

Java 專案：物品

TYIC 桃高資訊社

物品和物品堆疊

物品(`item`)是構成 Minecraft 很重要的部分

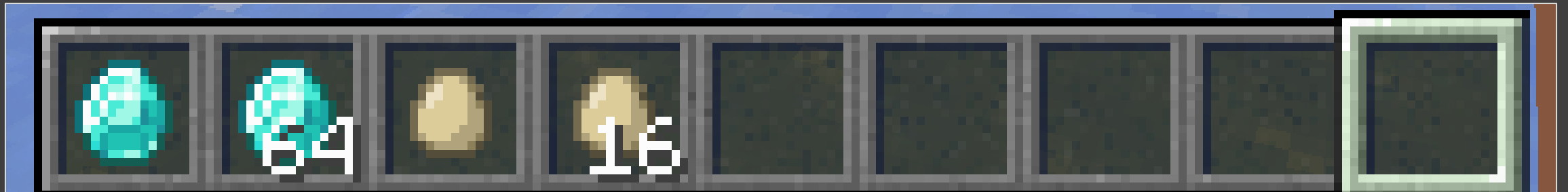
物品的類別為 `net.minecraft.item.Item`

物品堆疊(`item stack`)則是代表一種物品及擁有的數量(`count`)

其類別為 `net.minecraft.item.ItemStack`

如快捷欄(`hotbar`)的前四格(`slot`)為 4 個不同的物品堆疊

但前兩格物品堆疊的物品都是鑽石，後兩格物品堆疊的物品都是雞蛋



註冊表

幾乎所有東西都要向註冊表(`registry`)註冊(`register`)

如物品就需要註冊，但物品堆疊不需要註冊

其目的是為了讓遊戲知道有這東西，以便進行其他處理

每種註冊類別都有獨立的註冊表，並分為靜態註冊表和動態註冊表

靜態註冊表用於註冊永遠不變的項目

只能在遊戲初始化階段註冊項目

動態註冊表用於註冊可能會變更的項目

可以在任何時候註冊項目

註冊表為實作 `net.minecraft.registry.Registry<T>` 介面的類別

註冊表

註冊表中，不同項目會有唯一的註冊鍵(`registry key`)

用於區別及檢索註冊表中的不同項目

欲向註冊表註冊項目

需呼叫 `Registry<T>` 介面的靜態方法 `register`

該方法會返回註冊的項目

所有靜態註冊表

位於 `net.minecraft.registry.Registeries` 類別內

標識符

標識符(`identifier`，簡稱 `id`)

由命名空間(`namespace`)和路徑(`path`)組成

其中命名空間通常為模組 `id`

而路徑則由小寫英文、下滑線(`_`)、斜線(`/`)組成

在遊戲各處都會使用到標識符

標識符以字串表示為 `"namespace:path"`

如雞蛋物品的標識符以字串表示為 `"minecraft:egg"`

其在 `Java` 中處理的類別為 `net.minecraft.util.Identifier`

該類別的建構子為 `private`，需使用靜態方法

`of(String namespace, String path)` 創建實例

註冊鍵

註冊鍵(**registry key**)由註冊表 and 值組成

其中的註冊表 and 值為標識符

註冊鍵以字串表示為 **"ResourceKey[registry/value]"**

如雞蛋物品的註冊鍵以字串表示為

"ResourceKey[minecraft:item/Minecraft:egg]"

註冊鍵除了註冊表相關事項之外幾乎不會使用到

註冊鍵在 **Java** 中處理的類別為

net.minecraft.registry.RegistryKey<T>

該類別的建構子為 **private**，需呼叫靜態方法 **of** 創建實例

註冊表的註冊鍵位於 **net.minecraft.registry.RegistryKeys**

物品類別

Item 類別只有一個建構子 **Item(Item.Settings settings)**

Item.Settings 是一個用來控制物品行為的類別

如最大堆疊大小(**max stack size**，預設為 **64**)、

冷卻時間(**cooldown**，預設無冷卻時間)、註冊鍵(預設為空)等

其可以通過方法鏈式呼叫(**method chaining**)進行設定

用於註冊物品時，一定要設定註冊鍵，範例如下：

```
RegistryKey<Item> registryKey =  
    RegistryKey.of(RegistryKeys.ITEM, Identifier.of(TyicMod.MOD_ID, "example"));  
Item.Settings settings =  
    new Item.Settings().useCooldown(1.5f).maxCount(2).registryKey(registryKey);
```

java

基本物品

最簡單的物品就是直接創建一個 **Item** 類別的實例並向遊戲註冊
但為了邏輯分離，因此將物品相關事項放在 **item** 套件下
物品註冊事項放在該套件下的 **ModItems** 類別
並將註冊過程做成一個函式：

1. 設定 **Item.Setting** 實例的註冊鍵
2. 向註冊表註冊項目，並將其返回值(註冊項目)作為函式返回值
之後若要引用該物品，如用於比較，只需使用此返回值即可
故註冊後會儲存在公開靜態欄位，方便之後使用

基本物品

ModItems.init()

方法只是為了

加載 ModItems 類別

```
package org.tyic.tyicmod.item;
```

```
import (...)
```

```
public class ModItems {
```

```
    public static final Item TYIC_LOGO = register("tyic_logo", Item::new, new Item.Settings());
```

```
    public static Item register(String id, Function<Item.Settings, Item> itemFunction, Item.Settings settings) {  
        RegistryKey<Item> registryKey = RegistryKey.of(RegistryKeys.ITEM, Identifier.of(TyicMod.MOD_ID, id));  
        return Registry.register(Registries.ITEM, registryKey, itemFunction.apply(settings.registryKey(registryKey)));  
    }
```

註冊表

註冊鍵

設定註冊鍵後創建該物品的實例

```
    public static void init() {
```

```
        TyicMod.LOGGER.info("Registering mod items.");
```

```
    }
```

```
}
```

ModItems.java

```
package org.tyic.tyicmod;
```

```
import (...)
```

```
public class TyicMod implements ModInitializer {  
    public static final String MOD_ID = "tyicmod";
```

```
    public static final Logger LOGGER = LoggerFactory.getLogger(MOD_ID);
```

```
    @Override
```

```
    public void onInitialize() {
```

```
        LOGGER.info("Initializing Tyic Mod.");
```

```
        ModItems.init();
```

```
    }
```

```
}
```

TyicMod.java

實際測試

實際打開遊戲測試，會發現無法在
創造模式物品欄(creative tab、item group)找到該物品
這是因為我們並未添加該物品到創造模式物品欄
但仍可使用指令(command)直接獲取，使用 give 指令：

指令名稱 實體選擇器，@s 代表指令執行者，@a 代表全部玩家

/give @s tyicmod:tyic_logo mccmd

斜線開頭代表指令 物品 id

/give @s tyicmod:tyic_logo

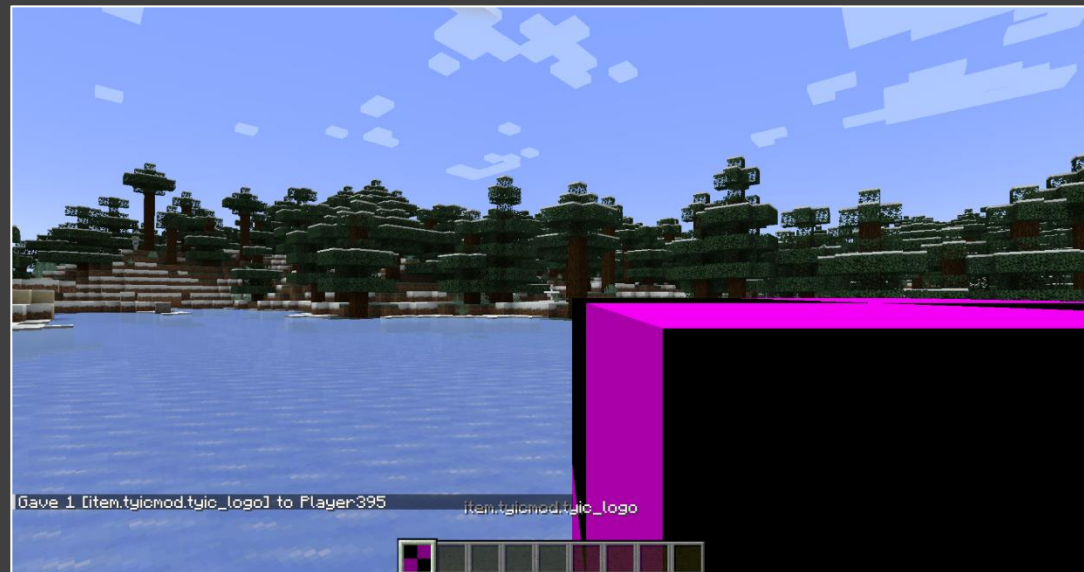
直接按下斜線(/)即可開始輸入指令，或打開文字聊天輸入
輸入指令時，可按 Tab 進行自動補全
關於更多指令的用法及資訊

請參考維基百科(<https://zh.minecraft.wiki/w/%E5%91%BD%E4%BB%A4>)

或指令教學影片(<https://youtube.com/playlist?list=PLbMLilemgCLY0bENyaNE-pXZoIy6PYwLS>)

實際測試

取得物品後可能會發現
其外觀非常奇怪
且名稱(name)也非常奇怪
這是因為我們並沒有設定
該物品的紋理(texture)及名稱
但我們確實成功製作了一個新物品



紋理

紋理(texture)就是遊戲中一切顯示的圖片

包含物品的外觀等幾乎一切的可視元素

所有紋理都被放置在 `assets/namespace/textures`

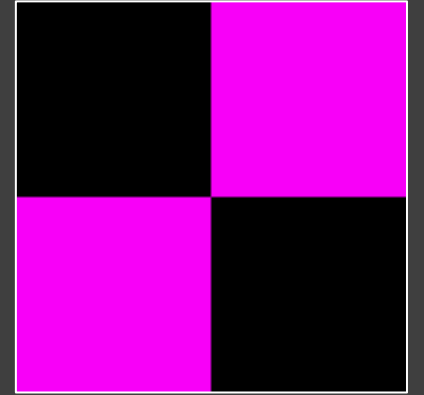
且其下方設有許多子資料夾區分不同紋理種類

強烈建議參考及模仿原版資源分類

紋理圖片皆採用 `png` 格式(`.png` 檔案)

若遊戲無法找到紋理，便會使用右方無效紋理

更多紋理資訊請參考[維基百科](https://zh.minecraft.wiki/w/%E7%BA%B9%E7%90%86)(<https://zh.minecraft.wiki/w/%E7%BA%B9%E7%90%86>)



紋理繪製軟體

若想要自行繪製紋理

雖然理論上任何繪圖軟體，包含小畫家，皆可用來繪製
但有幾個較為推薦的繪製軟體：

1. BlockBench：一個開源像素編輯器，極度適合 **Minecraft**

官方網站：<https://www.blockbench.net/>

2. Gimp：一個廣為人知的開源圖片編輯器，功能極多

官方網站：<https://www.gimp.org/>

免費紋理

若不想自行繪製紋理

可以使用此免費紋理：

<https://github.com/malcolmriley/unused-textures>

線上查看該庫所有紋理：

<https://oparisblue.github.io/minecraft-textures-viewer/#github/malcolmriley/unused-textures/master>

也可自行從網上尋找其他免費紋理，但須注意版權

物品紋理

物品紋理放置在 `assets/namespace/textures/item`

物品紋理的圖片長寬比應為 `1 : 1`，且像素應為 `2` 的次方數
但不建議超過 `1024x1024`，原版物品紋理則為 `16x16`

通常會將物品紋理圖片檔名取為物品名稱

範例紋理如右

檔名：`tyic_logo.png`

像素：`512x512`



模型

紋理決定了樣式

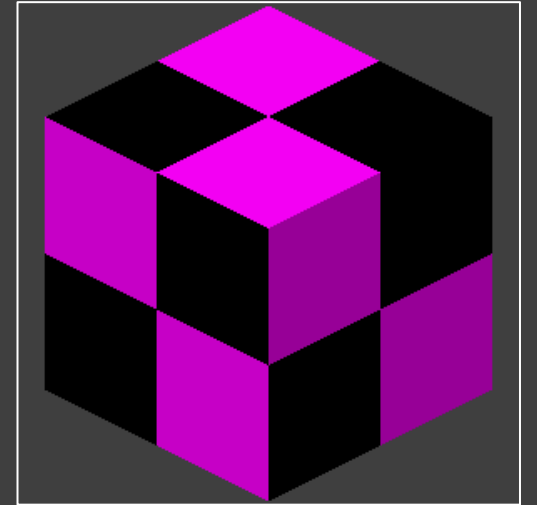
而模型(model)決定了要用哪種紋理和怎麼顯示
所有模型都被放置在

assets/namespace/models

模型皆為 json 檔案

若遊戲無法找到模型，便會使用右方無效模型

更多模型資訊請參考維基百科(<https://zh.minecraft.wiki/w/%E6%A8%A1%E5%9E%8B>)



物品模型

物品模型放置在 `assets/namespace/models/item`

物品模型的常見格式如左下

其中 `layer0` 的值為紋理的標識符

即為 `"namespace:path_to_texture"`

範例模型 `tyic_logo.json` 如右下

```
{
  "parent": "minecraft:item/generated",
  "textures": {
    "layer0": "namespace:path_to_texture"
  }
}
```

json

```
{
  "parent": "minecraft:item/generated",
  "textures": {
    "layer0": "tyicmod:item/tyic_logo"
  }
}
```

tyic_logo.json

物品模型映射

模型決定了要用哪種紋理

而物品模型映射(item models definition)

決定了物品要用哪種模型

所有物品模型映射都被放置在

`assets/namespace/items`

物品模型映射皆為 `json` 檔案

且檔案名稱和物品 `id` 需相同

物品模型映射的常見格式如右上

其中 `model` 的值為模型的標識符

即為 `"namespace:path_to_model"`

範例物品模型映射 `tyic_logo.json` 如右下

```
{
  "model": {
    "type": "minecraft:model",
    "model": "namespace:path_to_model"
  }
}
```

json

```
{
  "model": {
    "type": "minecraft:model",
    "model": "tyicmod:item/tyic_logo"
  }
}
```

tyic_logo.json

實際測試

當遊戲要渲染一個物品時
便會套用與物品 **id** 相同的物品映射模型
而物品映射模型則會決定要使用哪個模型
模型則會決定要使用和怎麼使用哪個紋理
最終渲染出物品的外觀

為新物品設定此三項之後
打開遊戲便能看見新物品有了想要外觀！



國際化與在地化

剛剛物品名稱的 `"item.tyicmod.tyic_logo"`

其實是翻譯鍵名(`translation key`)

用途是在使用者選擇不同語言時，便有不同的翻譯

這稱為國際化(`internationalization`，簡稱 `i18n`)

而將翻譯鍵名映射(翻譯)到各語言文字

這稱為在地化(`localization`，簡稱 `l10n`)

當遊戲找不到翻譯時，便會直接使用翻譯鍵名

物品預設的翻譯鍵名為 `"item.namespace.id"`

國際化與在地化

所有的在地化檔案都被放置在 `assets/namespace/lang` 命名空間為何無任何影響

在地化檔案皆為 `json` 檔案，且檔案名稱和語言代號需相同

該 `json` 檔案為一個物件，鍵為翻譯鍵名，而值為翻譯的字串

常用語言代號有 `en_us`(English(US))、`zh_tw`(繁體中文(台灣))

所有語言代號請參考維基百科(<https://zh.minecraft.wiki/w/%E8%AF%AD%E8%A8%80>)

範例在地化檔案 `en_us.json` 和 `zh_tw.json` 如下

```
{  
  "item.tyicmod.tyic_logo": "TYIC Logo"  
}
```

en_us.json

```
{  
  "item.tyicmod.tyic_logo": "TYIC 標誌"  
}
```

zh_tw.json

實際測試

為新物品設定在地化後
打開遊戲便能發現新物品
在特定語言下有了想要的名稱

右上為 **en_us**(English(US))
右下為 **zh_tw**(繁體中文(台灣))



進階物品

我們可以設計一個新物品的類別

其繼承自 `net.minecraft.item.Item` 類別

並覆寫當中的一些方法，如 `use`、`useOnBlock` 等方法

便能使新物品的功能變的更加的訂製和豐富

範例：製作一個物品「小刀」

使用後對自己造成一點傷害，並損失一點耐久度(`durability`)

進階物品

注意涉及到邏輯處理，只有伺服器端需要進行邏輯處理，客戶端不需要進行邏輯處理

`net.minecraft.world.World` 類別代表一個世界(world)

有許多世界處理的方法，如動態方法 `isClient()` 可檢測當前是否為客戶端

`net.minecraft.util.ActionResult` 枚舉類別表交互執行結果

主要用於控制手部揮動動畫

常用值有 **SUCCESS**、**FAIL**、

PASS(沒有發生事情)、

CONSUME(消耗物品)

```
package org.tyic.tyicmod.item;
```

```
import (...)
```

```
public class ModItems {
```

```
    public static final Item TYIC_LOGO = register("tyic_logo", Item::new, new Item.Settings());
```

```
    public static final Item KNIFE = register("knife", KnifeItem::new, new Item.Settings().maxCount(1).useCooldown(1).maxDamage(5));
```

```
    (...)
```

```
}
```

ModItems.java

```
package org.tyic.tyicmod.item;
```

```
import (...)
```

```
public class KnifeItem extends Item {
```

```
    public KnifeItem(Settings settings) {super(settings);}

    @Override
```

```
    public ActionResult use(World world, PlayerEntity user, Hand hand) {
```

// 邏輯處理只能在伺服器端進行，不能在客戶端

```
    if (world.isClient()) return ActionResult.PASS; // 傷害來源 造成傷害(滴)
```

```
    user.damage((ServerWorld) world, world.getDamageSources().playerAttack(user), 1f);
```

```
    user.getStackInHand(hand).damage(1, user, LivingEntity.getSlotForHand(hand));
```

```
    return ActionResult.SUCCESS; // 使物品堆疊損失耐久度
```

```
    } // 獲取手中物品堆疊
```

```
}
```

KnifeItem.java

進階物品

紋理：`assets/tyicmod/textures/item/knife.png`

像素：`16x16`

模型(左下)：

`assets/tyicmod/models/item/knife.json`

物品模型映射(右下)：

`assets/tyicmod/items/knife.json`



```
{
  "parent": "minecraft:item/generated",
  "textures": {
    "layer0": "tyicmod:item/knife"
  }
}
```

knife.json

```
{
  "model": {
    "type": "minecraft:model",
    "model": "tyicmod:item/knife"
  }
}
```

knife.json

進階物品

在地化：

English(US)：assets/tyicmod/lang/en_us.json

```
{  
  "item.tyicmod.tyic_logo": "TYIC Logo",  
  "item.tyicmod.knife": "Knife"  
}
```

en_us.json

繁體中文(台灣)：assets/tyicmod/lang/zh_tw.json

```
{  
  "item.tyicmod.tyic_logo": "TYIC 標誌",  
  "item.tyicmod.knife": "小刀"  
}
```

zh_tw.json

實際測試

展示影片：https://youtu.be/MpKjqfVo_cY

創造模式物品欄

```
package org.tyic.tyicmod.item;

import (...)

public class ModItemGroups {
    public static final RegistryKey<ItemGroup> TYIC_MOD =
        RegistryKey.of(RegistryKeys.ITEM_GROUP, Identifier.of(TyicMod.MOD_ID, "tyic_mod"));

    public static void init() {
        TyicMod.LOGGER.info("Registering mod item groups.");
        Registry.register(Registries.ITEM_GROUP, TYIC_MOD, FabricItemGroup.builder()
            .displayName(Text.translatable("itemGroup.tyic_mod"))
            .icon(() -> new ItemStack(ModItems.TYIC_LOGO))
            .build());
        ItemGroupEvents.modifyEntriesEvent(ModItemGroups.TYIC_MOD)
            .register((itemGroup) -> {
                itemGroup.add(ModItems.TYIC_LOGO);
                itemGroup.add(ModItems.KNIFE);
            });
    }
}
```

取得建造者(builder)

設定顯示名稱為 i18n 文字

ModItemGroups.java

建造(build)實例

創造模式物品欄註冊鍵

設定顯示圖標，需傳入 Supplier<ItemStack>

使用 **Fabric API** 可以輕鬆的
創建和修改創造模式物品欄

```
package org.tyic.tyicmod;

import (...)

public class TyicMod implements ModInitializer {
    (...)

    @Override
    public void onInitialize() {
        LOGGER.info("Initializing Tyic Mod.");
        ModItems.init();
        ModItemGroups.init();
    }
}
```

TyicMod.java

實際測試

在地化：

`assets/tyicmod/lang/en_us.json`

```
{  
  "item.tyicmod.tyic_logo": "TYIC Logo",  
  "item.tyicmod.knife": "Knife",  
  "itemGroup.tyic_mod": "TYIC Mod"  
}
```

en_us.json

`assets/tyicmod/lang/zh_tw.json`

```
{  
  "item.tyicmod.tyic_logo": "TYIC 標誌",  
  "item.tyicmod.knife": "小刀",  
  "itemGroup.tyic_mod": "TYIC 模組"  
}
```

zh_tw.json



成品

Github 連結：

https://github.com/TYSHIC/tyicmod/tree/01_first-item