Java 專案：方塊

TYIC 桃高資訊社

方塊

方塊(block)也是構成 Minecraft 很重要的部分

方塊的類別為 net.minecraft.block.Block

方塊物品(block item)是方塊的物品型態

大部分方塊都有對應的方塊物品，如鑽石磚、泥土等

但方塊可以沒有對應的方塊物品，如火方塊、水方塊等

方塊物品的類別為 net.minecraft.item.BlockItem

該類別繼承 net.minecraft.item.Item

並且當對方塊使用(右擊)時，會放置方塊

方塊類別

與物品類別類似，Block 類別只有一個建構子

```
Block(AbstractBlock.Settings settings)
```

AbstractBlock.Settings 是一個用來控制方塊行為的類別

如亮度(luminance，預設為 0)、註冊鍵(預設為空)等

其無建構子，需透過呼叫靜態方法 create() 創建實例

其可以通過方法鏈式呼叫進行設定

用於註冊方塊時，一定要設定註冊鍵

基本方塊

類似於物品，基本方塊就是
創建一個 Block 類別的實例並向遊戲註冊
故直接複製物品的程式碼並修改

```java
package org.tyic;

import (...)

public class TyicMod implements ModInitializer {
    (...)

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

TyicMod.java

```java
package org.tyic.block;

import (...)

public class ModBlocks {
    public static final Block TYSH_BLOCK = register("tysh_block", Block::new, AbstractBlock.Settings.create());

    public static Block register(String id, Function<AbstractBlock.Settings, Block> blockFunction, AbstractBlock.Settings settings) {
        RegistryKey<Block> registryKey = RegistryKey.of(RegistryKeys.BLOCK, Identifier.of(TyicMod.MOD_ID, id));
        return Registry.register(Registries.BLOCK, registryKey, blockFunction.apply(settings.registryKey(registryKey)));
    }

    public static void init() {
        TyicMod.LOGGER.info("Registering mod blocks.");
    }
}
```

ModBlocks.java

基本方塊

紋理：assets/tyicmod/textures/block/tysh_block.png

像素：256x256

模型(右下)：assets/tyicmod/models/block/tysh_block.json

tyic_block.png

```
{
  "parent": "minecraft:block/cube_all",
  "textures": {
    "all": "tyicmod:block/tysh_block"
  }
}
```

tyic_block.json

方塊模型

方塊模型常見的形式如下，其中 parent 的值為父模型的標識符

常見的方塊父模型有 minecraft:block/block、minecraft:block/cube、minecraft:block/cube_all

textures 物件中，鍵為紋理變數(texture variable)名稱

值為紋理的標識符或紋理變數，若為紋理變數須以井字(#)開頭

紋理變數的種類與父模型有關，具體可參考原版方塊模型

```json
{
  "parent": "namespace:path_to_parent_model",
  "textures": {
    "texture_variable_1": "namespace:path_to_texture",
    "texture_variable_2": "namespace:path_to_texture",
    (...)
  }
}
```

json

```tysh_block.json
{
  "parent": "minecraft:block/cube_all",
  "textures": {
    "all": "tyicmod:block/tysh_block"
  }
}
```

tysh_block.json

方塊狀態映射

方塊本身並沒有物品模型映射

與之相似的是方塊狀態映射(blockstates definition)

其決定了方塊要用哪種模型

所有方塊狀態映射都被放置在

`assets/namespace/blockstates`

方塊狀態映射皆為 `json` 檔案

且檔案名稱和方塊 `id` 需相同

一個最基本方塊的方塊狀態映射如右上

其中 `model` 的值為模型的標識符

即為 `"namespace:path_to_model"`

範例方塊狀態映射 `tysh_block.json` 如右下

```json
{
  "variants": {
    "": {
      "model": "namespace:path_to_model"
    }
  }
}
```

json

```tysh_block.json
{
  "variants": {
    "": {
      "model": "tyicmod:block/tysh_block"
    }
  }
}
```

tysh_block.json

基本方塊

方塊預設的翻譯鍵名為 "block.namespace.id"

在地化：

左下：assets/tyicmod/lang/en_us.json

右下：assets/tyicmod/lang/zh_tw.json

```
{
  "itemGroup.tyic_mod": "TYIC Mod",
  "item.tyicmod.tyic_logo": "TYIC Logo",
  "item.tyicmod.knife": "Knife",
  "block.tyicmod.tysh_block": "TYSH Block"
}
```
en_us.json

```
{
  "itemGroup.tyic_mod": "TYIC 模組",
  "item.tyicmod.tyic_logo": "TYIC 標誌",
  "item.tyicmod.knife": "小刀",
  "block.tyicmod.tysh_block": "TYSH 方塊"
}
```
zh_tw.json

實際測試

對於方塊，可以使用 setblock 指令直接放置

指令名稱　　座標，分別為 x、y、z，波浪號(~)表示執行者所在的座標，波浪號後方接數字表偏移量

```
/setblock ~ ~2 ~ tyicmod:tysh_block    mccmd
```

斜線開頭代表指令　　方塊 id

```
/setblock ~ ~2 ~ tyicmod:tysh_block
```

成功放置方塊後就可以盡情欣賞新方塊

然而當嘗試使用 give 指令獲取方塊物品時

會發現遊戲找不到該物品

這是因為剛剛的程式碼只註冊了方塊

而並沒有註冊方塊物品

```
未知的物品 'tyicmod:tysh_block'
give @s tyicmod:tysh_block<--[這裡]
第 9 個字元 未知的物品 'tyicmod:tysh_block': /give @s <--[HERE]
/give @s tyicmod:tysh_block_
```

方塊物品類別

與物品類別類似，`BlockItem` 類別只有一個建構子

```
BlockItem(Block block, Item.Settings settings)
```

而因為 `BlockItem` 本質上其實也是一種物品

所以跟其他物品的註冊方式一樣

故直接呼叫之前寫好的 `ModItems.register` 方法即可

若之後需要取得註冊的方塊物品

只須呼叫動態方法 `Block.asItem()` 即可

故不須將註冊的方塊物品額外儲存起來

設定方塊物品

在註冊方塊的函式中加入註冊方塊物品的部分

注意 `Item.Setting` 實例須呼叫方法 `useBlockPrefixedTranslationKey()`

使該物品使用與方塊相同的預設翻譯鍵名 `"block.namespace.id"`

否則其會使用物品的預設翻譯鍵名 `"item.namespace.id"`

```java
package org.tyic.tyicmod.block;

import (...)

public class ModBlocks {
    public static final Block TYSH_BLOCK = register("tysh_block", Block::new, AbstractBlock.Settings.create());

    public static Block register(String id, Function<AbstractBlock.Settings, Block> blockFunction, AbstractBlock.Settings settings) {
        RegistryKey<Block> registryKey = RegistryKey.of(RegistryKeys.BLOCK, Identifier.of(TyicMod.MOD_ID, id));
        Block block = Registry.register(Registries.BLOCK, registryKey, blockFunction.apply(settings.registryKey(registryKey)));
        ModItems.register(id, (itemSettings) -> new BlockItem(block, itemSettings), new Item.Settings().useBlockPrefixedTranslationKey());
        return block;
    }

    public static void init() {
        TyicMod.LOGGER.info("Registering mod blocks.");
    }
}
```

ModBlocks.java

設定方塊物品

方塊物品無須再設定紋理、模型和在地化
但仍須設定物品模型映射
使方塊物品直接使用方塊的模型

物品模型映射：assets/tyicmod/items/tysh_block.json

```
{
  "model": {
    "type": "minecraft:model",
    "model": "tyicmod:block/tysh_block"
  }
}
```
tysh_block.json

設定方塊物品

也可以將方塊物品加入創造模式物品欄

```java
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);
                    itemGroup.add(ModBlocks.TYSH_BLOCK);
                });
    }
}
```

ModItemGroups.java

實際測試

對方塊點擊滑鼠中鍵，可直接取得對應的方塊物品

方塊狀態

方塊狀態(blockstate)是用來描述方塊的一些屬性(property)
如半磚的是上半磚還是下半磚、門是開還是關
階梯是含水還是不含水、方塊朝向哪個方向等
所有的方塊狀態可參見維基百科(https://zh.minecraft.wiki/w/%E6%96%B9%E5%9D%97%E7%8A%B6%E6%80%81)
方塊狀態的類別為 `net.minecraft.block.BlockState`
屬性的類別為 `net.minecraft.state.property.Property<T>`
所有屬性位於 `net.minecraft.state.property.Properties`
最常見的屬性就是方塊朝向方向(facing direction)

進階方塊

與物品一樣，我們可以設計一個新方塊的類別
其繼承自 `net.minecraft.block.Block` 類別
並覆寫當中的一些方法，如 `onUse`、`onUseWithItem` 等方法
便能使新物品的功能變的更加的訂製和豐富

進階方塊

範例：製作一個方塊「注水器」
有一面為出水口，其他面為銅方塊紋理
出水口要朝向放置的玩家
若注水器相鄰水方塊，且手持特定物品對其出水口面右鍵使用時
將該物品轉換為特定新物品，一次轉換一個：
泥土 -> 泥巴
蒼白苔蘚方塊 -> 苔蘚方塊
蒼白覆地苔蘚 -> 覆地苔蘚

進階方塊

若方塊有屬性，則需要覆寫 appendProperties 方法

並在該方法內呼叫 builder.add(Property<?>... properties)

且須在建構子設定預設方塊狀態，設定各新屬性的預設值

BlockState 的動態方法 with 用於更改特定屬性的值

```java
package org.tyic.tyicmod.block;

import (...)

public class WaterFeederBlock extends Block {
    public static final EnumProperty<Direction> FACING = Properties.FACING;

    public WaterFeederBlock(Settings settings) {
        super(settings);
        this.setDefaultState(this.stateManager.getDefaultState().with(FACING, Direction.NORTH));
    }

    @Override
    protected void appendProperties(StateManager.Builder<Block, BlockState> builder) {
        builder.add(FACING);
    }
```

通常會將主要面面朝方向設為北方
這樣方塊物品渲染才能看見該面

WaterFeederBlock.java (2/3)

進階方塊

`BlockState getPlacementState(ItemPlacementContext ctx)`

方法會在方塊放置時被呼叫，用於取得方塊放置時的方塊狀態

其中 `ItemPlacementContext` 類別帶有許多方塊被放置時的資訊

如方塊放置的座標(position、coordinate)、玩家朝向的方向等

對於有方向的方塊，建議覆寫 `rotate` 和 `mirror` 方法

且此二方法內容通常固定為下方所示，用於結構方塊等功能

```java
@Override
public BlockState getPlacementState(ItemPlacementContext ctx) {
    return this.getDefaultState().with(FACING, ctx.getPlayerLookDirection().getOpposite());
}

@Override
protected BlockState rotate(BlockState state, BlockRotation rotation) {
    return state.with(FACING, rotation.rotate(state.get(FACING)));
}

@Override
protected BlockState mirror(BlockState state, BlockMirror mirror) {
    return state.rotate(mirror.getRotation(state.get(FACING)));
}
```

玩家放置注水器時，注水器應朝玩家的反向，出水口才會面向玩家

WaterFeederBlock.java (2/3)

進階方塊

BlockHitResult 類別帶有許多方塊與玩家交互時的資訊

```
    private boolean nextToWater(World world, BlockPos pos) {
        for (Direction direction : Direction.values())
            if (world.getBlockState(pos.offset(direction)).isOf(Blocks.WATER))
                return true;
        return false;
    }

    private final Map<Item, Item> itemMap = Map.<Item, Item>of(
            Items.DIRT, Items.MUD,
            Items.PALE_MOSS_BLOCK, Items.MOSS_BLOCK,
            Items.PALE_MOSS_CARPET, Items.MOSS_CARPET
    );

    @Override
    protected ActionResult onUseWithItem(ItemStack stack, BlockState state, World world, BlockPos pos,
                                         PlayerEntity player, Hand hand, BlockHitResult hit) {
        if (world.isClient() || hit.getSide() != state.get(FACING) || !nextToWater(world, pos)) return ActionResult.PASS;
        Item item = stack.getItem();
        if (!itemMap.containsKey(item)) return ActionResult.PASS;
        ItemStack itemStack = ItemUsage.exchangeStack(stack, player, new ItemStack(itemMap.get(item)));
        player.setStackInHand(hand, itemStack);
        return ActionResult.SUCCESS.withNewHandStack(itemStack);
    }
}
```

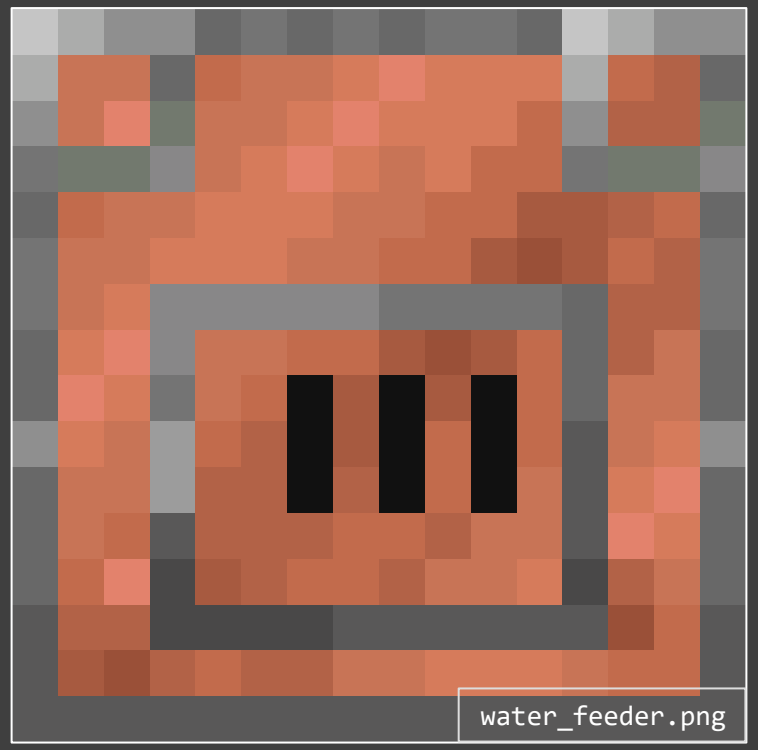
消耗某一物品堆疊並獲得新物品堆疊

WaterFeederBlock.java (3/3)

進階方塊

紋理：assets/tyicmod/textures/blcok/water_feeder.png

像素：16x16



進階方塊

模型(左下)：

assets/tyicmod/models/block/water_feeder.json

物品模型映射(右下)：

assets/tyicmod/items/water_feeder.json

```
{
  "parent": "minecraft:block/cube",
  "textures": {
    "particle": "minecraft:block/copper_block",
    "down": "minecraft:block/copper_block",
    "up": "minecraft:block/copper_block",
    "north": "tyicmod:block/water_feeder",
    "south": "minecraft:block/copper_block",
    "west": "minecraft:block/copper_block",
    "east": "minecraft:block/copper_block"
  }
}
```

water_feeder.json

```
{
  "model": {
    "type": "minecraft:model",
    "model": "tyicmod:block/water_feeder"
  }
}
```

water_feeder.json

進階方塊

對於需要因不同方塊狀態產生不同變化的模型
需在方塊狀態映射中明定各種方塊狀態下
所使用的模型以及模型的旋轉

`variants` 物件的鍵為屬性
值為候選模型
候選模型的 `model` 值
為模型的標識符
`x` 或 `y` 值分別為
對 `x` 軸或 `y` 軸旋轉角度
須為 `90` 的倍數

```json
{
  "variants": {
    "property1=value1,property2=value1,...": {
      "model": "namespace:path_to_model",
      "x": x_axis_rotate_degree
    },
    "property1=value2,property2=value1,...": {
      "model": "namespace:path_to_model",
      "y": y_axis_rotate_degree
    },
    "property1=value1,property2=value2,...": {
      "model": "namespace:path_to_model"
    },
    "property1=value2,property2=value2,...": {
      "model": "namespace:path_to_model",
      "x": x_axis_rotate_degree,
      "y": y_axis_rotate_degree
    }
}
```

進階方塊

方塊狀態映射：

assets/tyicmod/blockstates/block/water_feeder.json

```
{
  "variants": {
    "facing=down": {
      "model": "tyicmod:block/water_feeder",
      "x": 90
    },
    "facing=east": {
      "model": "tyicmod:block/water_feeder",
      "y": 90
    },
    "facing=north": {
      "model": "tyicmod:block/water_feeder"
    },
    "facing=south": {
      "model": "tyicmod:block/water_feeder",
      "y": 180
    },
    "facing=up": {
      "model": "tyicmod:block/water_feeder",
      "x": 270
    },
    "facing=west": {
      "model": "tyicmod:block/water_feeder",
      "y": 270
    }
  }
}
```

water_feeder.json

進階方塊

在地化：

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

```
{
  (...),
  "block.tyicmod.water_feeder": "Water Feeder"
}
```
en_us.json

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

```
{
  (...),
  "block.tyicmod.water_feeder": "注水器"
}
```
zh_tw.json

實際測試

展示影片：https://youtu.be/TZ3dOSH5-y0

成品

Github 連結：

https://github.com/TYSHIC/tyicmod/tree/02_first-block