします。変数の初期化と同時に取得するのであれば、`boolean config_test = config.getBoolean("first");` のようになります。

> ■コラム: 使用可能な get****() シリーズの紹介
>
> get****() シリーズには多くの種類があって、`getBoolean(String)`,`getInt(String)`,`getString(String)` などの簡単に想像できるものから、階層でリストアップされているものすべてを取得する `getList(String)`,`getStringList(String)` のようなものもあります。

これでコンフィグデータを取り込むことはできました。あとは getLogger を使用して Enable 時に true と false1 を表示する処理を実装してみましょう。

true か false かを取得して出力する

```
if(config_test){
    getLogger().info("TRUE");
}else{
    getLogger().info("FALSE");
}
```

ここまでのソースコードをまとめるとこうなります。

onEnable メソッドの中

```
this.saveDefaultConfig();

FileConfiguration config = getConfig();

boolean config_test = config.getBoolean("first");

if(config_test){
    getLogger().info("TRUE");
}else{
    getLogger().info("FALSE");
}
```

ここまでできたら、エクスポートして実行してみましょう。このときに、エクスポートするフォルダに config.yml を含めてください。

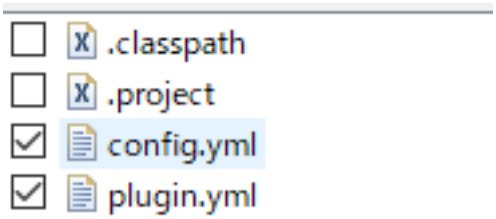

図 6.16: エクスポートするフォルダ

エクスポートできたら、サーバーを起動してください。サーバーログに TRUE と出てきましたか？

図 6.17: TRUE と出てくる

確認したら、サーバーを一度止めて、plugin/firstplugin の中の config.yml の `first:  true` を `first:  false` に書き換えてください。書き換えた後でサーバーを起動してください。すると...?

図 6.18: FALSE と出てくる

先ほどと違って false と出てきました。成功ですね。

6.8 おわりに

いかがだったでしょうか？ 僕は中 3 なのでこれが初めて部誌を書く経験になります。難しいですね... 部誌書くの...。とても拙く不自由な日本語だったと自分でも思います。来年こそはもっとうまく書くぞ。

この部誌を読んでくれた人が一人でも「マインクラフトのプラグインを作ること」に興味を持ってくれるとうれしいです。