前言

NeigeItems是一个基于TabooLib，使用Kotlin编写的随机物品插件

功能还行，尚且能用

交流

QQ群: 648142579

开始

安装

需求

默认配置

config.yml

需求

1.12.2-1.19.3 bukkit服务端

已测试：

- paper1.12.2-1.19.3
- arclight1.16.5
- spigot1.12.2
- catserver1.12.2

安装NeigeItems

NeigeItems-Release

1. 点击上方链接，通过GitHub下载NeigeItems插件
2. 将文件丢入plugins文件夹
3. 重启服务器

注意

> **警告**
>
> 不要尝试通过Plugman等热重载工具加载NeigeItems

config.yml

```yaml
Main:
  # MM物品默认保存路径
  MMItemsPath: MMItems.yml
  # 是否开启debug模式
  Debug: false
# 将消息设置为""则不进行提示
Messages:
  # 一些消息的提示类型
  type:
    # Pack/Items
    # 给予物品包是发送物品包提示还是发送所有物品提示
    givePackMessage: Pack
  # 玩家不在线提示
  invalidPlayer: §e[NI] §6玩家不在线或不存在
  # 给予成功提示
  successInfo: §e[NI] §6成功给予 §f{player} §a{amount} §6个 §f{name}
  # 被给予成功提示
  givenInfo: §e[NI] §6你得到了 §a{amount} §6个 §f{name}
  # 给予物品包成功提示
  successPackInfo: §e[NI] §6成功给予 §f{player} §a{amount} §6个 §f{name} §6物品包
  # 被给予成功物品包提示
  givenPackInfo: §e[NI] §6你得到了 §a{amount} §6个 §f{name} §6物品包
  # 给予成功提示
  dropSuccessInfo: §e[NI] §6成功在 §a{world} §6的 §a{x},{y},{z} §6掉落了 §a{amount} §6个 §f{name}
  # 未知物品提示
  unknownItem: §e[NI] §6找不到ID为 §a{itemID} §6的物品
  # 未知物品包提示
  unknownItemPack: §e[NI] §6找不到ID为 §a{packID} §6的物品包
```

GlobalSections/ExampleSection.yml

```yaml
global-strings-1:
  # 随机字符节点
  type: strings
  values:
  - test1
  - test2
global-number-1:
  # 随机数节点
  type: number
  # 随机数最小值
  min: 1
  # 随机数最大值
  max: 2
  # 小数保留位数
  fixed: 3
global-calculation-1:
  # 公式节点
  type: calculation
  # 计算公式
  formula: 1+2+3<global-number-1>
  # 公式结果最小值
  min: 1
  # 公式结果最大值
  max: 100
  # 小数保留位数
  fixed: 3
global-weight-1:
  # 权重字符串节点
  type: weight
  values:
  # 权重::字符串内容
  - 5::第一行
```

Items/ExampleItem.yml

```yaml
ExampleItem:
  # 物品材质
  material: LEATHER_HELMET
  # 物品CustomModelData(适用于1.14+)
  custommodeldata: 1
  # 物品损伤值
  damage: 1
  # 物品名
  name: §6一件皮革甲
  # 物品Lore
  lore:
  - 'PAPI变量测试: %player_level%'
  - '16进制颜色测试: <#ABCDEF>好耶'
  - '私有简单节点测试: <simple-1>'
  - '私有字符串节点测试: <strings-1>'
  - '私有随机数节点测试: <number-1>'
  - '私有公式节点测试: <calculation-1>'
  - '私有权重节点测试: <weight-1>'
  - '私有JavaScript节点测试: <js-1>'
  - '即时声明字符串节点测试: <strings::number-1_weight-1>'
  - '即时声明随机数节点测试: <number::0_10_0>'
  - '即时声明公式节点测试: <calculation::1+1+3+<number-1>_2>'
  - '即时声明权重节点测试: <weight::5::权重文本1_1::权重文本2>'
  - '即时声明papi节点测试: <papi::<papiString-1><papiString-2>>'
  - '即时声明JavaScript节点测试: <js::ExampleScript.js::main>'
  - '全局节点调用测试: <global-strings-1>'
  - '嵌套识别测试: <<strings-1>>'
  - '文本中小于号请添加反斜杠, 防止错误识别'
  - '形如: \<\<\<\>\>\>'
  - '请尽量避免使用即时声明节点'
  - "换行符测试\n换行符测试"
  # 物品附魔
```

Expansions/ExampleExpansion.yml

```
/**
 * 服务器开启后 及 ni reload后异步执行
 */
function enable() {
  // 调用指令注册
  commandExample()
  // 调用监听器注册
  listenerExample()
  // 调用PAPI变量注册
  placeholderExample()
}

/**
 * 服务器关闭前同步执行 及 ni reload前异步执行
 */
function disable() {
}

/**
 * 服务器开启后异步执行
 */
function serverEnable() {
}

/**
 * 服务器关闭前同步执行
 */
function serverDisable() {
}

/**
 * 指令注册示例
```

Scripts/ExampleScript.js

```
function main() {
  if (typeof this.player != "undefined") {
    return this.vars("<strings-1>") + this.player.getName()
  } else {
    return this.vars("<strings-1>")
  }
}
```

ItemActions/ExampleAction.yml

```
ExampleItem2:
 # 左键触发
 left:
  # 动作内容
  actions:
   # 这条动作没有condition, 所以必定执行
   - "tell: 你正尝试触发&e ExampleItem2 &f物品"
   # 当前这条动作的执行条件
   - condition: perm("item.ExampleItem2")
    # 满足条件后执行的动作
    actions:
     # 后台执行
     - "console: say &e%player_name% &f拥有&e item.ExampleItem2 &f权限"
     # 玩家执行
     - "command: say 我拥有&e item.ExampleItem2 &f权限"
    # 不满足条件时执行的动作
    deny:
     - "tell: 你没有&e item.ExampleItem2 &f权限"
```

CustomSection/CustomSection.js

```
// 文件名不重要, 写成啥都行
// main函数会自动执行
function main() {
  // 导入相应的类, 这两行看不懂的话直接抄就行
  const SectionManager =
Packages.pers.neige.neigeitems.manager.SectionManager.INSTANCE
  const CustomSection =
Packages.pers.neige.neigeitems.section.impl.CustomSection
  const SectionUtils = Packages.pers.neige.neigeitems.utils.SectionUtils

  // 创建自定义节点
  const customSection = new CustomSection(
    // 节点id
    "test",
    /**
     * 用于私有节点解析
     * @param data ConfigurationSection 节点内容
     * @param cache HashMap<String, String>? 解析值缓存
     * @param player OfflinePlayer? 待解析玩家
     * @param sections ConfigurationSection? 节点池
     * @return 解析值
     */
    function(data, cache, player, sections) {
      if (data.contains("values")) {
        // SectionUtils.parseSection("待解析字符串", cache, player, sections)用于解析
节点内容
        return SectionUtils.parseSection("<number::0_1_2>", cache, player, sections)
      }
      return null
    },
    /**
     * 用于即时节点解析
```

CustomActions/CustomAction.js

```
// 文件名不重要, 写成啥都行
// main函数会自动执行
function main() {
  // 导入相应的类, 这两行看不懂的话直接抄就行
  const ActionManager =
Packages.pers.neige.neigeitems.manager.ActionManager.INSTANCE
  const SectionUtils = Packages.pers.neige.neigeitems.utils.SectionUtils

  // 插入新的自定义动作
  ActionManager.addAction(
    // 动作名称
    "test",
    // 动作内容(一般是异步调用的, 所以需要同步执行的内容需要自行同步)
    function(player, string) {
      // 调用动作
      ActionManager.runAction(player, "tell: 123")
      ActionManager.runAction(player, "tell: 456")
      player.sendMessage(SectionUtils.parseSection("<number::0_10_2>"))
      // 每个动作都一定要返回一个布尔量(true或false), 返回false相当于终止一连串动作
的执行
      return true
    })
}
```

CustomItemEditors/CustomItemEditor.js

```
// 文件名不重要, 写成啥都行
// main函数会自动执行
```

ItemPacks/ExampleItemPack.yml

```yaml
Example1:
  Items:
  # 支持解析即时声明节点
  # [物品ID] (数量(或随机最小数量-随机最大数量)) (生成概率) (是否反复随机) (指向数据)
  - ExampleItem 1-5 0.5
  - test
  FancyDrop:
    # 偏移量
    offset:
      # 横向偏移量(或随机最小偏移量-随机最大偏移量)
      x: 0.1
      # 纵向偏移量(或随机最小偏移量-随机最大偏移量)
      y: 0.8
    angle:
      # 抛射类型(round/random)
      type: round
Example2:
  Items:
  - <test>
  FancyDrop:
    offset:
      x: 0.1
      y: 0.8
    angle:
      type: round
  # 引用的全局节点
  globalsections:
  # 这种直接填写文件名的方式可以直接调用文件内的全部全局节点
  # - ExampleSection.yml
  - global-strings-1
  - global-number-1
  # 物品私有节点
```

Scripts/ItemTime.js

```
function main(time) {
  const date = new Date()
  date.setTime(date.getTime() + (Number(time) * 1000))
  return date.getFullYear() + "年" + (date.getMonth() + 1) + "月" + date.getDate() + "日"
+ date.getHours() + "时" + date.getMinutes() + "分" + date.getSeconds() + "秒"
}
```

指令

物品动作

全部命令需要OP权限/后台执行, []为必填, ()为选填

物品列表

全部命令需要OP权限/后台执行, []为必填, ()为选填

物品获取

全部命令需要OP权限/后台执行, []为必填, ()为选填

物品掉落

全部命令需要OP权限/后台执行, []为必填, ()为选填

物品保存

全部命令需要OP权限/后台执行, []为必填, ()为选填

物品编辑

全部命令需要OP权限/后台执行, []为必填, ()为选填

杂项

全部命令需要OP权限/后台执行, []为必填, ()为选填

测试

全部命令需要OP权限/后台执行, []为必填, ()为选填

物品动作

> 全部命令需要OP权限/后台执行, []为必填, ()为选填

action

> /ni action [玩家ID][动作内容] > 执行NI物品动作

- `[玩家ID]` 在线的玩家ID
- `[动作内容]` 要执行的动作内容(支持即时声明节点)

  如：`tell: hello`

  如：`giveMoney: <number::1_1000>`

物品列表

> 全部命令需要OP权限/后台执行, []为必填, ()为选填

list

> /ni list (页码) > 查看所有NI物品

- `(页码)` 打开对应页的物品列表(默认为1)

物品获取

> 全部命令需要OP权限/后台执行, []为必填, ()为选填

get

> /ni get [物品ID] (数量) (是否反复随机) (指向数据) > 根据ID获取NI物品

- `[物品ID]` NI物品ID
- `(数量)` 获取的数量 (默认为1)
- `(是否反复随机)` 默认为true
- `(指向数据)` 字符串化JSON文本

  形如`{"string-1":"文本文本文本"}`

  这样物品生成时节点`string-1`的值将变为`文本文本文本`

  形如`{"test1":"test1","test2":"test2"}`

  这样物品生成时节点`test1`的值将变为`test1`, 节点`test2`的值将变为`test2`

give

> /ni give [玩家ID][物品ID] (数量) (是否反复随机) (指向数据) > 根据ID给予NI物品

- `[玩家ID]` 待给予玩家的ID

- `[物品ID]` NI物品ID
- `(数量)` 获取的数量 (默认为1)
- `(是否反复随机)` 默认为true
- `(指向数据)` 字符串化JSON文本

  形如`{"string-1":"文本文本文本"}`

  这样物品生成时节点`string-1`的值将变为`文本文本文本`

  形如`{"test1":"test1","test2":"test2"}`

  这样物品生成时节点`test1`的值将变为`test1`, 节点`test2`的值将变为`test2`

giveAll

> /ni giveAll [物品ID] (数量) (是否反复随机) (指向数据) > 根据ID给予所有人NI物品

- `[物品ID]` NI物品ID
- `(数量)` 获取的数量 (默认为1)
- `(是否反复随机)` 默认为true
- `(指向数据)` 字符串化JSON文本

  形如`{"string-1":"文本文本文本"}`

  这样物品生成时节点`string-1`的值将变为`文本文本文本`

  形如`{"test1":"test1","test2":"test2"}`

这样物品生成时节点`test1`的值将变为`test1`，节点`test2`的值将变为`test2`

givePack

> /ni givePack [玩家ID][物品包ID] (数量) (指向数据) > 根据ID给予NI物品包

- `[玩家ID]` 待给予玩家的ID
- `[物品包ID]` NI物品包ID
- `(数量)` 获取的数量 (默认为1)
- `(指向数据)` 字符串化JSON文本

mm get

> /ni mm get [物品ID] (数量) > 根据ID获取MM物品

- `[物品ID]` MM物品ID
- `(数量)` 获取的数量 (默认为1)

相较于MM自带的物品给予，优势在于满背包将自动掉落至地上，且消息文本可自定义。

mm give

> /ni mm give [玩家ID][物品ID] (数量) > 根据ID给予MM物品

- `[玩家ID]` 待给予玩家的ID
- `[物品ID]` MM物品ID
- `(数量)` 获取的数量 (默认为1)

相较于MM自带的物品给予，优势在于满背包将自动掉落至地上，且消息文本可自定义。

mm giveAll

> /ni mm giveAll [物品ID] (数量) > 根据ID给予所有人MM物品

- `[物品ID]` MM物品ID
- `(数量)` 获取的数量 (默认为1)

相较于MM自带的物品给予，优势在于满背包将自动掉落至地上，且消息文本可自定义。

物品掉落

> 全部命令需要OP权限/后台执行, []为必填, ()为选填

drop

> /ni drop [物品ID][数量] [世界名][X坐标] [Y坐标][Z坐标] [是否反复随机][物品解析对象] (指向数据) > 于指定位置掉落NI物品

- `[物品ID]` NI物品ID
- `[数量]` 获取的数量，默认为1
- `[世界名]` 物品掉落世界的名称
- `[X坐标]` 物品掉落世界的X轴坐标
- `[Y坐标]` 物品掉落世界的Y轴坐标
- `[Z坐标]` 物品掉落世界的Z轴坐标
- `[是否反复随机]` 默认为true
- `(物品解析对象)` 用于物品解析的玩家ID

  用于解析物品内的PAPI变量及随机节点
- `(指向数据)` 字符串化JSON文本

  形如`{"string-1":"文本文本文本"}`

这样物品生成时节点`string-1`的值将变为`文本文本文本`

形如`{"test1":"test1","test2":"test2"}`

这样物品生成时节点`test1`的值将变为`test1`，节点`test2`的值将变为`test2`

> 提示
>
> 如果你想让MM怪物被玩家击杀后掉落NI物品，你可以直接查看：插件适配-物品掉落

dropPack

> /ni dropPack [物品包ID] (数量) [世界名][X坐标] [Y坐标][Z坐标] (物品解析对象) (指向数据) > 于指定位置掉落NI物品包

- `[物品ID]` NI物品包ID
- `[数量]` 获取的数量，默认为1
- `[世界名]` 物品掉落世界的名称
- `[X坐标]` 物品掉落世界的X轴坐标
- `[Y坐标]` 物品掉落世界的Y轴坐标
- `[Z坐标]` 物品掉落世界的Z轴坐标
- `[物品解析对象]` 用于物品解析的玩家ID
- `(指向数据)` 字符串化JSON文本

  用于解析物品内的PAPI变量及随机节点

物品保存

> 全部命令需要OP权限/后台执行, []为必填, ()为选填

save

> /ni save [物品ID] (保存路径) > 将手中物品以对应ID保存至对应路径

- `[物品ID]` 保存后的NI物品ID
- `(保存路径)` 物品存储的文件路径, 默认为 `物品ID.yml`

  形如 `test.yml`，将存储于 `plugins/NeigeItems/Items/test.yml`

> **提示**
> 如果物品ID重复(已存在对应ID的NI物品)，将保存失败并收到提示。

cover

> /ni cover [物品ID] (保存路径) > 将手中物品以对应ID覆盖至对应路径

- `[物品ID]` 保存后的NI物品ID
- `(保存路径)` 物品存储的文件路径, 默认为 `物品ID.yml`

  形如 `test.yml`，将存储于 `plugins/NeigeItems/Items/test.yml`

> 提示
> 如果物品ID重复(已存在对应ID的NI物品)，将直接覆盖原物品，强行保存。

mm load

> /ni mm load [物品ID] (保存路径) > 将对应ID的MM物品保存为NI物品

- `[物品ID]` 待转换的MM物品ID
- `(保存路径)` 物品存储的文件路径，默认为配置文件中的Main.MMItemsPath

  形如`test.yml`，将存储于`plugins/NeigeItems/Items/test.yml`

> 提示
> 如果物品ID重复(已存在对应ID的NI物品)，将保存失败并收到提示。

mm cover

> /ni mm cover [物品ID] (保存路径) > 将对应ID的MM物品覆盖为NI物品

- `[物品ID]` 待转换的MM物品ID
- `(保存路径)` 物品存储的文件路径，默认为配置文件中的Main.MMItemsPath

  形如`test.yml`，将存储于`plugins/NeigeItems/Items/test.yml`

> 提示
> 如果物品ID重复(已存在对应ID的NI物品)，将直接覆盖原物品，强行保存。

mm loadAll

> /ni mm loadAll (保存路径) > 将全部MM物品转化为NI物品

- `(保存路径)` 物品存储的文件路径，默认为配置文件中的Main.MMItemsPath 形如 `test.yml`，将存储于 `plugins/NeigeItems/Items/test.yml`

> **提示**
>
> 如果物品ID重复(已存在对应ID的NI物品)，将保存失败并收到提示。

物品编辑

> 全部命令需要OP权限/后台执行, []为必填, ()为选填

edithand

> /ni edithand [玩家ID][物品编辑函数ID] [函数内容] > 通过对应编辑函数编辑主手物品

- `[玩家ID]` 待操作玩家的ID
- `[物品编辑函数ID]` 待调用物品编辑函数的ID
- `[函数内容]` 物品编辑函数的内容

editoffhand

> /ni editoffhand [玩家ID][物品编辑函数ID] [函数内容] > 通过对应编辑函数编辑副手物品

- `[玩家ID]` 待操作玩家的ID
- `[物品编辑函数ID]` 待调用物品编辑函数的ID
- `[函数内容]` 物品编辑函数的内容

editslot

> /ni editslot [玩家ID][对应槽位] [物品编辑函数ID][函数内容] > 通过对应编辑函数编辑对应槽位物品

- [玩家ID] 待操作玩家的ID
- [对应槽位] 对应物品槽位, 如图

- [物品编辑函数ID] 待调用物品编辑函数的ID
- [函数内容] 物品编辑函数的内容

杂项

> 全部命令需要OP权限/后台执行, []为必填, ()为选填

help

> /ni help > 查看帮助信息

reload

> /ni reload > 重新加载NI物品

测试

> 全部命令需要OP权限/后台执行, []为必填, ()为选填

itemNBT

> /ni itemNBT > 查看当前手中物品的NBT

物品

物品配置

4 个项目

物品动作

6 个项目

物品编辑函数

3 个项目

物品变量

简介

物品包

路径

自定义耐久

简介

物品配置

简介

存放路径

配置项

ID

模板继承

以默认指令配置为例

额外选项

以默认配置为例:

简介

存放路径

所有物品配置文件应存放于`plugins/NeigeItems/Items`文件夹

重复 ID 的物品仍然会被加载，但可能互相覆盖

最后哪个物品活下来。。。随缘了属于是

默认配置

默认配置包含多种实例

编写你的物品

/ni save是万物起源

遇事不决，/ni save。如果不行，就/ni cover。这是最简单最便捷的快速生成物品配置的方法

物品保存指令

物品覆盖指令

某人不看配置不进游戏，草草看过两遍wiki，声称wiki看不懂，被众群友嘲笑良久。

配置项

ID

所有物品都应该有一个ID，如下格式：

```yaml
物品ID:
  # 具体的配置项, 以物品材质为例
  material: STONE
```

材质

即，物品是石头还是木头还是钻石剑

```yaml
物品1:
  # 这个物品是石头
  material: STONE
物品2:
  # 这个物品是钻石
  material: DIAMOND
```

ID都有哪些，见下方链接

https://hub.spigotmc.org/javadocs/spigot/org/bukkit/Material.html

如果你看着 ID 不知道它对应什么物品。。。

一般来讲，你可以在游戏中同时按下 F3+H，启用高级显示框，这样物品下方就会出现对应的ID。

如上图所示，`minecraft:stone` 对应 `STONE`

对于 mod 物品，前缀不能省略。

比如一个名称为`mod:test`的物品，对应的 ID 应为 `MOD_TEST`

但是啊但是，你有没有看上面啊?

/ni save是万物起源。别搁这儿看ID了，保存一下什么都有了，看个锤子看。

物品名

具体配置如下

```
有名字的铁剑:
  material: IRON_SWORD
  name: 我有名字
```

物品Lore

具体配置如下

```
有Lore的铁剑:
  material: IRON_SWORD
```

你可以通过换行符`\n`换行, 在一行中书写多行lore

值得一提的是, 在yaml语法中, 双引号包裹的`"\n"`才代表换行符

单引号包裹的`'\n'`只代表一段形似`\n`的字符

例:

```yaml
有Lore的铁剑:
  material: IRON_SWORD
  lore:
  - "我有lore\n我真有lore\n信我"
```

子ID/损伤值

在 1.12.2 及以下的版本中，某些物品存在“子ID”。

比如 WOOL 是白色羊毛，而子ID为 1 的 WOOL 是橙色羊毛。

对应配置方法如下

```yaml
白色羊毛:
  material: WOOL
橙色羊毛:
  material: WOOL
  # 子ID为1
  damage: 1
```

而对于有耐久的物品，damage对应损伤值，即，物品消耗了几点耐久。

```yaml
铁剑:
  material: IRON_SWORD
用了一下的铁剑:
  material: IRON_SWORD
  # 消耗了1点耐久
  damage: 1
```

CustomModelData

对于 1.14+ 的服务器，物品有了一个新的属性，CustomModelData。

一般人们用它搭配材质包制作自定义材质物品。

对应配置方法如下

```yaml
铁剑:
  material: IRON_SWORD
  # CustomModelData 为 1
  custommodeldata: 1
```

附魔

附魔名称列表，应前往以下链接查看

https://hub.spigotmc.org/javadocs/spigot/org/bukkit/enchantments/Enchantment.html

具体配置方法如下

```yaml
有附魔的铁剑:
  material: IRON_SWORD
  enchantments:
    # 锋利5
    DAMAGE_ALL: 5
```

啥？你说全是英文你根本看不懂哪个对哪个？

/ni save干什么用的

无法破坏

具体配置如下

```yaml
无法破坏的铁剑:
  material: IRON_SWORD
  unbreakable: true
```

隐藏属性

有的物品明明无法破坏，物品信息里却看不到。

有的物品明明有附魔，物品信息里却看不到。

具体配置方法如下

```yaml
啥都看不到的铁剑:
  material: IRON_SWORD
  hideflags:
  # 隐藏物品属性
```

物品颜色

药水和皮革护甲可以拥有自定义颜色，具体配置方法如下

```
有颜色的皮革头盔1:
  material: LEATHER_HELMET
  color: 'ABCDEF'
有颜色的皮革头盔2:
  material: LEATHER_HELMET
  color: 666666
```

如上所示，你可以用十进制和十六进制两种方式配置物品颜色。

如果你想要以十进制表示颜色，那么color必须配置一个数字（不被引号包裹）

如果你想要以十六进制表示颜色，那么color必须是一个字符串（被引号包裹）

比如，`color: '666666'` 表示的是十六进制，等价于 `color: 6710886`

自定义NBT

许多插件会向物品中插入一些自定义NBT，用来记录某些信息。

NeigeItems也允许你这样做。

你可以通过插入自定义NBT，兼容一些基于NBT的插件，比如

```
超猛镐子:
  material: IRON_PICKAXE
  nbt:
```

如果你装了MMOItems，那这个镐子现在应该有100万攻击力了。

你可能注意到，1000000前面有一个`(Double)`。

这个前缀代表，生成这条NBT的时候，会以 Double 类型生成（写的时候不要忘记括号后面的空格）。

如果你不写的话，生成时这条NBT很有可能就变成了Int类型或者Long类型。

这种用于转换类型的前缀应该应用于数值类型的NBT

具体有以下类型可以选择

```
# Byte 类型的 1
(Byte) 1
# Short 类型的 1
(Short) 1
# Int 类型的 1
(Int) 1
# Long 类型的 1
(Long) 1
# Float 类型的 1
(Float) 1
# Double 类型的 1
(Double) 1
```

使用类型转换前缀，一定要加空格

但是啊但是，别搁这儿看了，你直接/ni save一下，自动就都出来了。

额外选项

使用次数，物品光效，掉落技能什么的，都属于额外选项。

具体配置如下

```yaml
嗯叠BUFF的铁剑:
  material: IRON_SWORD
  options:
    charge: 10
    color: GOLD
```

options下面的就是额外选项。

具体内容请查看额外选项

模板继承

你可以让一个配置继承其他配置的部分或全部内容

具体内容请查看模板继承

随机节点

私有节点应直接配置与物品下方，比如

```yaml
随机名称的铁剑:
  material: IRON_SWORD
  name: <weight-1>
  sections:
    weight-1:
      type: weight
      values:
      - 5::名字1
      - 4::名字2
```

有关私有节点的各个类型，具体请查看私有/全局节点

全局节点引用

你可以在物品配置中引用全局节点。

插件会在初始化的时候检查各个物品是否引用全局节点，如果引用了，就将所有引用到的节点加载到物品配置中，当做私有节点解析和调用。（当然，这个过程不会反应到物品配置上）

具体调用方式如下

```
铁剑:
  material: IRON_SWORD
  globalsections:
  # 引用 ExampleSection.yml 文件中的全部全局节点
  - ExampleSection.yml
  # 引用名为 global-strings-1 的全局节点
  - global-strings-1
```

模板继承

以默认指令配置为例

```
# 一个测试模板
template1:
  material: IRON_SWORD
  lore:
  - "&e攻击伤害: &f<damage>"
  nbt:
    MMOITEMS_ATTACK_DAMAGE: (Double) <damage>
# 一个测试模板
template2:
  material: DIAMOND_SWORD

# 一个全局继承测试, 它继承了"template1"的所有内容
templateItem1:
  inherit: template1
  name: §f物品继承测试
  sections:
    damage: 100
# 一个部分继承测试, 它继承了"template1"的lore, 以及"template2"的material
templateItem2:
  inherit:
    lore: template1
    material: template2
  name: §f物品继承测试
  sections:
    damage: 100
# 一个顺序继承测试, 它将按顺序进行节点继承. 先继承"template1"的所有内容，再继承"template2"的所有内容
templateItem3:
```

可以看到，我们可以通过在物品配置中添加"inherit"来继承其他物品的配置。

```yaml
inherit: template1
```

代表这个物品将继承"template1"的全部内容

```yaml
inherit:
  lore: template1
  material: template2
```

代表这个物品将继承"template1"的"lore"配置项，以及"template2"的"material"配置项

```yaml
inherit:
- template1
- template2
```

代表这个物品将先继承"template1"的所有配置项，再继承"template2"的所有配置项。

因此对于重复的项，后者会对前者进行覆盖。

额外选项

以默认配置为例:

```yaml
ExampleItem4:
  material: STONE
  lore:
  - '物品使用次数: %neigeitems_charge%/%neigeitems_maxCharge%'
  options:
    charge: 10
```

`options` 下的所有配置项, 即为"额外选项"

使用次数

```yaml
test:
  material: STONE
  options:
    charge: 10
```

`charge` 该物品可使用的次数 (与可触发物品动作的次数有关, 不是原版的物品耐久)

关于该选项在Name和Lore中的显示, 请查看物品变量

最大使用次数

```yaml
test:
  material: STONE
  options:
    charge: 5
    maxcharge: 10
```

`maxcharge` 该物品最大可使用的次数 (与可触发物品动作的次数有关, 不是原版的物品耐久)

不填写`maxcharge`仅填写`charge`的情况下, `maxcharge`默认等于`charge`

关于该选项在Name和Lore中的显示, 请查看物品变量

自定义耐久

```yaml
test:
  material: STONE_SWORD
  options:
    durability: 10
```

`durability` 该物品自定义耐久

关于该选项在Name和Lore中的显示, 请查看物品变量

最大自定义耐久

```yaml
test:
  material: STONE_SWORD
  options:
    durability: 5
    maxdurability: 10
```

`maxdurability` 该物品最大自定义耐久

不填写`maxdurability`仅填写`durability`的情况下, `maxdurability`默认等于`durability`

关于该选项在Name和Lore中的显示, 请查看物品变量

自定义耐久耗尽后是否损坏

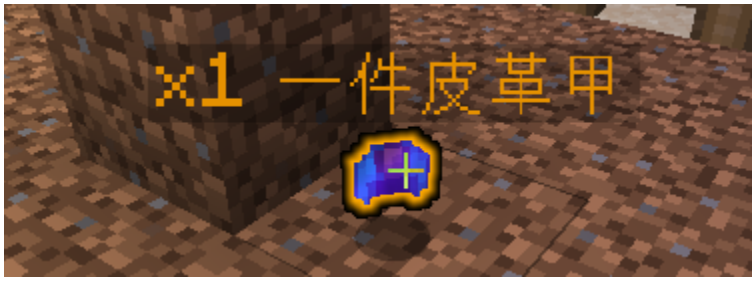
```yaml
test:
  material: STONE_SWORD
  options:
    durability: 5
    maxdurability: 10
    itembreak: false
```

`itembreak` 自定义耐久耗尽后是否损坏

关于该选项在Name和Lore中的显示, 请查看物品变量

物品光效

```yaml
ExampleItem:
  material: STONE
  options:
    color: GOLD
```



此选项可以使掉落物产生发光效果

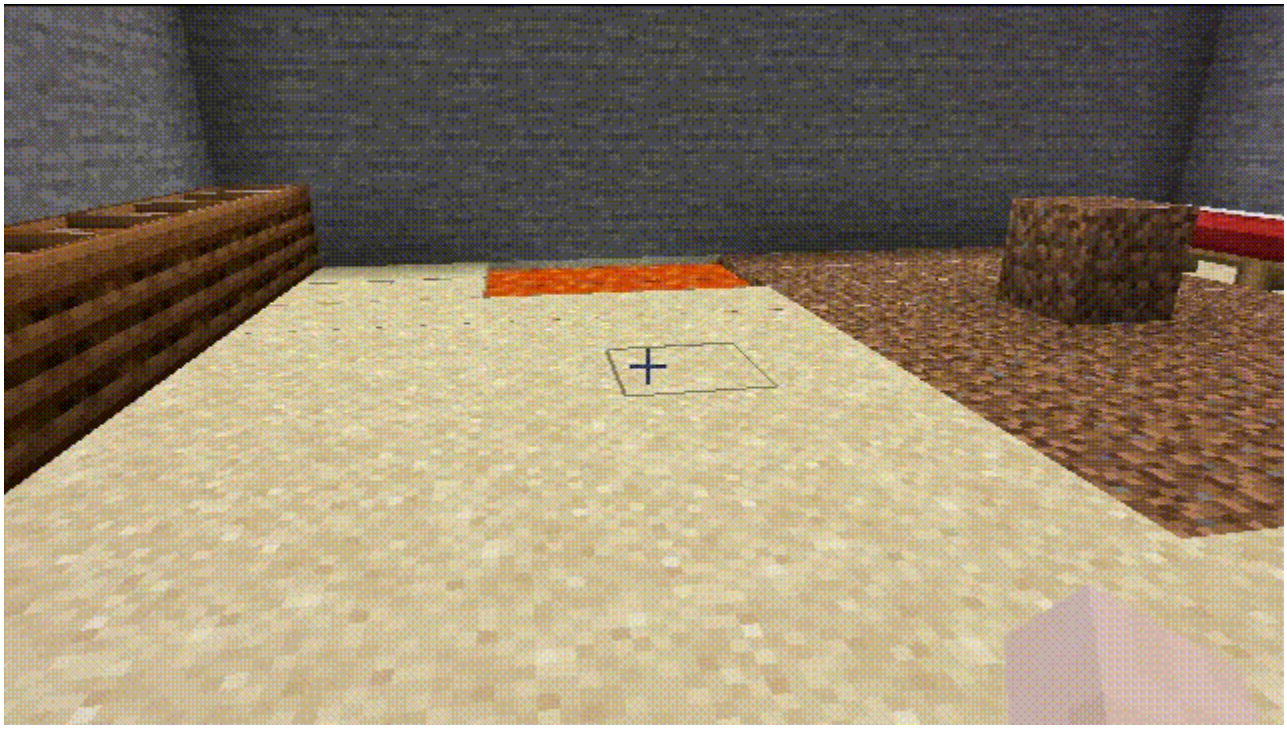
可用颜色有：

- AQUA
- BLACK
- BLUE
- DARK_AQUA
- DARK_BLUE
- DARK_GRAY
- DARK_GREEN
- DARK_PURPLE
- DARK_RED
- GOLD
- GRAY

- GREEN
- LIGHT_PURPLE
- RED
- WHITE
- YELLOW

掉落技能

```
ExampleItem:
  material: STONE
  options:
    dropskill: SkillTest
```



如图所示，此选项可使物品在掉落时触发MM技能。

只有通过`/ni drop`指令，以及通过击杀MM怪物掉落的NI物品才会触发，玩家主动丢弃不会。

作者并没有图中所示技能的版权，因此不在这里具体写出该技能。

掉落物归属

以默认配置为例

```
ownerTest:
  material: STONE
  name: 你捡我啊
  options:
    owner: Neige
```

上述物品通过`/ni drop`或击杀MM怪物掉落该物品, 该物品首次拾取只能由`Neige`完成

你可以将`owner`填写为`"%player_name%"`, 这样就是谁击杀就属于谁了

首次拾取后将不再有掉落物归属效果

服务器重启后效果重置 (掉了, 关服了, 再次开服, 谁都能捡)

> **提示**
>
> 通过/ni get或/ni give直接获取拥有掉落物归属的物品
>
> 物品将包含特殊NBT (用于记录归属人)
>
> 但通过/ni drop或击杀MM怪物掉落的物品将不包含该nbt (掉落的时候移除了)
>
> 你可以启用config中的`ItemOwner.removeNBTWhenGive`选项, 在/ni get或/ni give时

移除相关NBT

物品时限

```
物品ID:
  material: STONE
  options:
    itemtime: 物品时限(单位是秒)
```

以默认配置为例

```
itemTimeTest:
  material: STONE
  name: 限时物品-到期时间-<js::ItemTime.js::main_<itemtime>>
  options:
    itemtime: <itemtime>
  sections:
    itemtime: 60
```

搭配默认脚本

```
function main(time) {
    const date = new Date()
    date.setTime(date.getTime() + (Number(time) * 1000))
    return date.getFullYear() + "年" + (date.getMonth() + 1) + "月" + date.getDate() + "日"
+ date.getHours() + "时" + date.getMinutes() + "分" + date.getSeconds() + "秒"
}
```

可以生成形如`限时物品-到期时间-2022年8月11日21时59分5秒`的物品，物品到期即自动删除并提示信息。

如默认配置所示，你可以在对应位置放置一个节点，然后通过指向数据给予物品时自定义时长。

例如: `/ni give Neige itemTimeTest 1 true {"itemtime":"120"}` 将给予玩家一个剩余时间120秒的默认物品

物品动作

简介

通过左键/右键、食用/饮用、丢弃/捡起物品等方式，触发一系列物品动作（支持papi变量与动作变量）

配置项

配置

动作类型

全部基础动作支持papi变量, 动作类型不区分大小写

条件类型

原理

📄 动作变量

在物品动作中，你可以使用即时声明节点，并通过特殊的物品节点调用物品的nbt及节点缓存。

📄 自定义动作

自定义动作需要一定的 javascript 和 java 基础。

简介

通过左键/右键、食用/饮用、丢弃/捡起物品等方式，触发一系列物品动作（支持papi变量与动作变量）

可自定义每次是否消耗物品、消耗的物品数量、物品冷却、触发方式、触发条件

存放路径

所有物品动作配置文件应存放于`plugins/NeigeItems/ItemActions`文件夹

重复配置同一 ID 的物品不会导致报错，但可能互相覆盖

最后哪套动作活下来。。。随缘了属于是

配置项

配置

以默认指令配置为例

```yaml
ExampleItem2:
 left:
  actions:
   - "tell: 你正尝试触发&e ExampleItem2 &f物品"
   - condition: perm("item.ExampleItem2")
     actions:
      - "console: say &e%player_name% &f拥有&e item.ExampleItem2 &f权限"
      - "command: say 我拥有&e item.ExampleItem2 &f权限"
     deny:
      - "tell: 你没有&e item.ExampleItem2 &f权限"
  sync:
   - "tell: 你好, 这条消息通过主线程发送"
ExampleItem3:
 left:
  cooldown: 3000
  group: test2
  consume:
   condition: perm("item.ExampleItem3")
   amount: 1
   deny:
    - "tell: 你没有&e item.ExampleItem3 &f权限"
  actions:
   - "tell: 你正尝试触发&e ExampleItem3 &f物品"
```

结构

一个物品的动作配置可以表示为如下结构:

- 物品ID
  - 触发类型
    - 冷却时间(cooldown, 默认1000ms)
    - 冷却组(group)
    - 消耗信息(consume)
      - 消耗条件(condition)
      - 消耗数量(amount)
      - 不满足条件/数量不足时执行的动作(deny)
    - 同步执行的物品动作(sync)
    - 异步执行的物品动作(actions)

触发类型

触发类型	含义
left	手持物品左键点击
right	手持物品右键点击
all	手持物品左键或右键点击
shift_left	潜行状态手持物品左键点击

触发类型	含义
shift_right	潜行状态手持物品右键点击
shift_all	潜行状态手持物品左键或右键点击
eat	饮用/食用物品
drop	丢弃物品
pick	拾取物品
click	背包内使用该物品点击其他物品
beclicked	背包内使用其他物品点击该物品

冷却时间

冷却时间的单位是毫秒(ms), 不填写的话默认为1000ms, 支持解析动作变量及即时声明节点, 如:

```yaml
# 冷却时间为3秒
test1:
  left:
    cooldown: 3000
    actions:
      - "tell: 你好"
# 冷却时间为随机5-10秒
test2:
  left:
```

冷却组

冷却组默认为`触发类型-物品ID`, 如:

```yaml
test1:
  left:
    cooldown: 3000
    actions:
      - "tell: 你好"
test2:
  left:
    cooldown: 3000
    group: 'left-test1'
    actions:
      - "tell: 你好"
```

上述test1及test2物品的left触发器共享冷却时长

消耗信息

消耗信息中可配置消耗条件、消耗数量及不满足条件/数量不足时执行的动作, 如:

```yaml
test1:
  left:
    consume:
      # 检测是否持有 test1 权限
      condition: 'perm("test1")'
      # 消耗一个
      amount: 1
      # 未满足条件/数量不足时执行的动作
```

其中, amount支持代入动作变量及即时声明节点, 如:

```yaml
# 需消耗 test1 NBT值数量的物品
test1:
  left:
    consume:
      amount: <nbt::test1>
    actions:
      - 'tell: 物品消耗成功'
# 随机消耗5-10个物品
test2:
  left:
    consume:
      amount: <number::5_10>
    actions:
      - 'tell: 物品消耗成功'
# 消耗玩家等级数量个物品
test3:
  left:
    consume:
      amount: <papi::player_level>
    actions:
      - 'tell: 物品消耗成功'
# 消耗玩家等级数量个物品, 数据保留0位小数, 最小值限定为1, 最大值限定为10
test4:
  left:
    consume:
      amount: <fastcalc::<papi::player_level>_0_1_10>
    actions:
      - 'tell: 物品消耗成功'
```

deny项在条件不满足/数量不足时都会触发, 意味着你可以不书写condition项, 只写amount和deny:

```yaml
test1:
```

关于condition和actions的具体写法, 详见动作类型和条件类型

同步？异步？

节选默认配置为例：

```
# 一句话概括: 不想看的话, 优先使用"sync", 别用"actions"
#
# 从实际功能而言, “sync”与"actions"没有区别.
# 区别在于, "sync"下的所有内容都是同步解析, 同步触发的
#
# 即: 所有非线程安全的行为都应该在"sync"下进行
# 比如: 检测玩家是否拥有某个物品, 然后通过指令扣除
# 模拟情境: 你将A物品配置为"满足 papi("%checkitem_amount_mat:stone%") >= 1 时,
扣除一个石头, 给予100元"
#       如果你将这些动作配置到"actions"下, 那么可能出现:
#       判断玩家确实拥有大于等于1个石头, 然后玩家将石头丢出背包
#       后面扣除石头时, 因为玩家没有足够的石头, 扣除操作相当于失效
#       之后给予100元的行为却仍然执行, 玩家就成功在不消耗石头的情况下获得了100元
#       因此这些行为应该配置于"sync"下, 而非"actions"
#
# 注: "sync"下所有动作同步触发, 但这不代表"actions"下所有内容异步触发.
#    所有基础物品动作都作了相关判断, 比如takeHealth, takeFood, takeLevel等行为,
#    都会挪到主线程实施, 避免出现线程安全问题
sync:
 - "tell: 你好, 这条消息通过主线程发送"
```

动作类型

> 全部基础动作支持papi变量, 动作类型不区分大小写

动作写法

物品动作大致可归为三类: 列表式、字符式、条件式:

```yaml
# 字符式
actions: 'tell: 你好'
```

```yaml
# 列表式
actions:
- 'tell: 你好'
- 'tell: 你好'
- 'tell: 你好'
```

```yaml
# 条件式
actions:
 condition: 'perm("test")'
 actions:
 - 'tell: 你好'
 deny:
 - 'tell: 你好'
```

这三种形式可以任意组合, 在此我仅作一例:

```yaml
# 列表式组合条件式
actions:
- 'tell: 你好'
- condition: 'perm("test")'
  actions:
  - 'tell: 你好'
  deny:
  - 'tell: 你好'
- 'tell: 你好'
```

发送文本

> 向玩家发送一条消息(可使用&作为颜色符号)

```yaml
- 'tell: &eHello'
```

发送文本

> 向玩家发送一条消息(不将&解析为颜色符号)

```yaml
- 'tellNoColor: §eHello, can you see "&"?'
```

强制聊天

> 强制玩家发送一条消息(不将&解析为颜色符号)

```yaml
- 'chat: see, I can send "&"!'
```

强制聊天

> 强制玩家发送一条消息(可使用&作为颜色符号)

```
- 'chatWithColor: &eHello'
```

执行指令(玩家)

> 强制玩家执行一条指令(可使用&作为颜色符号)

```
- 'command: say Hello'
- 'player: say Hello'
```

执行指令(玩家)

> 强制玩家执行一条指令(不将&解析为颜色符号)

```
- 'commandNoColor: say Hello'
```

执行指令(后台)

> 后台执行一条指令(可使用&作为颜色符号)

```
- 'console: say Hello'
```

执行指令(后台)

> 后台执行一条指令(不将&解析为颜色符号)

```
- 'consoleNoColor: say Hello'
```

发送标题

> 向玩家发送一条标题(可使用&作为颜色符号)

```
- 'title: Title SubTitle FadeIn Stay FadeOut'
```

- `Title` 主标题
- `SubTitle` 次标题, 默认为空
- `FadeIn` 标题显现消耗的时间(以tick为单位, 20tick = 1s), 默认10
- `Stay` 标题持续显示的时间(以tick为单位, 20tick = 1s), 默认70
- `FadeOut` 标题消失消耗的时间(以tick为单位, 20tick = 1s), 默认20

`SubTitle` `FadeIn` `Stay` `FadeOut`存在默认值, 可以不填写

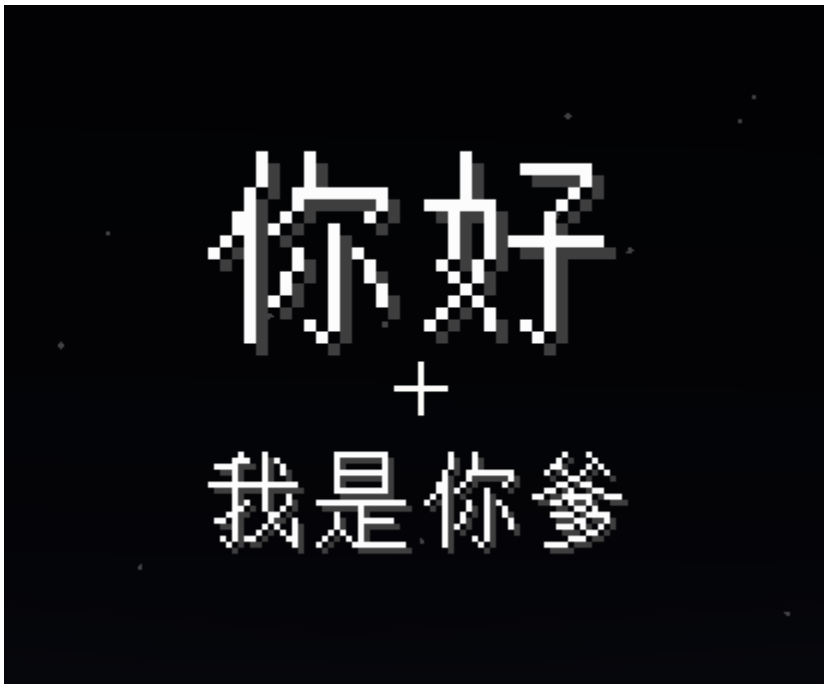
参数之间以空格做分隔, 文本形式的空格需要通过反斜杠转义, 如:

```
- 'title: 你好\ 我是你爹'
```

效果:

```
- 'title: 你好 我是你爹'
```

效果:



发送标题

> 向玩家发送一条标题(不将&解析为颜色符号)

```
- 'titleNoColor: Title SubTitle FadeIn Stay FadeOut'
```

- `Title` 主标题
- `SubTitle` 次标题
- `FadeIn` 标题显现消耗的时间(以tick为单位, 20tick = 1s), 默认10
- `Stay` 标题持续显示的时间(以tick为单位, 20tick = 1s), 默认70
- `FadeOut` 标题消失消耗的时间(以tick为单位, 20tick = 1s), 默认20

发送ActionBar

> 向玩家发送一条ActionBar消息(可使用&作为颜色符号)

```
- 'actionBar: &eHello'
```

发送ActionBar

> 向玩家发送一条ActionBar消息(不将&解析为颜色符号)

```
- 'actionBarNoColor: §eHello, can you see "&"?'
```

给予金币(Vault)

> 给予玩家一定数量金币

```
- 'giveMoney: 100'
```

扣除金币(Vault)

> 扣除玩家一定数量金币

```
- 'takeMoney: 100'
```

给予经验

> 给予玩家一定数量经验

```
- 'giveExp: 100'
```

扣除经验

> 扣除玩家一定数量经验

```
- 'takeExp: 100'
```

设置经验

> 设置玩家当前经验

```
- 'setExp: 100'
```

给予经验等级

> 给予玩家一定数量经验等级

```
- 'giveLevel: 100'
```

扣除经验等级

> 扣除玩家一定数量经验等级

```
- 'takeLevel: 100'
```

设置经验等级

> 设置玩家当前经验等级

```
- 'setLevel: 100'
```

给予饱食度

> 给予玩家一定数量饱食度

```
- 'giveFood: 5'
```

扣除饱食度

> 扣除玩家一定数量饱食度

```
- 'takeFood: 5'
```

设置饱食度

> 设置玩家当前饱食度

```
- 'setFood: 20'
```

给予饱和度

> 给予玩家一定数量饱和度(玩家饱和度不能超过饱食度)

```
- 'giveSaturation: 5'
```

扣除饱和度

> 扣除玩家一定数量饱和度(玩家饱和度不能超过饱食度)

```
- 'takeSaturation: 5'
```

设置饱和度

> 设置玩家当前饱和度(玩家饱和度不能超过饱食度)

```
- 'setSaturation: 20'
```

给予生命值

> 给予玩家一定数量生命值

```
- 'giveHealth: 5'
```

扣除生命值

> 扣除玩家一定数量生命值

```
- 'takeHealth: 5'
```

设置生命值

> 设置玩家当前生命值

```
- 'setHealth: 20'
```

释放MM技能

> 释放MM技能, 对创造模式玩家无效

```
- 'castSkill: 技能名称'
```

组合技记录

> 在对应组记录当前触发技能

```
- 'combo: 触发组 触发ID'
```

语言描述较为抽象, 在此我以默认配置为例, 实现左键-右键-左键触发连击技的示范:

```
ComboTest:
 left:
  sync:
   # 在ComboTest组记录, 触发了类型为left的连击
   - "combo: ComboTest left"
   # 检测ComboTest组是否完成了left-right-left连击
   - condition: combo("ComboTest", ["left", "right", "left"])
    actions:
     # 进行对应操作
     - 'tell: &e连击 &bL &f+ &bR &f+ &bL'
     # 已达成最终需要的连击, 清空ComboTest组的连击记录
     - 'comboClear: ComboTest'
    deny:
     # 检测ComboTest组是否完成了left连击
     condition: combo("ComboTest", ["left"])
     actions:
```

组合技清除

> 清除对应组的技能记录

```
- 'comboClear: 触发组'
```

延时

> 延迟动作执行(单位是tick)

```
- 'delay: 10'
```

终止

> 终止动作执行

```
- 'return'
```

JavaScript

> 执行一段javascript代码

```
- 'js: player.sendMessage("你好")'
```

> ⚠ **警告**
>
> 不要在js动作中进行变量声明!!!
>
> 如:
>
> ```
> test = 1
> var test = 1
> let test = 1
> const test = 1
> ```
>
> 这将产生严重的线程安全问题

对此, 我提供了替代方案: 将变量存放于默认提供的名为`variables`的HashMap如:

```
variables["test"] = 1
```

如果你想让一个变量传递到下一条js动作/condition中, 可以使用global

```
global["test"] = 1
```

js动作实际上与动作条件运行于同一环境, 因此你可以使用动作条件中的所有对象/类/函数, 例如:

```
// 调用player对象, 发送消息"你好"
player.sendMessage("你好")
```

条件类型

原理

条件解析本质上是执行一段javascript代码, 为了追求更高的性能, 我对用于解析条件的脚本引擎做了特殊设置。

因此需要注意: 不要在condition内进行变量声明

请使用`variables["test"] = 1`代替`var test = 1`

为降低javascript上手门槛, 我内置了很多用于条件判断的函数。

注意

警告

不要在condition内进行变量声明!!!

如:

```
test = 1
var test = 1
let test = 1
const test = 1
```

这将产生严重的线程安全问题

对此, 我提供了替代方案: 将变量存放于默认提供的名为`variables`的HashMap如:

```
variables["test"] = 1
```

如果你想让一个变量传递到下一条js动作/condition中, 可以使用global

```
global["test"] = 1
```

默认存在的类/对象

类

`java.util.Calendar`

`java.util.concurrent.ThreadLocalRandom`

`org.bukkit.Bukkit`

`org.bukkit.ChatColor`

`org.bukkit.GameMode`

`org.bukkit.Material`

`pers.neige.neigeitems.utils.ActionUtils`

`pers.neige.neigeitems.utils.ConfigUtils`

`pers.neige.neigeitems.utils.FileUtils`

`pers.neige.neigeitems.utils.ItemUtils`

`pers.neige.neigeitems.utils.JsonUtils`

`pers.neige.neigeitems.utils.LangUtils`

`pers.neige.neigeitems.utils.PlayerUtils`

`pers.neige.neigeitems.utils.SamplingUtils`

`pers.neige.neigeitems.utils.ScriptUtils`

`pers.neige.neigeitems.utils.SectionUtils`

`pers.neige.neigeitems.utils.StringUtils`

`pers.neige.neigeitems.manager.HookerManager`

单例

`ActionManager` = `pers.neige.neigeitems.manager.ActionManager.INSTANCE`

`ConfigManager` = `pers.neige.neigeitems.manager.ConfigManager.INSTANCE`

`ItemEditorManager` = `pers.neige.neigeitems.manager.ItemEditorManager.INSTANCE`

`ItemManager` = `pers.neige.neigeitems.manager.ItemManager.INSTANCE`

`ItemPackManager` = `pers.neige.neigeitems.manager.ItemPackManager.INSTANCE`

对象

`bukkitScheduler` = `Bukkit.getScheduler()`

`bukkitServer` = `Bukkit.getServer()`

`consoleSender` = `bukkitServer.getConsoleSender()`

`pluginManager` = `Bukkit.getPluginManager()`

`plugin` = `pluginManager.getPlugin("NeigeItems")`

`player` = `触发玩家`

`itemStack` = `触发物品`

`itemTag` = `触发物品NBT`

`event` = `触发事件`

`variables` = `HashMap<String, Any?>()` 每条condition都是一个单独的variables

`global` = `HashMap<String, Any?>()` 每套动作(或者说每一次触发)共用一个global

`value` = `仅存在与check节点: check节点中传入的值`

同时满足(&&)

> 同时满足多个条件

```
condition: '条件1 && 条件2'
```

示例:

```
# 持有权限 test1 及 test2
condition: 'perm("test1") && perm("test2")'
```

满足一个(||)

> 满足多个条件中的一个

```
condition: '条件1 || 条件2'
```

示例:

```
# 持有权限 test1 或持有权限 test2
condition: 'perm("test1") || perm("test2")'
```

同时满足与满足一个嵌套使用

> 懂不懂括号的含金量

```
condition: '(条件1 || 条件2) && 条件3'
```

示例:

```
# 持有权限 test1 或持有权限 test2 的同时, 持有权限test3
condition: '(perm("test1") || perm("test2")) && perm("test3")'
```

是否相等(==, ===)

> 字符串记得用引号包起来

```
# papi变量 %player_name% 的解析值是否等于 Neige
condition: 'papi("%player_name%") == "Neige"'
condition: 'papi("%player_name%") === "Neige"'
```

== 比较的是值, ===比较的是值和类型

```
# 满足条件
condition: '"10" == 10'
# 不满足条件, 因为一个是字符串一个是数字
condition: '"10" === 10'
```

不想动脑子可以无脑使用==

大小判断(><)

> 字符串记得用引号包起来

```
# 懂?
condition: '10 > 10'
condition: '10 < 10'
condition: '10 >= 10'
condition: '10 <= 10'

# papi变量 %player_level% 的解析值是否大于等于 10
condition: 'Number(papi("%player_level%")) >= 10'
```

字符串转数字(Number, parseInt, parseFloat)

> 字符串记得用引号包起来

```
condition: 'Number("10") === 10'
condition: 'parseInt("10") === 10'
condition: 'parseFloat("10.0") === 10.0'
```

权限检测(perm)

> 字符串记得用引号包起来

```
# 玩家是否拥有 权限名 权限
condition: 'perm("权限名")'
```

替换颜色代码(color)

> 字符串记得用引号包起来

```
condition: 'color("&7nb666") == "§7nb666"'
```

解析PAPI变量(papi)

> 字符串记得用引号包起来

```
# papi变量 %player_name% 的解析值是否等于 Neige
condition: 'papi("%player_name%") == "Neige"'
```

解析即时节点(parse)

> 字符串记得用引号包起来

```
# 即时节点 <strings::test1_test2> 的解析值是否等于 test1
condition: 'parse("<strings::test1_test2>") == "test1"'
```

解析动作变量(parseItem)

> 字符串记得用引号包起来

```
# 检测test1这条NBT的值是否等于"666"
condition: 'parseItem("<nbt::test1>") == "666"'

# 检测test1这条data的值是否等于"666"
condition: 'parseItem("<data::test1>") == "666"'
```

获取指向数据(data)

> 字符串记得用引号包起来

```
# 当前NI物品名为"test"的节点值是否等于"666"
condition: 'data["test"] == "666"'
```

获取NBT文本(getNBT)

> 字符串记得用引号包起来

```
# getNBT获取的NBT值全是转成字符串的
# 检测test这条NBT的值是否等于"666"
condition: 'getNBT("test") == "666"'

# 可以用.分隔不同层级的键
# 当前检测的NBT对应在物品中体现为:
# test:
#   test: "666"
condition: 'getNBT("test.test") == "666"'
```

获取NBT值(getNBTTag)

> 字符串记得用引号包起来

`getNBTTag`获取的NBT都是ItemTag的形式, 需要你自行转换后对比

如:

```
getNBTTag("test").asString() == "666"
getNBTTag("test").asDouble() == 666
getNBTTag("test").asInt() == 666
getNBTTag("test").asFloat() == 666
getNBTTag("test").asByte() == 1
```

```
# getNBT获取的NBT值全是转成字符串的
```

随机数(random)

```
# 生成一个0-1的随机数, 检测其是否大于等于0.5
condition: 'random() >= 0.5'

# 生成一个5-10的随机数, 检测其是否大于等于7
condition: 'random(5, 10) >= 7'
```

随机概率(chance)

```
# 50%返回满足条件
condition: 'chance(0.5)'

# 50%返回满足条件
condition: 'chance(50, 100)'
```

连击检测(chance)

```
# 检测ComboTest组是否完成了left-right-left连击
condition: 'combo("ComboTest", ["left", "right", "left"])'
```

请结合组合技记录了解

默认500ms内点击算作连击, 可通过配置文件修改

包含某字符(contains)

> 字符串记得用引号包起来

```
# 测试测试 包含 测试 时返回true
condition: '"测试测试".contains("测试")'

# 即时节点 <strings::测试_别测试了> 的解析值是否包含 测试
condition: 'parse("<strings::测试_别测试了>").contains("测试")'
```

玩家IP(address)

> 字符串记得用引号包起来

```
condition: 'address() == "127.0.0.1"'
```

获取/修改飞行能力(allowFlight)

```
# 玩家可以飞行(双击空格起飞)
condition: 'allowFlight()'

# 设置玩家可以飞行, 并返回true满足条件
condition: 'allowFlight(true)'
```

获取/修改飞行状态(fly)

```
# 玩家正在飞行
condition: 'fly()'

# 设置玩家正在飞行, 并返回true满足条件
condition: 'fly(true)'
```

飞行速度(flySpeed)

```
# 玩家飞行速度是否等于1
condition: 'flySpeed() == 1'

# 将玩家飞行速度设置为10, 然后判断一下是否等于10以满足条件
condition: 'flySpeed(100) == 10'
```

行走速度(walkSpeed)

```
# 玩家行走速度是否等于1
condition: 'walkSpeed() == 1'

# 将玩家行走速度设置为10, 然后判断一下是否等于10以满足条件
condition: 'walkSpeed(100) == 10'
```

攻击冷却(attackCooldown)

```
# 是否冷却完毕
condition: 'attackCooldown() == 1'
```

重生点坐标(bedSpawnX/Y/Z)

```
# 重生点位于1, 1, 1
condition: 'bedSpawnX() == 1 && bedSpawnY() == 1 && bedSpawnY() == 1'
```

格挡状态(blocking)

```
# 玩家是否正在格挡
condition: 'blocking()'
```

指南针坐标(compassTargetX/Y/Z)

```
# 指南针坐标位于1, 1, 1
condition: 'compassTargetX() == 1 && compassTargetY() == 1 && compassTargetY() ==
1'
```

本月日期(day/dayOfMonth)(1-31)

```
# 今儿1号
condition: 'day() == 1'
```

本周日期(dayOfWeek)(1-7)

```
# 今儿周一
condition: 'dayOfWeek() == 1'
```

本年日期(dayOfYear)(1-365)

```
# 今儿元旦
condition: 'dayOfYear() == 1'
```

月份(month)(1-12)

```
# 今儿一月
condition: 'month() == 1'
```

年份(year)

```
# 今儿3202年了
condition: 'year() == 3202'
```

小时(hour)

```
# 现在八点
condition: 'hour() == 8'
```

分钟(minute)

```
# 这小时刚过去15分钟
condition: 'minute() == 15'
```

秒(second)

```
# 这分钟刚过去15秒
condition: 'second() == 15'
```

本月周数(weekInMonth)

```
# 现在是本月第一周
condition: 'weekInMonth() == 1'
```

上午/下午(amOrPm)(0/1)

```
# 现在是上午
condition: 'amOrPm() == 0'

# 现在是下午
condition: 'amOrPm() == 1'
```

时间戳(time)

```
# 1970年1月1日0时0分0秒到现在, 经过了1000000000毫秒
condition: 'time() == 1000000000'
```

死亡状态(dead)

```
# 玩家是否死亡
condition: 'dead()'
```

是否首次登录(firstPlay)

```
# 玩家是否首次登录
condition: 'firstPlay()'
```

疲劳度(exhaustion)

```
# 玩家疲劳度是否等于0.1
condition: 'exhaustion() == 0.1'

# 将玩家疲劳度设置为0.1, 然后判断一下是否等于0.1以满足条件
condition: 'exhaustion(0.1) == 0.1'
```

获取/修改经验值(exp)

```
# 玩家经验值是否等于100
condition: 'exp() == 100'

# 将玩家经验值设置为100, 然后判断一下是否等于100以满足条件
condition: 'exp(100) == 100'
```

给予经验(addExp)

```
# 给予100经验(本函数必定返回null)
condition: 'addExp(100) == null'
```

扣除经验(takeExp)

```
# 扣除100经验(本函数必定返回null)
condition: 'takeExp(100) == null'

# 扣除100经验,然后判断玩家经验值是否等于100
condition: 'takeExp(100);exp() == 100'
```

获取/修改等级(level)

```
# 玩家等级是否等于100
condition: 'level() == 100'

# 将玩家等级设置为100, 然后判断一下是否等于100以满足条件
condition: 'level(100) == 100'
```

给予等级(addLevel)

```
# 给予100等级(本函数必定返回null)
condition: 'addLevel(100) == null'

# 给予100等级,然后判断玩家等级是否等于100
condition: 'addLevel(100);level() == 100'
```

扣除等级(takeLevel)

```
# 扣除100等级(本函数必定返回null)
condition: 'takeLevel(100) == null'

# 扣除100等级,然后判断玩家等级是否等于100
condition: 'takeLevel(100);level() == 100'
```

获取/修改饥饿度(level)

```
# 玩家饥饿度是否等于10
condition: 'food() == 10'

# 将玩家饥饿度设置为10, 然后判断一下是否等于10以满足条件
condition: 'food(10) == 10'
```

给予饥饿度(addFood)

```
# 给予100饥饿度(本函数必定返回null)
condition: 'addFood(00) == null'

# 给予10饥饿度,然后判断玩家饥饿度是否等于10
condition: 'addFood(10);food() == 10'
```

扣除饥饿度(takeFood)

```
# 扣除100饥饿度(本函数必定返回null)
condition: 'takeFood(10) == null'

# 扣除10饥饿度,然后判断玩家饥饿度是否等于10
condition: 'takeFood(10);food() == 10'
```

获取/修改游戏模式(gamemode)(ADVENTURE/CREATIVE/SPECTATOR/SURVIVAL)

> 字符串记得用引号包起来

```
# 玩家是否处于生存模式
condition: 'gamemode() == "SURVIVAL"'

# 将玩家设置为生存模式, 然后判断是否处于生存模式, 以满足条件
condition: 'gamemode("SURVIVAL");gamemode() == "SURVIVAL"'
```

获取/修改滑翔状态(guilding)

```
# 玩家正在滑翔
condition: 'guilding()'

# 设置玩家正在滑翔, 并返回true满足条件
```

获取/修改发光状态(glowing)

```
# 玩家正在发光
condition: 'glowing()'

# 设置玩家正在发光, 并返回true满足条件
condition: 'glowing(true)'
```

获取/修改重力状态(gravity)

```
# 玩家是否拥有重力
condition: 'gravity()'

# 设置玩家拥有重力, 并返回true满足条件
condition: 'gravity(true)'
```

生命值(health)

```
# 玩家当前生命值大于10
condition: 'health() > 10'
```

最大生命值(maxHealth)

```
# 玩家当前最大生命值大于10
condition: 'maxHealth() > 10'
```

玩家名(name)

> 字符串记得用引号包起来

```
# 玩家名为Neige
condition: 'name() == "Neige"'
```

获取/修改剩余氧气(remainingAir)

```
# 玩家剩余氧气是否等于10
condition: 'remainingAir() == 10'

# 将玩家剩余氧气设置为10, 然后判断一下是否等于10以满足条件
condition: 'remainingAir(10) == 10'
```

睡觉状态(sleeping)

```
# 玩家是否正在睡觉
condition: 'sleeping()'
```

获取/修改潜行状态(sneaking)

```
# 玩家正在潜行
condition: 'sneaking()'

# 设置玩家正在潜行, 并返回true满足条件
condition: 'sneaking(true)'
```

获取/修改疾跑状态(sprinting)

```
# 玩家正在疾跑
condition: 'sprinting()'

# 设置玩家正在疾跑, 并返回true满足条件
condition: 'sprinting(true)'
```

获取/修改游泳状态(swimming)

```
# 玩家正在游泳
condition: 'swimming()'

# 设置玩家正在游泳, 并返回true满足条件
condition: 'swimming(true)'
```

所处世界(world)

> 字符串记得用引号包起来

```
# 玩家正处在名为world的世界中
condition: 'world() == "world"'
```

动作变量

在物品动作中，你可以使用即时声明节点，并通过特殊的物品节点调用物品的nbt及节点缓存。

以默认配置为例：

```yaml
actionTest:
 material: STONE
 nbt:
  test1: "666"
  test2:
   test3: "777"
  test4:
  - "888"
  - "999"
 sections:
  test: "000"
```

```yaml
actionTest:
 all:
 - "console: say 名为test1的NBT的值为: <nbt::test1>"
 - "console: say 名为test2.test3的NBT的值为: <nbt::test2.test3>"
 - "console: say 名为test4.0的NBT的值为: <nbt::test4.0>"
 - "console: say 名为test4.1的NBT的值为: <nbt::test4.1>"
 - "console: say 名为test的节点的值为: <data::test>"
 - "console: say 随机数尝试: <number::0_10_2>"
```

后台返回值如下

```
[Server] 名为test1的NBT的值为: 666
[Server] 名为test2.test3的NBT的值为: 777
[Server] 名为test4.0的NBT的值为: 888
[Server] 名为test4.1的NBT的值为: 999
[Server] 名为test的节点的值为: 000
[Server] 随机数尝试: 0.74
```

用法类似于即时声明节点，data表示调用节点，nbt表示调用物品nbt。

一层一层id以小数点`.`分隔

如果id中存在`.`, 请在书写时通过`\.`代替, 这里的反斜杠不是转义符, 因此请注意当前最外层括号的引号类型

如:

```yaml
actionTest:
  all:
  - "console: say <nbt::te\\.st1.test2>"
  - 'console: say <nbt::te\.st1.test2>'
```

上面两行动作取的都是`te.st1`下的`test2`的值

yaml语法中双引号包裹的`\`代表转义符, `\\`才是反斜杠

单引号包裹的情况下则所见即所得

自定义动作

自定义动作需要一定的 javascript 和 java 基础。

自定义动作文件存放于`NeigeItems/CustomSections`文件夹

下面是示例配置

```
// 文件名不重要, 写成啥都行
// main函数会自动执行
function main() {
  // 导入相应的类, 这两行看不懂的话直接抄就行
  const ActionManager =
Packages.pers.neige.neigeitems.manager.ActionManager.INSTANCE
  const SectionUtils = Packages.pers.neige.neigeitems.utils.SectionUtils

  // 插入新的自定义动作
  ActionManager.addAction(
    // 动作名称
    "test",
    // 动作内容(一般是异步调用的, 所以需要同步执行的内容需要自行同步)
    function(player, string) {
      // 调用动作
      ActionManager.runAction(player, "tell: 123")
      ActionManager.runAction(player, "tell: 456")
      player.sendMessage(SectionUtils.parseSection("<number::0_10_2>"))
      // 每个动作都一定要返回一个布尔量(true或false)
      return true
    })
}
```

物品编辑函数

简介

通过物品编辑指令编辑你手中的物品。详见函数类型

函数类型

动作ID不区分大小写

自定义函数

自定义函数需要一定的 javascript 和 java 基础。

简介

通过物品编辑指令编辑你手中的物品。详见函数类型

函数类型

> 动作ID不区分大小写

材质

> 给物品设置材质

函数ID: `setMaterial`

函数参数: `材质的Bukkit英文ID`

参数示例: `STONE`

示例解析: `待设置物品材质将变为石头`

> **信息**
>
> 所有ID可查看: https://hub.spigotmc.org/javadocs/spigot/org/bukkit/Material.html
>
> 但Neige更加推荐: 使用`/ni save`指令保存对应物品, 然后前往物品配置查看material

材质(解析papi变量)

> 给物品设置材质

函数ID: `setMaterialPapi`

函数参数: `材质的Bukkit英文ID`

参数示例: `%player_name%`

示例解析: `如果你的ID是STONE, 那么待设置物品材质将变为石头`

> 信息
>
> 很明显, 解析papi变量会消耗额外的性能, 所以没有特殊需求可以使用`setMaterial`函数

材质(解析动作变量)

> 给物品设置材质

函数ID: `setMaterialSection`

函数参数: `材质的Bukkit英文ID`

参数示例: `<strings::STONE_SUGAR>`

示例解析: `待设置物品将随机变为石头或糖`

> 信息
>
> 很明显, 解析随机节点会消耗更多的性能, 所以没有特殊需求可以使用`setMaterial`或`setMaterialPapi`函数
>
> 当前函数中的papi变量可以通过<papi::变量内容>表示, 形如<papi::player_name>

设置数量

> 给物品设置数量

函数ID: `setAmount`

函数参数: `目标数量`

参数示例: `1`

示例解析: `待设置物品数量将变为1`

> 信息
>
> 目标数量无法超过物品最大堆叠数, 无法小于0, 等于0将销毁物品

设置数量(解析papi变量)

> 给物品设置数量

函数ID: `setAmountPapi`

函数参数: `目标数量`

参数示例: `%player_level%`

示例解析: `假设我的等级是10, 那么待设置物品数量将变为10`

> 信息
>
> 目标数量无法超过物品最大堆叠数, 无法小于0, 等于0将销毁物品

设置数量(解析动作变量)

> 给物品设置数量

函数ID: `setAmountSection`

函数参数: `目标数量`

参数示例: `<number::1_10>`

示例解析: `待设置物品数量将随机变为1-10`

> 信息
>
> 目标数量无法超过物品最大堆叠数, 无法小于0, 等于0将销毁物品

添加数量

> 给物品添加数量

函数ID: `addAmount`

函数参数: `添加数量`

参数示例: `1`

示例解析: `待设置物品数量将增加1`

> 信息

> 目标数量无法超过物品最大堆叠数, 无法小于0, 等于0将销毁物品
>
> 这只影响当前物品, 不会对玩家背包其他物品造成影响, `addAmount`一万不会给予玩家10000个物品, 只会让当前物品达到堆叠上限

添加数量(解析papi变量)

> 给物品添加数量

函数ID: `addAmountPapi`

函数参数: `添加数量`

参数示例: `%player_level%`

示例解析: `假设我的等级是10, 那么待设置物品数量将增加10`

> 信息
>
> 目标数量无法超过物品最大堆叠数, 无法小于0, 等于0将销毁物品
>
> 这只影响当前物品, 不会对玩家背包其他物品造成影响, `addAmount`一万不会给予玩家10000个物品, 只会让当前物品达到堆叠上限

添加数量(解析动作变量)

> 给物品添加数量

函数ID: `addAmountSection`

函数参数: `添加数量`

参数示例: `<number::1_10>`

示例解析: `待设置物品数量将随机增加1-10`

> 信息
>
> 目标数量无法超过物品最大堆叠数, 无法小于0, 等于0将销毁物品
>
> 这只影响当前物品, 不会对玩家背包其他物品造成影响, `addAmount`一万不会给予玩家10000个物品, 只会让当前物品达到堆叠上限

扣除数量

> 给物品扣除数量

函数ID: `takeAmount`

函数参数: `扣除数量`

参数示例: `1`

示例解析: `待设置物品数量将减少1`

> 信息
>
> 目标数量无法超过物品最大堆叠数, 无法小于0, 等于0将销毁物品
>
> 这只影响当前物品, 不会对玩家背包其他物品造成影响, `takeAmount`一万不会扣除玩家背包所有物品, 只会让当前物品消失

扣除数量(解析papi变量)

> 给物品扣除数量

函数ID: `takeAmountPapi`

函数参数: `扣除数量`

参数示例: `%player_level%`

示例解析: `假设我的等级是10, 那么待设置物品数量将减少10`

> **信息**
>
> 目标数量无法超过物品最大堆叠数, 无法小于0, 等于0将销毁物品
>
> 这只影响当前物品, 不会对玩家背包其他物品造成影响, `takeAmount`一万不会扣除玩家背包所有物品, 只会让当前物品消失

扣除数量(解析动作变量)

> 给物品扣除数量

函数ID: `takeAmountSection`

函数参数: `扣除数量`

参数示例: `<number::1_10>`

示例解析: `待设置物品数量将随机减少1-10`

> 信息
>
> 目标数量无法超过物品最大堆叠数, 无法小于0, 等于0将销毁物品
>
> 这只影响当前物品, 不会对玩家背包其他物品造成影响, `takeAmount` 一万不会扣除玩家背包所有物品, 只会让当前物品消失

显示名

> 给物品设置显示名称

函数ID: `setName`

函数参数: `待设置显示名称`

参数示例: `&e测试物品`

示例解析: `待设置物品的显示名称将变为 § e测试物品`

> 信息
>
> 文本中的&将被自动替换为颜色符号 §

显示名(解析papi变量)

> 给物品设置显示名称

函数ID: `setNamePapi`

函数参数: `待设置显示名称`

参数示例: `&e%player_name%`

示例解析: `我的玩家ID是Neige, 所以待设置物品的显示名称将变为 § eNeige`

> 信息
>
> 文本中的&将被自动替换为颜色符号 §

显示名(解析动作变量)

> 给物品设置显示名称

函数ID: `setNameSection`

函数参数: `待设置显示名称`

参数示例: `&e<strings::测试物品1_测试物品2>`

示例解析: `待设置物品的显示名称将变为 § e测试物品1或 § e测试物品2`

> 信息
>
> 文本中的&将被自动替换为颜色符号 §

显示名前缀

> 为物品显示名称添加前缀

函数ID: `addNamePrefix`

函数参数: `待添加前缀`

参数示例: `&4史诗-`

示例解析: `假设原先物品的显示名为§e测试物品, 函数执行后将变为§4史诗-§e测试物品`

> 信息
>
> 文本中的&将被自动替换为颜色符号§

显示名前缀(解析papi变量)

> 为物品显示名称添加前缀

函数ID: `addNamePrefixPapi`

函数参数: `待添加前缀`

参数示例: `&4%player_name%的-`

示例解析: `我的玩家ID是Neige, 假设原先物品的显示名为§e测试物品, 那么物品的显示名称将变为§4Neige的-§e测试物品`

> 信息
>
> 文本中的&将被自动替换为颜色符号§

显示名前缀(解析动作变量)

> 为物品显示名称添加前缀

函数ID: `addNamePrefixSection`

函数参数: `待添加前缀`

参数示例: `&e<strings::&4史诗-_&f垃圾->`

示例解析: `假设原先物品的显示名为§e测试物品, 函数执行后将随机变为&4史诗-§e测试物品或&f垃圾-§e测试物品`

> 信息
>
> 文本中的&将被自动替换为颜色符号§

显示名后缀

> 为物品显示名称添加后缀

函数ID: `addNamePostfix`

函数参数: `待添加后缀`

参数示例: `-后缀`

示例解析: `假设原先物品的显示名为§e测试物品, 函数执行后将变为§e测试物品-后缀`

> 信息
>
> 文本中的&将被自动替换为颜色符号§

显示名后缀(解析papi变量)

> 为物品显示名称添加后缀

函数ID: `addNamePostfixPapi`

函数参数: `待添加后缀`

参数示例: `-%player_name%`

示例解析: `我的玩家ID是Neige, 假设原先物品的显示名为§e测试物品, 那么物品的显示名称将变为§e测试物品-Neige`

> 信息
>
> 文本中的&将被自动替换为颜色符号§

显示名后缀(解析动作变量)

> 为物品显示名称添加后缀

函数ID: `addNamePostfixSection`

函数参数: `待添加后缀`

参数示例: `&e<strings::-后缀1_-后缀2>`

示例解析: `假设原先物品的显示名为§e测试物品, 函数执行后将随机变为§e测试物品-后缀1或§e测试物品-后缀2`

> 信息
>
> 文本中的&将被自动替换为颜色符号§

替换显示名(替换一次)

> 替换物品显示名中的对应文本(只替换一次)

函数ID: `replaceName`

函数参数: `json形式的"待替换文本":"替换文本"`

参数示例: `{"A":"B","C":"D"}`

示例解析: `假设物品原先名为"AACC", 替换后将变为"BADC"`

> 信息
>
> 文本中的&将被自动替换为颜色符号 §

替换显示名(替换一次, 解析papi变量)

> 替换物品显示名中的对应文本(只替换一次)

函数ID: `replaceNamePapi`

函数参数: `json形式的"待替换文本":"替换文本"`

参数示例: `{"玩家名":"%player_name%"}`

示例解析: `假设物品原先名为"玩家名的物品", 我的玩家ID是Neige, 替换后名称将变为"Neige的物品"`

> 信息

> 文本中的&将被自动替换为颜色符号 §

替换显示名(替换一次, 解析动作变量)

> 替换物品显示名中的对应文本(只替换一次)

函数ID: `replaceNameSection`

函数参数: `json形式的"待替换文本":"替换文本"`

参数示例: `{"品质":"<strings::普通_精良>"}`

示例解析: `假设物品原先名为"品质 长剑", 替换后名称将随机变为"普通 长剑"或"精良 长剑"`

> 信息
>
> 文本中的&将被自动替换为颜色符号 §

替换显示名(替换全部)

> 替换物品显示名中的对应文本(替换全部)

函数ID: `replaceAllName`

函数参数: `json形式的"待替换文本":"替换文本"`

参数示例: `{"A":"B","C":"D"}`

示例解析: `假设物品原先名为"AACC", 替换后将变为"BBDD"`

> 信息
>
> 文本中的&将被自动替换为颜色符号 §

替换显示名(替换全部, 解析papi变量)

> 替换物品显示名中的对应文本(替换全部)

函数ID: `replaceAllNamePapi`

函数参数: `json形式的"待替换文本":"替换文本"`

参数示例: `{"玩家名":"%player_name%"}`

示例解析: `假设物品原先名为"玩家名的物品", 我的玩家ID是Neige, 替换后名称将变为"Neige的物品"`

> 信息
>
> 文本中的&将被自动替换为颜色符号 §

替换显示名(替换全部, 解析动作变量)

> 替换物品显示名中的对应文本(替换全部)

函数ID: `replaceAllNameSection`

函数参数: `json形式的"待替换文本":"替换文本"`

参数示例: `{"品质":"<strings::普通_精良>"}`

示例解析: `假设物品原先名为"品质 长剑", 替换后名称将随机变为"普通 长剑"或"精良 长剑"`

> 信息
>
> 文本中的&将被自动替换为颜色符号 §

替换显示名(使用正则, 替换一次)

> 替换物品显示名中的对应文本(只替换一次)

函数ID: `replaceNameRegex`

函数参数: `json形式的"正则表达式":"替换文本"`

参数示例: `{"\\d+":"不准写数字"}`

示例解析: `假设物品原先名为"114514", 替换后将变为"不准写数字"`

> 信息
>
> 文本中的&将被自动替换为颜色符号 § , $+索引表示组的调用
>
> $0代表匹配值全文, $1代表第一个组的返回值, 以此类推
>
> 你看不懂上面那行字说明你需要学习正则表达式: 跟着海螺学正则

替换显示名(使用正则, 替换一次, 解析papi变量)

> 替换物品显示名中的对应文本(只替换一次)

函数ID: `replaceNameRegexPapi`

函数参数: `json形式的"正则表达式":"替换文本"`

参数示例: `{"(强化等级: )(\\d+)":"$1%math_0:0_$2+1%"}`

示例解析: `假设物品原先名为"强化等级: 1", 替换后名称将变为"强化等级: 2"`

> 信息
>
> 文本中的&将被自动替换为颜色符号§, $+索引表示组的调用
>
> $0代表匹配值全文, $1代表第一个组的返回值, 以此类推
>
> 你看不懂上面那行字说明你需要学习正则表达式: 跟着海螺学正则

替换显示名(使用正则, 替换一次, 解析动作变量)

> 替换物品显示名中的对应文本(只替换一次)

函数ID: `replaceNameRegexSection`

函数参数: `json形式的"正则表达式":"替换文本"`

参数示例: `{"(强化等级: )(\\d+)":"$1<calculation::$2+1>"}`

示例解析: `假设物品原先名为"强化等级: 1", 替换后名称将变为"强化等级: 2"`

> 信息

> 文本中的&将被自动替换为颜色符号§, $+索引表示组的调用
>
> $0代表匹配值全文, $1代表第一个组的返回值, 以此类推
>
> 你看不懂上面那行字说明你需要学习正则表达式: 跟着海螺学正则

替换显示名(使用正则, 替换全部)

> 替换物品显示名中的对应文本(替换全部)

函数ID: `replaceAllNameRegex`

函数参数: `json形式的"正则表达式":"替换文本"`

参数示例: `{"\\d+":"不准写数字"}`

示例解析: `假设物品原先名为"114514", 替换后将变为"不准写数字"`

> 信息
>
> 文本中的&将被自动替换为颜色符号§, $+索引表示组的调用
>
> $0代表匹配值全文, $1代表第一个组的返回值, 以此类推
>
> 你看不懂上面那行字说明你需要学习正则表达式: 跟着海螺学正则

替换显示名(使用正则, 替换全部, 解析papi变量)

> 替换物品显示名中的对应文本(替换全部)

函数ID: `replaceAllNameRegexPapi`

函数参数: `json形式的"正则表达式":"替换文本"`

参数示例: `{"(强化等级: )(\\d+)":"$1%math_0:0_$2+1%"}`

示例解析: `假设物品原先名为"强化等级: 1", 替换后名称将变为"强化等级: 2"`

> 信息
>
> 文本中的&将被自动替换为颜色符号§, $+索引表示组的调用
>
> $0代表匹配值全文, $1代表第一个组的返回值, 以此类推
>
> 你看不懂上面那行字说明你需要学习正则表达式: 跟着海螺学正则

替换显示名(使用正则, 替换全部, 解析动作变量)

> 替换物品显示名中的对应文本(替换全部)

函数ID: `replaceAllNameRegexSection`

函数参数: `json形式的"正则表达式":"替换文本"`

参数示例: `{"(强化等级: )(\\d+)":"$1<calculation::$2+1>"}`

示例解析: `假设物品原先名为"强化等级: 1", 替换后名称将变为"强化等级: 2"`

> 信息

> 文本中的&将被自动替换为颜色符号§, $+索引表示组的调用
>
> $0代表匹配值全文, $1代表第一个组的返回值, 以此类推
>
> 你看不懂上面那行字说明你需要学习正则表达式: 跟着海螺学正则

添加Lore

> 为物品添加Lore

函数ID: `addLore`

函数参数: `待添加Lore`

参数示例: `描述1\n描述2`

示例解析: `原物品将被添加2行Lore：描述1、描述2`

> 信息
>
> 文本中的&将被自动替换为颜色符号§, \n代表换行

添加Lore(解析papi变量)

> 为物品添加Lore

函数ID: `addLorePapi`

函数参数: `待添加Lore`

参数示例: `拥有者: %player_name%`

示例解析: `我的玩家ID是Neige, 所以原物品将被添加1行Lore：拥有者: Neige`

> 信息
>
> 文本中的&将被自动替换为颜色符号§, \n代表换行

添加Lore(解析其中的动作变量)

> 为物品添加Lore

函数ID: `addLoreSection`

函数参数: `待添加Lore`

参数示例: `<strings::描述1_描述2>`

示例解析: `原物品将被添加1行Lore：描述1或描述2`

> 信息
>
> 文本中的&将被自动替换为颜色符号§, \n代表换行

设置Lore

> 为物品设置Lore, 原先的Lore将被移除

函数ID: `setLore`

函数参数: `待设置Lore`

参数示例: `描述1\n描述2`

示例解析: `原物品的Lore将被设置为：描述1、描述2`

> 信息
>
> 文本中的&将被自动替换为颜色符号§, \n代表换行

设置Lore(解析papi变量)

> 为物品设置Lore, 原先的Lore将被移除

函数ID: `setLorePapi`

函数参数: `待设置Lore`

参数示例: `拥有者: %player_name%`

示例解析: `我的玩家ID是Neige, 所以原物品的Lore将被设置为：拥有者: Neige`

> 信息
>
> 文本中的&将被自动替换为颜色符号§, \n代表换行

设置Lore(解析动作变量)

> 为物品设置Lore, 原先的Lore将被移除

函数ID: `setLoreSection`

函数参数: `待设置Lore`

参数示例: `<strings::描述1_描述2>`

示例解析: `原物品的Lore将被设置为：描述1或描述2`

> 信息
>
> 文本中的&将被自动替换为颜色符号§, \n代表换行

替换Lore(替换一次)

> 替换物品Lore中的对应文本(只替换一次)

函数ID: `replaceLore`

函数参数: `json形式的"待替换文本":"替换文本"`

参数示例: `{"红宝石槽":"已镶嵌 红宝石\n物理伤害: 100"}`

示例解析: `假设原先物品Lore为`

```
红宝石槽
红宝石槽
```

`替换后将变为`

```
已镶嵌 红宝石
物理伤害: 100
红宝石槽
```

> 信息
>
> 文本中的&将被自动替换为颜色符号§, \n代表换行

替换Lore(替换一次, 解析papi变量)

> 替换物品Lore中的对应文本(只替换一次)

函数ID: `replaceLorePapi`

函数参数: `json形式的"待替换文本":"替换文本"`

参数示例: `{"玩家名":"%player_name%"}`

示例解析: `假设原先物品Lore为`

```
XXXXXXX
拥有者: 玩家名
```

`我的ID是Neige, 替换后将变为`

```
XXXXXXX
拥有者: Neige
```

> **信息**
>
> 文本中的&将被自动替换为颜色符号§, \n代表换行

替换Lore(替换一次, 解析动作变量)

> 替换物品Lore中的对应文本(只替换一次)

函数ID: `replaceLoreSection`

函数参数: `json形式的"待替换文本":"替换文本"`

参数示例: `{"<品质>":"<strings::普通_精良>"}`

示例解析: `假设原先物品Lore为`

```
品质: <品质>
```

`替换后将随机变为`

```
品质: 普通
```

`或`

```
品质: 精良
```

> **信息**
>
> 文本中的&将被自动替换为颜色符号§, \n代表换行

替换Lore(替换全部)

> 替换物品Lore中的对应文本(替换全部)

函数ID: `replaceAllLore`

函数参数: `json形式的"待替换文本":"替换文本"`

参数示例: `{"红宝石槽":"已镶嵌 红宝石\n物理伤害: 100"}`

示例解析: `假设原先物品Lore为`

```
红宝石槽
红宝石槽
```

`替换后将变为`

```
已镶嵌 红宝石
物理伤害: 100
已镶嵌 红宝石
物理伤害: 100
```

> **信息**
>
> 文本中的&将被自动替换为颜色符号§, \n代表换行

替换Lore(替换全部, 解析papi变量)

> 替换物品Lore中的对应文本(替换全部)

函数ID: `replaceAllLorePapi`

函数参数: `json形式的"待替换文本":"替换文本"`

参数示例: `{"玩家名":"%player_name%"}`

示例解析: `假设原先物品Lore为`

```
XXXXXXX
拥有者: 玩家名
```

`我的ID是Neige, 替换后将变为`

```
XXXXXXX
拥有者: Neige
```

> **信息**
>
> 文本中的&将被自动替换为颜色符号§, \n代表换行

替换Lore(替换全部, 解析动作变量)

> 替换物品Lore中的对应文本(替换全部)

函数ID: `replaceAllLoreSection`

函数参数: `json形式的"待替换文本":"替换文本"`

参数示例: `{"<品质>":"<strings::普通_精良>"}`

示例解析: `假设原先物品Lore为`

```
品质: <品质>
```

`替换后将随机变为`

```
品质: 普通
```

`或`

品质: 精良

> 信息
>
> 文本中的&将被自动替换为颜色符号§, \n代表换行

替换Lore(使用正则, 只替换一次)

> 替换物品Lore中的对应文本(只替换一次)

函数ID: `replaceLoreRegex`

函数参数: `json形式的"正则表达式":"替换文本"`

参数示例: `{"(我是)(你叠)":"$2$1"}`

示例解析: `假设原先物品Lore为`

```
我是你叠
我是你叠
```

`替换后将变为`

```
你叠我是
我是你叠
```

> 信息
>
> 文本中的&将被自动替换为颜色符号§, \n代表换行, $+索引表示组的调用

> $0代表匹配值全文, $1代表第一个组的返回值, 以此类推
>
> 你看不懂上面那行字说明你需要学习正则表达式: 跟着海螺学正则

替换Lore(使用正则, 只替换一次, 解析papi变量)

> 替换物品Lore中的对应文本(只替换一次)

函数ID: `replaceLoreRegexPapi`

函数参数: `json形式的"正则表达式":"替换文本"`

参数示例: `{"(强化等级: )(\\d+)":"$1%math_0:0_$2+1%"}`

示例解析: `假设物品原先Lore为`

```
强化等级: 1
强化等级: 1
```

`替换后将变为`

```
强化等级: 2
强化等级: 1
```

> **信息**
>
> 文本中的&将被自动替换为颜色符号§, \n代表换行, $+索引表示组的调用

> $0代表匹配值全文, $1代表第一个组的返回值, 以此类推
>
> 你看不懂上面那行字说明你需要学习正则表达式: 跟着海螺学正则

替换Lore(使用正则, 只替换一次, 解析动作变量)

> 替换物品Lore中的对应文本(只替换一次)

函数ID: `replaceLoreRegexSection`

函数参数: `json形式的"正则表达式":"替换文本"`

参数示例: `{"(强化等级: )(\\d+)":"$1<calculation::$2+1>"}`

示例解析: `假设物品原先Lore为`

```
强化等级: 1
强化等级: 1
```

`替换后将变为`

```
强化等级: 2
强化等级: 1
```

> **信息**
>
> 文本中的&将被自动替换为颜色符号§, \n代表换行, $+索引表示组的调用

> $0代表匹配值全文, $1代表第一个组的返回值, 以此类推
>
> 你看不懂上面那行字说明你需要学习正则表达式: 跟着海螺学正则

替换Lore(使用正则, 替换全部)

> 替换物品Lore中的对应文本(替换全部)

函数ID: `replaceAllLoreRegex`

函数参数: `json形式的"正则表达式":"替换文本"`

参数示例: `{"(我是)(你叠)":"$2$1"}`

示例解析: `假设原先物品Lore为`

```
我是你叠
我是你叠
```

`替换后将变为`

```
你叠我是
你叠我是
```

> **信息**
>
> 文本中的&将被自动替换为颜色符号§, \n代表换行, $+索引表示组的调用
>
> $0代表匹配值全文, $1代表第一个组的返回值, 以此类推

> 你看不懂上面那行字说明你需要学习正则表达式: 跟着海螺学正则

替换Lore(使用正则, 替换全部, 解析papi变量)

> 替换物品Lore中的对应文本(替换全部)

函数ID: `replaceAllLoreRegexPapi`

函数参数: `json形式的"正则表达式":"替换文本"`

参数示例: `{"(强化等级: )(\\d+)":"$1%math_0:0_$2+1%"}`

示例解析: `假设物品原先Lore为`

```
强化等级: 1
强化等级: 1
```

`替换后将变为`

```
强化等级: 2
强化等级: 2
```

> **信息**
>
> 文本中的&将被自动替换为颜色符号§, \n代表换行, $+索引表示组的调用
>
> $0代表匹配值全文, $1代表第一个组的返回值, 以此类推
>
> 你看不懂上面那行字说明你需要学习正则表达式: 跟着海螺学正则

替换Lore(使用正则, 替换全部, 解析动作变量)

> 替换物品Lore中的对应文本(替换全部)

函数ID: `replaceAllLoreRegexSection`

函数参数: `json形式的"正则表达式":"替换文本"`

参数示例: `{"(强化等级: )(\\d+)":"$1<calculation::$2+1>"}`

示例解析: `假设物品原先Lore为`

```
强化等级: 1
强化等级: 1
```

`替换后将变为`

```
强化等级: 2
强化等级: 2
```

> **信息**
>
> 文本中的&将被自动替换为颜色符号§, \n代表换行, $+索引表示组的调用
>
> $0代表匹配值全文, $1代表第一个组的返回值, 以此类推
>
> 你看不懂上面那行字说明你需要学习正则表达式: 跟着海螺学正则

设置子ID/损伤值

> 为物品设置子ID/损伤值

函数ID: `setDamage`

函数参数: `待设置子ID/损伤值`

参数示例: `1`

示例解析: `假设原物品为石剑(满耐久131), 设置后耐久将变为130`

> ⚠ **警告**
>
> 对于有耐久的物品, 损伤值超过耐久上限将销毁物品
>
> 例如石剑的耐久上限为131, 设置损伤值为132将导致物品损坏(数量变为0)

设置子ID/损伤值(解析papi变量)

> 为物品设置子ID/损伤值

函数ID: `setDamagePapi`

函数参数: `待设置子ID/损伤值`

参数示例: `%player_level%`

示例解析: `假设原物品为石剑(满耐久131), 我的等级是10, 设置后耐久将变为121`

> ⚠ 警告
>
> 对于有耐久的物品, 损伤值超过耐久上限将销毁物品
>
> 例如石剑的耐久上限为131, 设置损伤值为132将导致物品损坏(数量变为0)

设置子ID/损伤值(解析动作变量)

> 为物品设置子ID/损伤值

函数ID: `setDamageSection`

函数参数: `待设置子ID/损伤值`

参数示例: `<number::1_10>`

示例解析: `假设原物品为石剑(满耐久131), 设置后耐久将随机变为121-130`

> ⚠ 警告
>
> 对于有耐久的物品, 损伤值超过耐久上限将销毁物品
>
> 例如石剑的耐久上限为131, 设置损伤值为132将导致物品损坏(数量变为0)

增加子ID/损伤值

> 为物品增加子ID/损伤值

函数ID: `addDamage`

函数参数: `待增加子ID/损伤值`

参数示例: `1`

示例解析: `假设原物品为石剑, 设置后耐久将减1(耐久为-1即销毁)`

> ⚠ 警告
>
> 对于有耐久的物品, 损伤值超过耐久上限将销毁物品
>
> 例如石剑的耐久上限为131, 当前耐久为130, 增加131点损伤值将导致物品损坏(数量变为0)

增加子ID/损伤值(解析papi变量)

> 为物品增加子ID/损伤值

函数ID: `addDamagePapi`

函数参数: `待增加子ID/损伤值`

参数示例: `%player_level%`

示例解析: `假设原物品为石剑, 我的等级是10, 设置后耐久将减10(耐久为-1即销毁)`

> ⚠ 警告
>
> 对于有耐久的物品, 损伤值超过耐久上限将销毁物品
>
> 例如石剑的耐久上限为131, 当前耐久为130, 增加131点损伤值将导致物品损坏(数量变为0)

增加子ID/损伤值(解析动作变量)

> 为物品增加子ID/损伤值

函数ID: `addDamageSection`

函数参数: `待增加子ID/损伤值`

参数示例: `<number::1_10>`

示例解析: `假设原物品为石剑, 设置后耐久将随机减去1到10(耐久为-1即销毁)`

> ⚠ **警告**
>
> 对于有耐久的物品, 损伤值超过耐久上限将销毁物品
>
> 例如石剑的耐久上限为131, 当前耐久为130, 增加131点损伤值将导致物品损坏(数量变为0)

减少子ID/损伤值

> 为物品减少子ID/损伤值

函数ID: `takeDamage`

函数参数: `待减少子ID/损伤值`

参数示例: `1`

示例解析: `假设原物品为石剑, 当前耐久为130, 设置后耐久将变为131(损伤值减少了1)`

> **警告**
>
> 对于有耐久的物品, 损伤值超过耐久上限将销毁物品
>
> 例如石剑的耐久上限为131, 当前耐久为130, 减少-131点损伤值将导致物品损坏(数量变为0)

减少子ID/损伤值(解析papi变量)

> 为物品减少子ID/损伤值

函数ID: `takeDamagePapi`

函数参数: `待减少子ID/损伤值`

参数示例: `%player_level%`

示例解析: `假设原物品为石剑, 当前耐久为120, 我的等级是10, 设置后耐久将变为130(损伤值减少了10)`

> **警告**
>
> 对于有耐久的物品, 损伤值超过耐久上限将销毁物品
>
> 例如石剑的耐久上限为131, 当前耐久为130, 减少-131点损伤值将导致物品损坏(数量变为0)

减少子ID/损伤值(解析动作变量)

> 为物品减少子ID/损伤值

函数ID: `takeDamageSection`

函数参数: `待减少子ID/损伤值`

参数示例: `<number::1_10>`

示例解析: `假设原物品为石剑, 当前耐久为120, 设置后耐久将随机恢复1到10`

> **警告**
>
> 对于有耐久的物品, 损伤值超过耐久上限将销毁物品
>
> 例如石剑的耐久上限为131, 当前耐久为130, 减少-131点损伤值将导致物品损坏(数量变为0)

CustomModelData

> 为物品设置CustomModelData

函数ID: `setCustomModelData`

函数参数: `待设置CustomModelData`

参数示例: `1`

示例解析: `假设原物品的CustomModelData将被设置为1`

> **信息**
>
> 适用于1.14+版本

CustomModelData(解析papi变量)

> 为物品设置CustomModelData

函数ID: `setCustomModelDataPapi`

函数参数: `待设置CustomModelData`

参数示例: `%player_level%`

示例解析: `假设我的等级是10, 设置后原物品的CustomModelData将被设置为1`

> 信息
>
> 适用于1.14+版本

CustomModelData(解析动作变量)

> 为物品设置CustomModelData

函数ID: `setCustomModelDataSection`

函数参数: `待设置CustomModelData`

参数示例: `<number::1_10>`

示例解析: `原物品的CustomModelData将随机变为1-10`

> 信息
>
> 适用于1.14+版本

无法破坏

> 为物品设置无法破坏

函数ID: `setUnbreakable`

函数参数: `true/false`

参数示例: `true`

示例解析: `待设置物品将变为无法破坏`

无法破坏(解析papi变量)

> 为物品设置无法破坏

函数ID: `setUnbreakablePapi`

函数参数: `true/false`

参数示例: `%player_name%`

示例解析: `假设玩家ID为true, 设置后物品将变为无法破坏`

无法破坏(解析动作变量)

> 为物品设置无法破坏

函数ID: `setUnbreakableSection`

函数参数: `true/false`

参数示例: `<strings::true_false>`

示例解析: `待设置物品将随机变为无法破坏/可破坏状态`

设置附魔

> 为物品设置附魔(移除原有附魔)

函数ID: `setEnchantment`

函数参数: `json形式的"附魔ID":附魔等级`

参数示例: `{"DAMAGE_ALL":10,"LOOT_BONUS_MOBS":10}`

示例解析: `物品附魔将变为锋利10、抢夺10`

> ⚠ 警告
>
> !> 物品原有附魔将被移除

设置附魔(解析papi变量)

> 为物品设置附魔(移除原有附魔)

函数ID: `setEnchantmentPapi`

函数参数: `json形式的"附魔ID":附魔等级`

参数示例: `{"%player_name%":10}`

示例解析: `假设你的ID是DAMAGE_ALL, 设置后物品附魔将变为锋利10`

> ⚠ 警告
>
> !> 物品原有附魔将被移除

设置附魔(解析动作变量)

> 为物品设置附魔(移除原有附魔)

函数ID: `setEnchantmentSection`

函数参数: `json形式的"附魔ID":附魔等级`

参数示例: `{"DAMAGE_ALL":<number::1_10>}`

示例解析: `物品附魔将随机变为锋利1-锋利10`

> ⚠ 警告
>
> !> 物品原有附魔将被移除

添加附魔

> 为物品添加附魔(原有相同附魔将被覆盖)

函数ID: `addEnchantment`

函数参数: `json形式的"附魔ID":附魔等级`

参数示例: `{"DAMAGE_ALL":10,"LOOT_BONUS_MOBS":10}`

示例解析: `物品将获得锋利10、抢夺10的附魔`

> ⚠ 警告
>
> 假设物品原先为锋利100, 我添加一个锋利10, 将导致锋利100变为锋利10

添加附魔(解析papi变量)

> 为物品添加附魔(原有相同附魔将被覆盖)

函数ID: `addEnchantmentPapi`

函数参数: `json形式的"附魔ID":附魔等级`

参数示例: `{"%player_name%":10}`

示例解析: `假设玩家ID为DAMAGE_ALL, 设置后物品将获得锋利10附魔`

> ⚠ 警告
>
> 假设物品原先为锋利100, 我添加一个锋利10, 将导致锋利100变为锋利10

添加附魔(解析动作变量)

> 为物品添加附魔(原有相同附魔将被覆盖)

函数ID: `addEnchantmentSection`

函数参数: `json形式的"附魔ID":附魔等级`

参数示例: `{"DAMAGE_ALL":<number::1_10>}`

示例解析: `物品将获得锋利1-锋利10的附魔`

> ⚠ 警告
>
> 假设物品原先为锋利100, 我添加一个锋利10, 将导致锋利100变为锋利10

添加附魔(不覆盖)

> 为物品添加附魔(原有相同附魔不覆盖)

函数ID: `addNotCoverEnchantment`

函数参数: `json形式的"附魔ID":附魔等级`

参数示例: `{"DAMAGE_ALL":10,"LOOT_BONUS_MOBS":10}`

示例解析: `物品将获得锋利10、抢夺10的附魔(如果原先物品没有锋利、抢夺附魔的话)`

> ⚠ 警告
>
> 假设物品原先为锋利100, 我添加一个锋利10, 物品仍为锋利100

添加附魔(不覆盖, 解析papi变量)

> 为物品添加附魔(原有相同附魔不覆盖)

函数ID: `addNotCoverEnchantmentPapi`

函数参数: `json形式的"附魔ID":附魔等级`

参数示例: `{"%player_name%":10}`

示例解析: `假设玩家ID为DAMAGE_ALL, 设置后物品将获得锋利10附魔(如果原先物品没有锋利附魔的话)`

> ⚠ 警告
>
> 假设物品原先为锋利100, 我添加一个锋利10, 物品仍为锋利100

添加附魔(不覆盖, 解析动作变量)

> 为物品添加附魔(原有相同附魔不覆盖)

函数ID: `addNotCoverEnchantmentSection`

函数参数: `json形式的"附魔ID":附魔等级`

参数示例: `{"DAMAGE_ALL":<number::1_10>}`

示例解析: `物品将获得锋利1-锋利10的附魔(如果原先物品没有锋利附魔的话)`

> ⚠ 警告
>
> 假设物品原先为锋利100, 我添加一个锋利10, 物品仍为锋利100

移除附魔

> 为物品移除附魔

函数ID: `removeEnchantment`

函数参数: `附魔ID, 以空格间隔`

参数示例: `DAMAGE_ALL LOOT_BONUS_MOBS`

示例解析: `将移除物品的锋利、抢夺附魔`

移除附魔(解析papi变量)

> 为物品移除附魔

函数ID: `removeEnchantmentPapi`

函数参数: `附魔ID, 以空格间隔`

参数示例: `%player_name%`

示例解析: `假设玩家ID为DAMAGE_ALL, 将移除物品的锋利附魔`

移除附魔(解析动作变量)

> 为物品移除附魔

函数ID: `removeEnchantmentSection`

函数参数: `附魔ID, 以空格间隔`

参数示例: `<strings::DAMAGE\_ALL_LOOT\_BONUS\_MOBS>`

示例解析: `将随机移除物品的锋利或抢夺附魔`

附魔升级

> 为物品附魔升级(目标等级小于0将移除附魔)

函数ID: `levelUpEnchantment`

函数参数: `json形式的"附魔ID":附魔等级`

参数示例: `{"DAMAGE_ALL":10,"LOOT_BONUS_MOBS":10}`

示例解析: `物品的锋利、抢夺附魔将提升10级`

附魔升级(解析papi变量)

> 为物品附魔升级(目标等级小于0将移除附魔)

函数ID: `levelUpEnchantmentPapi`

函数参数: `json形式的"附魔ID":附魔等级`

参数示例: `{"%player_name%":10}`

示例解析: `假设玩家ID为DAMAGE_ALL, 物品的锋利附魔将提升10级`

附魔升级(解析动作变量)

> 为物品附魔升级(目标等级小于0将移除附魔)

函数ID: `levelUpEnchantmentSection`

函数参数: `json形式的"附魔ID":附魔等级`

参数示例: `{"DAMAGE_ALL":<number::1_10>}`

示例解析: `物品的锋利附魔将随机提升1-10级`

附魔降级

> 为物品附魔降级(目标等级小于0将移除附魔)

函数ID: `levelDownEnchantment`

函数参数: `json形式的"附魔ID":附魔等级`

参数示例: `{"DAMAGE_ALL":10,"LOOT_BONUS_MOBS":10}`

示例解析: `物品的锋利、抢夺附魔将降低10级(目标等级小于0将移除附魔)`

附魔降级(解析papi变量)

> 为物品附魔降级(目标等级小于0将移除附魔)

函数ID: `levelDownEnchantmentPapi`

函数参数: `json形式的"附魔ID":附魔等级`

参数示例: `{"%player_name%":10}`

示例解析: `假设玩家ID为DAMAGE_ALL, 物品的锋利附魔将降低10级(目标等级小于0将移除附魔)`

附魔降级(解析动作变量)

> 为物品附魔降级(目标等级小于0将移除附魔)

函数ID: `levelDownEnchantmentSection`

函数参数: `json形式的"附魔ID":附魔等级`

参数示例: `{"DAMAGE_ALL":<number::1_10>}`

示例解析: `物品的锋利附魔将随机降低1-10级(目标等级小于0将移除附魔)`

设置属性隐藏

> 为物品设置属性隐藏(移除原有属性隐藏)

函数ID: `setItemFlag`

函数参数: `属性隐藏ID, 以空格间隔`

参数示例: `HIDE_ATTRIBUTES HIDE_DYE`

示例解析: `物品的属性、染料颜色将被隐藏`

> 信息
>
> 隐藏物品属性
>
> HIDE_ATTRIBUTES
>
> 隐藏物品可破坏方块
>
> HIDE_DESTROYS
>
> 隐藏物品染料颜色
>
> HIDE_DYE
>
> 隐藏物品附魔

> HIDE_ENCHANTS
>
> 隐藏物品可放置方块
>
> HIDE_PLACED_ON
>
> 隐藏物品药水效果
>
> HIDE_POTION_EFFECTS
>
> 隐藏物品无法破坏
>
> HIDE_UNBREAKABLE

设置属性隐藏(解析papi变量)

> 为物品设置属性隐藏(移除原有属性隐藏)

函数ID: `setItemFlagPapi`

函数参数: `属性隐藏ID, 以空格间隔`

参数示例: `%player_name%`

示例解析: `假设玩家ID为HIDE_DYE, 物品的染料颜色将被隐藏`

设置属性隐藏(解析动作变量)

> 为物品设置属性隐藏(移除原有属性隐藏)

函数ID: `setItemFlagSection`

函数参数: `属性隐藏ID, 以空格间隔`

参数示例: `<strings::HIDE\_ATTRIBUTES_HIDE\_DYE>`

示例解析: `物品的属性或染料颜色将被隐藏`

添加属性隐藏

> 为物品添加属性隐藏

函数ID: `addItemFlag`

函数参数: `属性隐藏ID, 以空格间隔`

参数示例: `HIDE_ATTRIBUTES HIDE_DYE`

示例解析: `物品的属性、染料颜色将被隐藏`

> 信息
>
> 隐藏物品属性
>
> HIDE_ATTRIBUTES
>
> 隐藏物品可破坏方块
>
> HIDE_DESTROYS
>
> 隐藏物品染料颜色
>
> HIDE_DYE
>
> 隐藏物品附魔

> HIDE_ENCHANTS
>
> 隐藏物品可放置方块
>
> HIDE_PLACED_ON
>
> 隐藏物品药水效果
>
> HIDE_POTION_EFFECTS
>
> 隐藏物品无法破坏
>
> HIDE_UNBREAKABLE

添加属性隐藏(解析papi变量)

> 为物品添加属性隐藏

函数ID: `addItemFlagPapi`

函数参数: `属性隐藏ID, 以空格间隔`

参数示例: `%player_name%`

示例解析: `假设玩家ID为HIDE_DYE, 物品的染料颜色将被隐藏`

添加属性隐藏(解析动作变量)

> 为物品添加属性隐藏

函数ID: `addItemFlagSection`

函数参数: `属性隐藏ID, 以空格间隔`

参数示例: `<strings::HIDE\_ATTRIBUTES_HIDE\_DYE>`

示例解析: `物品的属性或染料颜色将被隐藏`

移除属性隐藏

> 为物品移除属性隐藏

函数ID: `removeItemFlag`

函数参数: `属性隐藏ID, 以空格间隔`

参数示例: `HIDE_ATTRIBUTES HIDE_DYE`

示例解析: `物品的属性、染料颜色将显示出来`

> 信息
>
> 隐藏物品属性
>
> HIDE_ATTRIBUTES
>
> 隐藏物品可破坏方块
>
> HIDE_DESTROYS
>
> 隐藏物品染料颜色
>
> HIDE_DYE
>
> 隐藏物品附魔

> HIDE_ENCHANTS
>
> 隐藏物品可放置方块
>
> HIDE_PLACED_ON
>
> 隐藏物品药水效果
>
> HIDE_POTION_EFFECTS
>
> 隐藏物品无法破坏
>
> HIDE_UNBREAKABLE

移除属性隐藏(解析papi变量)

> 为物品移除属性隐藏

函数ID: `removeItemFlagPapi`

函数参数: `属性隐藏ID, 以空格间隔`

参数示例: `%player_name%`

示例解析: `假设玩家ID为HIDE_DYE, 物品的染料颜色将显示出来`

移除属性隐藏(解析动作变量)

> 为物品移除属性隐藏

函数ID: `removeItemFlagSection`

函数参数: `属性隐藏ID, 以空格间隔`

参数示例: `<strings::HIDE\_ATTRIBUTES_HIDE\_DYE>`

示例解析: `物品的属性或染料颜色将显示出来`

设置NBT(无法设置列表)

> 为物品设置NBT(原有设置NBT将被覆盖)

函数ID: `setNBT`

函数参数: `json形式的"NBT键":"NBT值"`

参数示例: `{"test1":"test1","test2.test3":"test3","test4":"(Double) 100"}`

示例解析: `别jb解析了, 直接看图`

> **信息**
>
> 通过.分隔NBTCompound, 数字需要通过前缀指定类型
>
> Byte 类型的 1: (Byte) 1
>
> Short 类型的 1: (Short) 1
>
> Int 类型的 1: (Int) 1

> Long 类型的 1: (Long) 1
>
> Float 类型的 1: (Float) 1
>
> Double 类型的 1: (Double) 1
>
> ByteArray: [(Byte) 1,(Byte) 2,(Byte) 3,(Byte) 4]
>
> IntArray: [(Int) 1,(Int) 2,(Int) 3,(Int) 4]
>
> 不要忘记括号后面的空格, (Double) 1生效, (Double)1不生效!

设置NBT(无法设置列表, 解析papi变量)

> 为物品设置NBT(原有设置NBT将被覆盖)

函数ID: `setNBTPapi`

函数参数: `json形式的"NBT键":"NBT值"`

参数示例: `{"myName":"%player_name%"}`

示例解析: `我的玩家ID为Neige, 看图`

设置NBT(无法设置列表, 解析动作变量)

> 为物品设置NBT(原有设置NBT将被覆盖)

函数ID: `setNBTSection`

函数参数: `json形式的"NBT键":"NBT值"`

参数示例: `{"test":"<strings::test1_test2>"}`

示例解析: `test的值随机为test1或test2, 看图`

设置NBT

> 为物品设置NBT(原有设置NBT将被覆盖)

函数ID: `setNBTWithList`

函数参数: `json形式的"NBT键":"NBT值"`

参数示例: `{"test.0.test":"test2","test.1.test":"test3"}`

示例解析:

`原有NBT:`

`设置后NBT:`

> 信息
>
> 通过.分隔NBTCompound、List、ByteArray与IntArray, List、ByteArray与IntArray中的NBT键即为相应索引, 以索引(数字)代替, 数字需要通过前缀指定类型
>
> Byte 类型的 1: (Byte) 1
>
> Short 类型的 1: (Short) 1
>
> Int 类型的 1: (Int) 1
>
> Long 类型的 1: (Long) 1
>
> Float 类型的 1: (Float) 1
>
> Double 类型的 1: (Double) 1
>
> ByteArray: [(Byte) 1,(Byte) 2,(Byte) 3,(Byte) 4]
>
> IntArray: [(Int) 1,(Int) 2,(Int) 3,(Int) 4]
>
> 不要忘记括号后面的空格, (Double) 1生效, (Double)1不生效!

设置NBT(解析papi变量)

> 为物品设置NBT(原有设置NBT将被覆盖)

函数ID: `setNBTWithListPapi`

函数参数: `json形式的"NBT键":"NBT值"`

参数示例: `{"myName":"%player_name%"}`

示例解析: `我的玩家ID为Neige, 看图`

设置NBT(解析动作变量)

> 为物品设置NBT(原有设置NBT将被覆盖)

函数ID: `setNBTWithListSection`

函数参数: `json形式的"NBT键":"NBT值"`

参数示例: `{"test":"<strings::test1_test2>"}`

示例解析: `test的值随机为test1或test2, 看图`

重构物品

> 重构NI物品

函数ID: `rebuild`

函数参数: `json形式的"节点名":"节点值"`

参数示例: `{"test1":"测试测试","test2":null}`

如示例所写, 你可以重新设置节点的值, 也可以通过将节点值设置为null让对应节点刷新, 比如:

```yaml
test:
  material: STONE
  name: <test2>
  sections:
    test1: <number::0_100>
    test2: <fastcalc::<test1>+1>
```

传入参数`{"test2":"测试测试"}`，物品名将变为`测试测试`

传入参数`{"test1":"100"}`，物品名不变

传入参数`{"test1":"100","test2":null}`，物品名将变为`101`

> **警告**
>
> 该函数将无视堆叠数量重构物品

重构物品(解析papi变量)

> 重构NI物品

函数ID: `rebuildPapi`

函数参数: `json形式的"节点名":"节点值"`

参数示例: `{"test1":"测试测试","test2":null}`

> **警告**
>
> 该函数将无视堆叠数量重构物品

重构物品(解析动作变量)

> 重构NI物品

函数ID: `rebuildSection`

函数参数: `json形式的"节点名":"节点值"`

参数示例: `{"test1":"测试测试","test2":null}`

> ⚠ **警告**
>
> 该函数将无视堆叠数量重构物品

重构物品(只刷新部分物品)

> 重构NI物品

函数ID: `rebuildAmount`

函数参数: `刷新数量 json形式的"节点名":"节点值"`

参数示例: `3 {"test1":"测试测试","test2":null}`

由于传入了参数`3`, 所以最多只刷新3个物品, 如果物品当前堆叠数量大于3, 将仅刷新3个物品, 并将剩余物品返还背包

重构物品(只刷新部分物品)(解析papi变量)

> 重构NI物品

函数ID: `rebuildAmountPapi`

函数参数: `刷新数量 json形式的"节点名":"节点值"`

参数示例: `3 {"test1":"测试测试","test2":null}`

由于传入了参数`3`, 所以最多只刷新3个物品, 如果物品当前堆叠数量大于3, 将仅刷新3个物品, 并将剩余物品返还背包

重构物品(只刷新部分物品)(解析动作变量)

> 重构NI物品

函数ID: `rebuildAmountSection`

函数参数: `刷新数量 json形式的"节点名":"节点值"`

参数示例: `3 {"test1":"测试测试","test2":null}`

由于传入了参数`3`, 所以最多只刷新3个物品, 如果物品当前堆叠数量大于3, 将仅刷新3个物品, 并将剩余物品返还背包

刷新部分节点

> 重构NI物品

函数ID: `refresh`

函数参数: `待刷新节点名`, 多个节点名以空格作间隔, 如果节点名中包含空格, 请在空格前加\转义符

参数示例: `test1 test2`

比如:

```yaml
test:
  material: STONE
  name: <test2>
  sections:
    test1: <number::0_100>
    test2: <fastcalc::<test1>+1>
```

传入参数`test1 test2`, 物品名将再次随机生成

传入参数`test1`, 物品名不变

传入参数`test2`, 物品名不变

> **警告**
>
> 该函数将无视堆叠数量重构物品

刷新部分节点(解析papi变量)

> 重构NI物品

函数ID: `refreshPapi`

函数参数: `待刷新节点名`, 多个节点名以空格作间隔, 如果节点名中包含空格, 请在空格前加\转义符

参数示例: `test1 test2`

> **警告**

> 该函数将无视堆叠数量重构物品

刷新部分节点(解析动作变量)

> 刷新部分节点

函数ID: `refreshSection`

函数参数: `待刷新节点名`, 多个节点名以空格作间隔, 如果节点名中包含空格, 请在空格前加\转义符

参数示例: `test1 test2`

> ⚠ **警告**
>
> 该函数将无视堆叠数量重构物品

刷新部分节点(只刷新部分物品)

> 重构NI物品

函数ID: `refreshAmount`

函数参数: `刷新数量 待刷新节点名`, 多个节点名以空格作间隔, 如果节点名中包含空格, 请在空格前加\转义符

参数示例: `3 test1 test2`

由于传入了参数`3`, 所以最多只刷新3个物品, 如果物品当前堆叠数量大于3, 将仅刷新3个物品, 并将剩余物品返还背包

刷新部分节点(只刷新部分物品)(解析papi变量)

> 重构NI物品

函数ID: `refreshAmountPapi`

函数参数: `刷新数量 待刷新节点名`, 多个节点名以空格作间隔, 如果节点名中包含空格, 请在空格前加\转义符

参数示例: `3 test1 test2`

由于传入了参数`3`, 所以最多只刷新3个物品, 如果物品当前堆叠数量大于3, 将仅刷新3个物品, 并将剩余物品返还背包

刷新部分节点(只刷新部分物品)(解析动作变量)

> 刷新部分节点

函数ID: `refreshAmountSection`

函数参数: `刷新数量 待刷新节点名`, 多个节点名以空格作间隔, 如果节点名中包含空格, 请在空格前加\转义符

参数示例: `3 test1 test2`

由于传入了参数`3`, 所以最多只刷新3个物品, 如果物品当前堆叠数量大于3, 将仅刷新3个物品, 并将剩余物品返还背包

设置使用次数

> 为物品设置使用次数(应用于物品动作)

函数ID:

`setCharge`

`setChargePapi` (解析papi变量)

`setChargeSection`(解析动作变量)

函数参数: `待设置使用次数`

参数示例: `1`

示例解析: `设置后使用次数将变为1 (最大不超过使用次数上限, 最小为0)`

增加使用次数

> 为物品增加使用次数(应用于物品动作)

函数ID:

`addCharge`

`addChargePapi` (解析papi变量)

`addChargeSection`(解析动作变量)

函数参数: `待增加使用次数`

参数示例: `1`

示例解析: `设置后使用次数将增加1 (最大不超过使用次数上限, 最小为0)`

减少使用次数

> 为物品减少使用次数(应用于物品动作)

函数ID:

`takeCharge`

`takeChargePapi` (解析papi变量)

`takeChargeSection`(解析动作变量)

函数参数: `待减少使用次数`

参数示例: `1`

示例解析: `设置后使用次数将减少1 (最大不超过使用次数上限, 最小为0)`

设置最大使用次数

> 为物品设置最大使用次数(应用于物品动作)

函数ID:

`setMaxCharge`

`setMaxChargePapi` (解析papi变量)

`setMaxChargeSection`(解析动作变量)

函数参数: `待设置最大使用次数`

参数示例: `1`

示例解析: `设置后最大使用次数将变为1 (最小为1, 使用次数将同时被设置为不超过最大使用次数)`

增加最大使用次数

> 为物品增加最大使用次数(应用于物品动作)

函数ID:

`addMaxCharge`

`addMaxChargePapi` (解析papi变量)

`addMaxChargeSection`(解析动作变量)

函数参数: `待增加最大使用次数`

参数示例: `1`

示例解析: `设置后最大使用次数将增加1 (最小为1, 使用次数将同时被设置为不超过最大使用次数)`

减少最大使用次数

> 为物品减少最大使用次数(应用于物品动作)

函数ID:

`takeMaxCharge`

`takeMaxChargePapi` (解析papi变量)

`takeMaxChargeSection`(解析动作变量)

函数参数: `待减少最大使用次数`

参数示例: `1`

示例解析: `设置后最大使用次数将减少1 (最小为1, 使用次数将同时被设置为不超过最大使用次数)`

设置自定义耐久度

> 为物品设置自定义耐久度(原版耐久度同时按比例发生变化)

函数ID:

`setCustomDurability`

`setCustomDurabilityPapi` (解析papi变量)

`setCustomDurabilitySection`(解析动作变量)

函数参数: `待设置自定义耐久度`

参数示例: `1`

示例解析: `设置后自定义耐久度将变为1 (最大不超过自定义耐久度上限, 最小为0)`

增加自定义耐久度

> 为物品增加自定义耐久度(原版耐久度同时按比例发生变化)

函数ID:

`addCustomDurability`

`addCustomDurabilityPapi` (解析papi变量)

`addCustomDurabilitySection`(解析动作变量)

函数参数: `待增加自定义耐久度`

参数示例: `1`

示例解析: `设置后自定义耐久度将增加1 (最大不超过自定义耐久度上限, 最小为0)`

减少自定义耐久度

> 为物品减少自定义耐久度(原版耐久度同时按比例发生变化)

函数ID:

`takeCustomDurability`

`takeCustomDurabilityPapi` (解析papi变量)

`takeCustomDurabilitySection`(解析动作变量)

函数参数: `待减少自定义耐久度`

参数示例: `1`

示例解析: `设置后自定义耐久度将减少1 (最大不超过自定义耐久度上限, 最小为0)`

设置最大自定义耐久度

> 为物品设置最大自定义耐久度

函数ID:

`setMaxCustomDurability`

`setMaxCustomDurabilityPapi` (解析papi变量)

`setMaxCustomDurabilitySection`(解析动作变量)

函数参数: `待设置最大自定义耐久度`

参数示例: `1`

示例解析: `设置后最大自定义耐久度将被变为1 (最小为1, 自定义耐久度将同时被设置为不超过最大自定义耐久度)`

增加最大自定义耐久度

> 为物品增加最大自定义耐久度

函数ID:

`addMaxCustomDurability`

`addMaxCustomDurabilityPapi` (解析papi变量)

`addMaxCustomDurabilitySection`(解析动作变量)

函数参数: `待增加最大自定义耐久度`

参数示例: `1`

示例解析: `设置后最大自定义耐久度将增加1 (最小为1, 自定义耐久度将同时被设置为不超过最大自定义耐久度)`

减少最大自定义耐久度

> 为物品减少最大自定义耐久度

函数ID:

`takeMaxCustomDurability`

`takeMaxCustomDurabilityPapi` (解析papi变量)

`takeMaxCustomDurabilitySection`(解析动作变量)

函数参数: `待减少最大自定义耐久度`

参数示例: `1`

示例解析: `设置后最大自定义耐久度将减少1 (最小为1, 自定义耐久度将同时被设置为不超过最大自定义耐久度)`

自定义函数

自定义函数需要一定的 javascript 和 java 基础。

自定义函数文件存放于`NeigeItems/CustomItemEditors`文件夹

下面是示例配置

```
// 文件名不重要, 写成啥都行
// main函数会自动执行
function main() {
  // 导入相应的类, 这两行看不懂的话直接抄就行
  const ItemEditorManager =
Packages.pers.neige.neigeitems.manager.ItemEditorManager.INSTANCE

  // 这是我写这段代码用到的类, 不是每次添加自定义物品编辑函数都要用到
  const ArrayList = Packages.java.util.ArrayList
  const ChatColor = Packages.org.bukkit.ChatColor
  const Material = Packages.org.bukkit.Material

  // 添加自定义物品编辑函数
  // 这里我添加了一个名为"test"的物品编辑函数, 但实际上它的功能与addLore函数相同
  ItemEditorManager.addItemEditor(
    // 函数名
    "test",
    /**
     * 物品编辑函数
     * @param player Player 物品拥有者
     * @param itemStack ItemStack 待编辑物品
     * @param content String 传入的文本
     */
    function(player, itemStack, content) {
      // 判断是不是空气
```

物品变量

简介

你可以在物品的名称/Lore中添加某些占位符

这些占位符将根据当前物品的nbt被发包替换

该功能仅对于生存模式的玩家生效

变量列表

- `%neigeitems_charge%` 物品当前剩余使用次数
- `%neigeitems_maxCharge%` 物品最大使用次数
- `%neigeitems_durability%` 物品当前剩余自定义耐久
- `%neigeitems_maxDurability%` 物品最大自定义耐久
- `%neigeitems_itembreak_不损坏时返回的文本_损坏时返回的文本%` 物品自定义耐久耗尽后是否损坏
- `%neigeitems_nbt_XXXXX%` 物品对应NBT的值

  例：`%neigeitems_nbt_NeigeItems.id%`
- `%neigeitems_nbtnumber_保留小数位数_XXXXXX%` 物品对应NBT的值(进行取整)

例：`%neigeitems_nbtnumber_0_NeigeItems.hashCode%`

物品包

路径

所有物品包配置文件应存放于`plugins/NeigeItems/ItemPacks`文件夹

重复 ID 的物品包仍然会被加载，但可能互相覆盖

最后哪个物品包活下来。。。随缘了属于是

格式

```
物品包ID:
 # 类似物品lore, 物品包的Items可以通过换行符"\n"换行
 Items:
 - 物品ID 随机最低数量-随机最高数量 生成概率 是否重复随机 指向数据
 MinItems: 最小掉落条目
 MaxItems: 最大掉落条目
 FancyDrop:
  offset:
   x: 横向偏移
   y: 纵向偏移
  angle:
   type: 旋转方式
 globalsections:
 - 引用的全局节点ID或者引用的全局节点文件路径
 sections:
 在此处声明私有节点, 就像物品配置一样
```

`物品ID` 可以是NI物品ID或者MM物品ID、EasyItem物品ID，优先检测NI物品

`随机最低数量-随机最高数量` 也可以直接写数量, 比如你可以写1-2, 代表掉落1-2个, 也可以写1, 代表掉落1个

`生成概率` 不写的话默认为1

`是否重复随机` 默认重复随机(对于MM物品、EasyItem物品, 这个配置项不代表是否随机生成, 代表物品是否合并)

`指向数据` 想写的话正常写就行

`最小掉落条目` 比如Items一共有5行, MinItems设置为1, 那么最少有1条Items生成, 物品的数量取决于Items中配置的数量, 与MinItems无关

`最大掉落条目` 比如Items一共有5行, MinItems设置为3, 那么最多有3条Items生成, 物品的数量取决于Items中配置的数量, 与MaxItems无关

`横向偏移` 表示物品向四周弹射的力度, 是一个数字

`纵向偏移` 表示物品向空中弹射的力度, 是一个数字

`旋转方式` 决定物品的弹射角度，是一个个绕一圈弹出去，还是随机弹出去(分为random和round)

同时可以像物品配置一样引用全局节点、声明私有节点、调用私有节点

但只有Items会应用节点配置, FancyDrop不会

以默认配置为例

```
Example1:
  Items:
```

具体调用指令如下

givePack

dropPack

自定义耐久

简介

你可以通过额外选项更改物品的自定义耐久, 当前版本下自定义耐久具有以下特性:

- 仅适用于拥有原版耐久的物品, 以及火焰弹
- 火焰弹仅在右键点燃TNT时消耗自定义耐久
- 创造模式的玩家使用物品不消耗自定义耐久
- 对于拥有原版耐久的物品, 自定义耐久的消耗规则与原版一致, 支持耐久附魔及经验修补
- 对于火焰弹, 自定义耐久支持耐久附魔但不支持经验修补
- 具有无法破坏标签的物品不消耗自定义耐久
- 物品的原版耐久条将随着自定义耐久的变化按比例变化, 比如自定义耐久`50/100`, 原版耐久`66/131`
- 将额外选项中的`options.itembreak`配置为`false`可以使物品自定义耐久耗尽时不消失, 这种耐久耗尽的物品将无法进行方块交互、无法进行实体交互、无法拉弓射箭、无法吃/饮用, 但耐久耗尽的盔甲不会自动从身上或某些插件的槽位中脱下来

随机节点

私有/全局节点

节点配置内全面支持节点调用/PAPI调用

即时声明节点

节点配置内全面支持节点调用/PAPI调用

自定义节点

自定义节点需要一定的 javascript 和 java 基础。

RGB颜色

如上所示

📄 节点调用

节点可以在任意位置通过的形式调用

📄 高级应用

直接展示例子:

私有/全局节点

> 节点配置内全面支持节点调用/PAPI调用

全局节点路径

所有全局节点配置文件应存放于`plugins/NeigeItems/GlobalSections`文件夹

重复 ID 的节点仍然会被加载，但可能互相覆盖

最后哪个节点活下来。。。随缘了属于是

私有节点配置

查看：私有节点配置，形如

```
随机名称的铁剑:
  material: IRON_SWORD
  name: <weight-1>
  sections:
    weight-1:
      type: weight
      values:
      - 5::名字1
      - 4::名字2
      - 3::名字3
      - 2::名字4
      - 1::名字5
```

字符串节点

```yaml
节点ID:
  type: strings
  values:
  - test1
  - test2
```

结果将在values中随机获取一个值

每个值被选中的几率相等

随机数节点

```yaml
节点ID:
  type: number
  min: 1
  max: 2
  fixed: 3
```

- `min` 随机数的最小值
- `max` 随机数的最大值
- `fixed` 小数保留位数

Gaussian节点

```yaml
节点ID:
```

简介: 类似MMOItems的, 符合正态分布的随机数, 随机数大概率在base附近, 小概率出现极大或极小的数值

- `base` 基础数值
- `spread` 浮动单位
- `maxSpread` 浮动范围上限
- `fixed` 小数保留位数 (默认为1)
- `min` 随机数的最小值
- `max` 随机数的最大值

详细介绍:

- `base`是基础数值, 随机数将以其为中心, 随机散布

- `spread`是浮动单位, 决定了随机数散步的幅度

  比如base设置为100, spread设置为0.1, 根据正态分布:

  使用该节点生成大量随机数

  68.27%的随机数介于90-110

  95.45%的随机数介于80-120

  99.74%的随机数介于70-130

  以此类推.....

- `maxSpread`是浮动范围上限, 限制了随机数的浮动极限, 防止出现过于离谱的数字

  比如我将maxSpread设置为0.3, 根据正态分布:

  0.26%的随机数将小于70或大于130, 即超过了0.3的幅度, 那么经过maxSpread的限制:

小于70的随机数将变为70, 而大于130的随机数将变为130

- `fixed`是取整位数, 默认为1

  比如随机数值为123.456, fixed设置为1, 那么你将得到123.4

  比如随机数值为123.456, fixed设置为0, 那么你将得到123

- `min`是随机数的最小值, `max`是随机数的最大值, 超过范围的随机数将被限制

  比如随机数值为123, min设置为200, 那么你将得到200

  比如随机数值为123, max设置为100, 那么你将得到100

公式节点

```
节点ID:
  type: calculation
  formula: 1+2+3<global-number-1>
  min: 1
  max: 100
  fixed: 3
```

- `formula` 待计算公式，支持代入节点及PAPI变量
- `min` 结果的最小值(默认不限制最小值)
- `max` 结果的最大值(默认不限制最大值)
- `fixed` 小数保留位数(默认为0, 即不保留小数)

> 信息

> 公式节点的本质是运行一段javascript代码
>
> 没有特殊需求应该优先使用快速计算(fastcalc)节点

快速计算节点

```
节点ID:
  type: fastcalc
  formula: 1+2+3<global-number-1>
  min: 1
  max: 100
  fixed: 3
```

- `formula` 待计算公式，支持代入节点及PAPI变量
- `min` 结果的最小值(默认不限制最小值)
- `max` 结果的最大值(默认不限制最大值)
- `fixed` 小数保留位数(默认为0, 即不保留小数)

> 信息
>
> 快速计算节点的本质是解析数学符号进行分析计算
>
> 计算速度高于公式节点

权重节点

```
节点ID:
```

`values`的格式为 权重::文本

结果将在`values`中根据权重随机获取一个值

例如，在该示例节点中

将有5/6的几率返回"第一行"，1/6的几率返回"第二行"

权重声明节点

```yaml
WeightDeclareTest:
  material: STONE
  lore:
    - '第1个值: <test>'
    - '第1个值: <test.0>'
    - '第2个值: <test.1>'
    - '第3个值: <test.2>'
    - '列表长度: <test.length>'
    - '第1个未选中值: <test.else.0>'
    - '第2个未选中值: <test.else.1>'
    - '未选中值长度: <test.else.length>'
  sections:
    test:
      type: weightdeclare
      key: test
      list:
        - '1::暴击率'
        - '1::暴击伤害'
        - '2::攻击力'
        - '2::攻击百分比'
        - '3::防御力'
        - '3::防御百分比'
        - '4::生命值'
        - '4::生命百分比'
```

- `list` 的格式为 权重::文本
- `key` 声明节点的键
- `amount` 声明数量
- `shuffled` 是否打乱列表顺序(默认为false)
- `putelse` 是否记录未选中的值(默认为false)

如上所示, weightdeclare节点可以根据权重和key声明一些节点

同时如果开启了putelse, 你可以通过`key.else.序号`访问未选中的值, 通过`key.else.length`获取未选中值的长度

需要注意的是, 节点必须被调用过才能生效, 比如:

```yaml
test1:
  material: STONE
  lore:
    - '<test>'
    - '<test.0>'
  sections:
    test:
      type: weightdeclare
      key: test
      list:
        - '1::暴击率'
        - '1::暴击伤害'
        - '2::攻击力'
      amount: 2
      shuffled: true
test2:
  material: STONE
  lore:
    - '<test.0>'
  sections:
    test:
```

以上两个物品, test1的Lore中将正常显示第一个数值两次

test2则将显示`<test.0>`, 因为test节点未经调用

对此, 可以通过以下方式弥补:

```yaml
test2-改:
  material: STONE
  test: <test>
  lore:
    - '<test.0>'
  sections:
    test:
      type: weightdeclare
      key: test
      list:
        - '1::暴击率'
        - '1::暴击伤害'
        - '2::攻击力'
      amount: 2
      shuffled: true
```

我随便写了一个`test`配置项, 在里面调用了`test`节点, 这个配置项不会对物品产生实际影响

你可以把`test`随便改成别的东西, 比如`ghbvuvvg: <test>`

同时, 对于传入指向数据的情况, 我做了相关优化

比如:

```yaml
test1:
  material: STONE
  test: <test>
  lore:
```

`test.0`和`test.1`都可能返回三个值中的一个, 同时他们二者是不重复的

而假设你通过指向数据指定`test.0`为`暴击率`

经过我的优化处理, `test.1`只可能出现`暴击伤害`或`攻击力`, 不会与`test.0`重复

> ⚠ **警告**
>
> 权重不要小于千分之一, 精度问题会导致计算失准

JavaScript节点

```
节点ID:
 type: js
 path: ExampleScript.js::main
 # (可选)
 # args:
 # - 参数1
 # - 参数2
```

`path`的格式为 脚本路径::调用函数

`args`项可选，`args`的所有内容将作为参数传入被调用函数

例如，在该示例节点中

将调用`plugins/NeigeItems/Scripts/ExampleScript.js`脚本文件中的main函数

并返回main函数的返回值

Join节点

```yaml
节点ID:
  type: join
  list:
    - 第一行
    - 第二行
    - 第三行
    - 第四行
  separator: "-"
  prefix: '<'
  postfix: '>'
  limit: 3
  truncated: "..."
  transform: |-
    return this.it + "哈哈"
  shuffled: false
```

简介: 将list中的多段文本连接成一段文本

- `list` 待操作的列表
- `separator` 分隔符 (默认为", ")
- `prefix` 前缀 (默认无前缀)
- `postfix` 后缀 (默认无后缀)
- `limit` 限制列表长度
- `truncated` 超过长度的部分用该符号代替 (默认直接吞掉超过长度的部分)
- `transform` 对列表的每一行进行一些操作 (使用javascript函数)
- `shuffled` 是否打乱列表顺序

示例中的节点将返回:

```
<第一行哈哈-第二行哈哈-第三行哈哈-...>
```

由于该节点功能较其他节点更加复杂, 因此我为它编写了多个示例配置帮助理解, 如下:

```yaml
# 帮助理解list
JoinTest1:
  material: STONE
  lore:
    # 结果: 1, 2, 3, 4, 5
    - 'join节点: <test>'
  sections:
    test:
      type: join
      # 待操作的列表
      list:
        - 1
        - 2
        - 3
        - 4
        - 5
# 帮助理解separator
JoinTest2:
  material: STONE
  lore:
    # 结果: 1-2-3-4-5
    - 'join节点: <test>'
  sections:
    test:
      type: join
      list:
        - 1
        - 2
        - 3
        - 4
        - 5
```

WeightJoin节点

```yaml
WeightJoinTest:
  material: STONE
  lore:
  - '<test>'
  sections:
    test:
      type: weightjoin
      list:
      # 权重::取值
      - '1::暴击率: 10%'
      - '1::暴击伤害: 10%'
      - '2::攻击力: 100'
      - '2::攻击百分比: 10%'
      - '3::防御力: 100'
      - '3::防御百分比: 10%'
      - '4::生命值: 100'
      - '4::生命百分比: 10%'
      # 选取3条
      amount: 3
      # 是否打乱顺序
      shuffled: true
      # 像下面这样写分隔符、前缀和后缀
      # 即可达到调用多行lore的效果
      separator: "\\n"
      prefix: '"'
      postfix: '"'
```

简介: 根据权重选取list中的多段文本连接成一段文本

- `list` 待操作的列表
- `separator` 分隔符 (默认为", ")

- `prefix` 前缀 (默认无前缀)
- `postfix` 后缀 (默认无后缀)
- `amount` 选取数量
- `transform` 对列表的每一行进行一些操作 (使用javascript函数)
- `shuffled` 是否打乱列表顺序

示例中的物品将根据权重返回相应的词条

> ⚠ 警告
>
> 权重不要小于千分之一, 精度问题会导致计算失准

Repeat节点

```yaml
节点ID:
  type: repeat
  content: '待重复文本'
  separator: "-"
  prefix: '<'
  postfix: '>'
  repeat: 3
  transform: |-
    return this.it + "哈哈"
```

简介: 将content的文本重复多次, 生成一整段文本

- `content` 待重复文本
- `separator` 分隔符 (默认无分隔符)
- `prefix` 前缀 (默认无前缀)
- `postfix` 后缀 (默认无后缀)

- `repeat` 重复次数
- `transform` 每次重复前对文本进行一些操作 (使用javascript函数)

示例中的节点将返回:

```
<待重复文本哈哈-待重复文本哈哈-待重复文本哈哈>
```

由于该节点功能较其他节点更加复杂, 因此我为它编写了多个示例配置帮助理解, 如下:

```yaml
# 不使用js的操作形式
RepeatTest1:
 material: STONE
 lore:
  # 结果: 形似&4||||||||||||&f||||||, &f出现的位置随机
  - 'repeat节点: &4<repeat1>&f<repeat2>'
 sections:
  repeat1:
   type: repeat
   content: "|"
   repeat: <number>
  repeat2:
   type: repeat
   content: "|"
   repeat: <calculation::20-<number>>
  number:
   type: number
   min: 0
   max: 20
   fixed: 0
# 使用js的操作形式
RepeatTest2:
 material: STONE
 lore:
  # 结果: 形似&4||||||||||||&f||||||, &f出现的位置随机
```

渐变色节点

```yaml
节点ID:
  type: gradient
  colorStart: "000000"
  colorEnd: "FFFFFF"
  step: 1
  text: 哈哈哈哈哈哈哈哈哈哈哈哈哈哈哈哈
```

- `colorStart` 起始颜色
- `colorEnd` 结尾颜色
- `step` 每几个字符变一次颜色(默认为1, 可省略)
- `text` 文本内容

继承节点

```yaml
节点ID:
  type: inherit
  template: 待继承节点ID
```

如上，相当于继承了对应节点的所有内容。例如：

```yaml
sections:
  templateTest: <strings::text1_text2_text3>
  inheritTest:
    type: inherit
    template: templateTest
```

其中`templateTest`有可能返回"text1"，"text2"或"text3"。

`inheritTest`同样有可能返回"text1"，"text2"或"text3"。

简单节点

```
节点ID: 值
```

如上所示，你直接添加节点的值。你可以搭配即时声明节点，优化你的配置。

比如：

```
节点ID: <strings::测试字符串1_测试字符串2_测试字符串3>
```

等效于

```
节点ID:
 type: strings
 values:
 - 测试字符串1
 - 测试字符串2
 - 测试字符串3
```

检查节点

```
节点ID:
 type: check
 value: 检测内容
```

- `value` 待检查内容
- `actions` 执行动作

有关condition写法的详细信息, 查看条件类型

`value`将作为变量传入`condition`供你判断, 示例配置如下:

```yaml
CheckTest:
  material: STONE
  name: <check>
  sections:
    # 待检查的节点, 随机返回test1, test2, test3中的一个值
    test:
      type: strings
      values:
        - test1
        - test2
        - test3
    check:
      type: check
      # 待检查的值
      value: <test>
      # 执行动作
      # 条件中默认导入了value
      actions:
        # 如果value为test1
        - condition: value == "test1"
          # 通知玩家
          actions:
          - "tell: 你得到了名为 test1 的物品"
          # value不为test1
          deny:
            # value为test2
            condition: value == "test2"
            # 通知玩家
```

When节点

```yaml
节点ID:
 type: when
 value: 1233211234567
 conditions:
 - condition: value == 114514
  result: nb
 - condition: value > 100
  result: 狠
 - "无匹配结果"
```

- `value` 待检查内容
- `conditions` 待进行的系列条件匹配

有关condition写法的详细信息, 查看条件类型

本节点将把value作为变量传入condition, 按照列表顺序进行一系列匹配, 如果条件满足则返回result中的结果

如果conditions中的某一条没有配置condition, 形如`- "无匹配结果"`, 节点将直接返回`无匹配结果`

示例配置如下:

```yaml
WhenTest:
 material: STONE
 name: <test> - <when>
 sections:
  test: <number::0_100>
  when:
   type: when
```

即时声明节点

> 节点配置内全面支持节点调用/PAPI调用

格式

`<节点类型::参数1_参数2_参数3...>`

即时声明节点无法指定节点ID, 如有需求，请配置私有/全局节点

即时声明节点中的`_`请用`\_`代替, 避免被当做参数分隔符, 这里的反斜杠不是转义符, 因此请注意当前最外层括号的引号类型

yaml语法中双引号包裹的`\`代表转义符, `\\`才是反斜杠, 单引号包裹的情况下则所见即所得

字符串节点

```
<strings::测试字符串1_测试字符串2_测试字符串3>
```

string节点将在各参数中随机返回一个

随机数节点

```
<number::0_10_0>
```

- `参数1` 随机数最小值
- `参数2` 随机数最大值
- `参数3` 保留小数位数

Gaussian节点

```
<gaussian::100_0.1_0.5_1_0_10000>
```

- `参数1` 基础数值
- `参数2` 浮动单位
- `参数3` 浮动范围上限
- `参数4` 小数保留位数 (默认为1)
- `参数5` 随机数最小值 (可不填)
- `参数6` 随机数最大值 (可不填)

关于Gaussian节点的详细介绍请看:

Gaussian节点

公式节点

```
<calculation::1+1+3+%player_level%_2_5_100>
```

- `参数1` 计算公式
- `参数2` 保留小数位数(默认为0, 即不保留小数)
- `参数3` 公式结果最小值(默认不限制最小值)

- `参数4` 公式结果最大值(默认不限制最大值)

> 信息
>
> 公式节点的本质是运行一段javascript代码
>
> 没有特殊需求应该优先使用快速计算(fastcalc)节点

快速计算节点

```
<fastcalc::1+1+3+%player_level%_2_5_100>
```

- `参数1` 计算公式
- `参数2` 保留小数位数(默认为0, 即不保留小数)
- `参数3` 公式结果最小值(默认不限制最小值)
- `参数4` 公式结果最大值(默认不限制最大值)

> 信息
>
> 快速计算节点的本质是解析数学符号进行分析计算
>
> 计算速度高于公式节点

权重节点

```
<weight::5::权重文本1_1::权重文本2>
```

参数格式 权重::权重文本

节点将根据权重随机返回一个权重文本

例如，在该示例节点中

将有5/6的几率返回"权重文本1"，1/6的几率返回"权重文本2"

PAPI节点

```
<papi::player_name>
```

参数为待解析文本

> 提示
>
> 节点解析前，物品会先全局解析一次papi变量。
>
> 因此直接写出的papi变量是不需要使用papi节点进行解析的。
>
> papi节点存在的意义是应对经过拼接的papi文本。
>
> 例如`<papi::<string-1><string-2>>`
>
> `<string-1>`返回`player_`
>
> `<string-2>`返回`name`

Javascript节点

```
<js::ExampleScript.js::main>
<js::ExampleScript.js::main_参数1_参数2_...>
```

参数格式 脚本路径::调用函数

或 脚本路径::调用函数_参数1_参数2_...

渐变色节点

```
<gradient::000000_FFFFFF_1_哈哈哈哈哈哈哈哈哈哈哈哈哈哈哈>
```

- `参数1` 起始颜色
- `参数2` 结尾颜色
- `参数3` 每几个字符变一次颜色
- `参数4` 文本内容

继承节点

```
<inherit::待继承节点ID>
```

如上，相当于继承了对应节点的所有内容。例如：

```
sections:
  templateTest: <strings::text1_text2_text3>
```

```
<inherit::templateTest>
```

其中`templateTest`有可能返回`text1`，`text2`或`text3`

即时声明节点`<inherit::templateTest>`同样有可能返回`text1`，`text2`或`text3`

自定义节点

自定义节点需要一定的 javascript 和 java 基础。

自定义节点文件存放于`NeigeItems/CustomSections`文件夹

下面是示例配置

```
// 文件名不重要, 写成啥都行
// main函数会自动执行
function main() {
  // 导入相应的类, 这两行看不懂的话直接抄就行
  const SectionManager =
Packages.pers.neige.neigeitems.manager.SectionManager.INSTANCE
  const CustomSection =
Packages.pers.neige.neigeitems.section.impl.CustomSection
  const SectionUtils = Packages.pers.neige.neigeitems.utils.SectionUtils

  // 创建自定义节点
  const customSection = new CustomSection(
    // 节点id
    "test",
    /**
     * 用于私有节点解析
     * @param data ConfigurationSection 节点内容
     * @param cache HashMap<String, String>? 解析值缓存
     * @param player OfflinePlayer? 待解析玩家
     * @param sections ConfigurationSection? 节点池
     * @return 解析值
     */
    function(data, cache, player, sections) {
      if (data.contains("values")) {
        // SectionUtils.parseSection("待解析字符串", cache, player, sections)用于解析
```

RGB颜色

```
<#FFFFFF>
```

如上所示

节点调用

节点可以在任意位置通过<节点ID>的形式调用

> **警告**
>
> 物品配置中出现的起装饰作用的<和>应替换为\<和\>，避免错误识别

高级应用

直接展示例子:

```yaml
stringTest:
  A:
    type: strings
    values:
    - test1
    - test2
  B:
    type: strings
    values:
    - test3
    - test4
```

如上配置节点后

调用`<stringTest.A>`将返回`test1`或`test2`

调用`<stringTest.B>`将返回`test3`或`test4`

如果这个节点是一个全局节点, 你可以通过

```yaml
globalsections:
- stringTest
```

引用该节点

扩展

简介

存放路径

功能详解

8 个项目

简介

存放路径

所有扩展文件应存放于`plugins/NeigeItems/Expansions`文件夹

重复 ID 的扩展仍然会被加载，但可能互相覆盖

最后哪个扩展活下来。。。随缘了属于是

功能

我尝试通过NeigeItems Expansion向你展示我对Nashorn脚本引擎的理解

你可以使用JavaScript语言, 通过Expansion实现几乎一切内容, 通俗来讲, 你可以通过JavaScript写"服务端插件"了

功能详解

基础知识

要使用JavaScript，你可能需要了解一些基础知识

自动触发

又或者说"生命周期"

类的引用

我在简介部分提到: 你可以使用JavaScript语言, 通过Expansion实现几乎一切内容

内置对象

Expansion中存在几个内置对象，你可以直接调用它们

📄 类的继承

在某些情况下，你可能需要继承并实现一个类。对此，Nashorn引擎有它神奇的实现方法。

📄 指令注册

NI Expansion提供了链式指令注册，示例如下：

📄 监听器注册

NI Expansion提供了链式监听器注册，示例如下：

📄 PAPI变量注册

NI Expansion提供了链式PAPI注册，示例如下：

基础知识

要使用JavaScript，你可能需要了解一些基础知识

扩展部分支持JavaScript的ES6标准，但没完全支持，如果你熟悉JavaScript，可以自行测试

变量定义

`let` 定义变量

```
let test = 1
```

`const` 定义常量

```
const test = 1
```

`不加符号直接定义` 定义全局变量/更改变量赋值

```
test = 1
```

函数定义

其实函数也是一种变量

普遍做法：

```
function test() {
  // 你的代码
}
```

将一个变量/常量赋值为函数：

```
test = function() {
  // 你的代码
}

let test = function() {
  // 你的代码
}

const test = function() {
  // 你的代码
}
```

全局变量

不加符号直接定义的变量称为全局变量，口头的描述可能过于贫乏，我选择使用例子解释：

```
function enable() {
  test = 1
}

function disable() {
  print(test)
}
```

重载后，后台显示：

```
[19:44:17 INFO]: 1
```

简单来说，只要你不重载插件或关服，全局变量就一直存在，你可以随便在哪里调用它。

弱类型语言

JavaScript是一门弱类型语言，所以你无需指明变量类型。

新建对象

跟Java差不多，以HashMap为例：

```
const HashMap = Packages.java.util.HashMap

const hashMap = new HashMap()
```

循环

跟Java十分类似：

```
for (let index = 0; index < array.length; index++) {
  const element = array[index]
}
```

但不存在Java的`增强for循环`

实际上JavaScript中存在类似`增强for循环`的`for-in`循环，但是相比于朴素的for循环，`for-in`循环会消耗更多的时间，因此我不做推荐，也不做讲解。

对于HashMap，你可以使用forEach：

```
const HashMap = Packages.java.util.HashMap

// 新建一个HashMap
const hashMap = new HashMap()
// 形如{1:1, 2:2, 3:3, 4:4}
for (let index = 0; index < 5; index++) {
   hashMap[index] = index
}
// 遍历键值
hashMap.forEach(function(key, value) {
   print(key)
   print(value)
   print("")
})
```

后台输出如下:

```
[19:55:09 INFO]: 0
[19:55:09 INFO]: 0
[19:55:09 INFO]:
[19:55:09 INFO]: 1
[19:55:09 INFO]: 1
[19:55:09 INFO]:
[19:55:09 INFO]: 2
[19:55:09 INFO]: 2
[19:55:09 INFO]:
[19:55:09 INFO]: 4
[19:55:09 INFO]: 4
[19:55:09 INFO]:
```

可见运行正常

调用指定参数方法

以高版本的`runTaskAsynchronously`为例：

`runTaskAsynchronously(Plugin plugin,Consumer<BukkitTask> task)`

`runTaskAsynchronously(Plugin plugin,Runnable task)`

在JavaScript中你可以这么操作：

```
Bukkit.getScheduler().runTaskAsynchronously(plugin, function() {
  print("123")
})
```

JavaScript的function会自动转换为Runnable或Consumer这些类似函数的东西。

但这样你就不知道你传入的到底是`Consumer<BukkitTask> task`还是`Runnable task`。

这时候你应该这么操作：

```
Bukkit.getScheduler()["runTaskAsynchronously(Plugin,Runnable)"](plugin, task)
```

我觉得你能看懂

自动触发

又或者说"生命周期"

NI Expansion会在某些阶段自动调用脚本中的某个函数，具体如下:

`enable` 服务器开启后 及 ni reload后异步执行

`disable` 服务器关闭前 及 ni reload前异步执行

`serverEnable` 服务器开启后异步执行

`serverDisable` 服务器关闭前异步执行

在.js文件中以如下形式体现:

```
/**
 * 服务器开启后 及 ni reload后异步执行
 */
function enable() {
}

/**
 * 服务器关闭前同步执行 及 ni reload前异步执行
 */
function disable() {
}

/**
 * 服务器开启后异步执行
```

你可以在function内部书写你要执行的内容，比如:

```
function enable() {
  print("看看远处的雪山吧家人们")
}
```

将在服务器启动时，及NeigeItems重载后，向后台发送"看看远处的雪山吧家人们"

同步?异步?

> 信息
>
> 如果你只是想简单地试用这个功能，那你现在还不需要看这一节，先跳过去吧。
>
> 如果你没有相关知识铺垫，你很可能永远无法把握住同步异步的切换问题。

你可能注意到了注释中的一些话语: `服务器开启后 及 ni reload后异步执行`

与异步代表在另外一个线程执行你的操作，不会对服务器的主要内容造成影响，而同步执行代表在主线程执行，也就是，执行这段代码的时候万物静止，等你这段代码执行完，才能处理其他的事情。假如你在主线程执行了一段耗时10分钟的代码,，很明显，你的服务器将中途崩溃。

但主线程也不是一无是处，有一些操作你需要在主线程进行，确保安全。比如一个经典的BUG：手速卡物品。这种BUG一般源于多线程处理，作者为了降低插件对服务器的影响，新开了一个线程处理相关事宜。你可以理解为：当前运行代码和服务器事务并驾齐驱，同时运行。

比较典型的糟糕案例有：

我判断玩家持有300金币，然后给予物品，并扣除300金币。

玩家起初有300金币，在通过你的判断后消耗掉了这些金币（比如购买了其他物品）。于

是，在你扣除金币时，玩家金币不足300，比如仅剩50，你就只扣除了50金币。于是玩家通过50金币购买得到了300金币的物品。

类的引用

我在简介部分提到: `你可以使用JavaScript语言, 通过Expansion实现几乎一切内容`

那么你一定需要Java中的import，很明显，没有import什么都干不了。下面我写出一段对比：

```
// Java
import org.bukkit.Bukkit
```

```
// JavaScript
const Bukkit = Packages.org.bukkit.Bukkit
```

我们需要通过Packages获取对应的类, 并通过const将它赋值给同名变量。

当然，你也可以自行更改名字。比如：

```
const r = Packages.java.util.concurrent.ThreadLocalRandom

// 向后台发送一个0-1的随机数
print(r.current().nextDouble())
```

你可以按你所想安排这段代码的存放位置，比如：

```
const r = Packages.java.util.concurrent.ThreadLocalRandom

function enable() {
  print(r.current().nextDouble())
```

又或者：

```
function enable() {
  const r = Packages.java.util.concurrent.ThreadLocalRandom

  print(r.current().nextDouble())
}
```

没什么差别，这不会造成性能差异。

内置对象

Expansion中存在几个内置对象，你可以直接调用它们

```
// Bukkit类
const Bukkit = Packages.org.bukkit.Bukkit
// 用来注册指令
const Command = Packages.pers.neige.neigeitems.script.tool.ScriptCommand
// 注册监听器时指定的优先级
const EventPriority =
Packages.pers.neige.neigeitems.taboolib.common.platform.event.EventPriority
// 用来注册监听器
const Listener = Packages.pers.neige.neigeitems.script.tool.ScriptListener
// 用来注册PAPI变量
const Placeholder = Packages.pers.neige.neigeitems.script.tool.ScriptPlaceholder
// 一些NI的工具类和管理器
const ActionUtils = Packages.pers.neige.neigeitems.utils.ActionUtils
const ConfigUtils = Packages.pers.neige.neigeitems.utils.ConfigUtils
const FileUtils = Packages.pers.neige.neigeitems.utils.FileUtils
const ItemUtils = Packages.pers.neige.neigeitems.utils.ItemUtils
const JsonUtils = Packages.pers.neige.neigeitems.utils.JsonUtils
const LangUtils = Packages.pers.neige.neigeitems.utils.LangUtils
const PlayerUtils = Packages.pers.neige.neigeitems.utils.PlayerUtils
const SamplingUtils = Packages.pers.neige.neigeitems.utils.SamplingUtils
const ScriptUtils = Packages.pers.neige.neigeitems.utils.ScriptUtils
const SectionUtils = Packages.pers.neige.neigeitems.utils.SectionUtils
const StringUtils = Packages.pers.neige.neigeitems.utils.StringUtils
const ActionManager =
Packages.pers.neige.neigeitems.manager.ActionManager.INSTANCE
const ConfigManager =
Packages.pers.neige.neigeitems.manager.ConfigManager.INSTANCE
const HookerManager = Packages.pers.neige.neigeitems.manager.HookerManager
```

我解释一下sync和async的使用方法：

```
// 在主线程通过后台发送"123"
sync(function() {
  print("123")
})
```

只要往里面塞一个function就好了

类的继承

在某些情况下，你可能需要继承并实现一个类。对此，Nashorn引擎有它神奇的实现方法。

以Runnable为例：

```
const Runnable = Packages.java.lang.Runnable

const MyRunnable = Java.extend(Runnable, {
  run: function() {
    print(123)
  }
})

new MyRunnable().run()
```

执行后后台输出:

```
[19:31:51 INFO]: 123
```

另外还有一种实现方式：

```
const Runnable = Packages.java.lang.Runnable

new Runnable {
  run: function() {
    print(123)
  }
}.run()
```

第二种方式可能让你的编辑器发出悲鸣，因为这并不符合JavaScript语法，是Nashorn自己的操作。

> ⚠ 警告
>
> 在Nashorn中，类的继承是非常耗时的，你应该避免高频率继承

指令注册

NI Expansion提供了链式指令注册，示例如下：

```
/**
 * 服务器开启后 及 ni reload后异步执行
 */
function enable() {
  // 调用指令注册
  commandExample()
}

/**
 * 指令注册示例
 */
function commandExample() {
  /**
   * 新建一个名为test的指令
   * 你需要在最后通过.register()注册它
   * 在ni reload前, 所有脚本执行过disable后, 所有通过脚本注册的指令都会自动取消注册
   * 因此你需要注册就好, 不用担心后续的事情
   * 插件内部已经做过同步处理, 因此你可以放心异步执行注册
   */
  new Command("test")
    // 设置指令命名空间, 默认命名空间为指令名(test:test), 当前设置后可通
过"test"或"neige:test"执行指令
    .setNameSpace("neige")
    // 我真不知道label是啥, 这玩意儿默认是跟指令名一样的
    .setLabel("test")
    // 指令别名, test指令可以用tt或ttt代替
    .setAliases(["tt", "ttt"])
    // 指令所需权限
```

相信你能通过注释了解指令注册的全部内容，但为了兼顾你是个懒蛋的情况，我还要在此写一个最简单的指令注册：

```
function enable() {
  commandExample()
}

function commandExample() {
  new Command("test")
    .setExecutor(
      function(sender, command, label, args) {
        sender.sendMessage("测试指令")
        return true
      }
    )
    .setTabCompleter(
      function(sender, command, label, args) {
        return ["测试补全"]
      }
    )
    .register()
}
```

看不懂说明你有问题

监听器注册

NI Expansion提供了链式监听器注册，示例如下：

```
/**
 * 服务器开启后 及 ni reload后异步执行
 */
function enable() {
  // 调用监听器注册
  listenerExample()
}

/**
 * 监听器注册示例
 */
function listenerExample() {
  /**
   * 新建一个AsyncPlayerChatEvent事件的监听器
   * 你需要在最后通过.register()注册它
   * 在ni reload前, 所有脚本执行过disable后, 所有通过脚本注册的监听器都会自动取消注册
   * 因此你需要注册就好, 不用担心后续的事情
   * 插件内部已经做过同步处理, 因此你可以放心异步执行注册
   */
  new Listener(Packages.org.bukkit.event.player.AsyncPlayerChatEvent.class)
    /**
     * 事件监听优先级, 默认为EventPriority.NORMAL(所以这行实际上不用写)
     * 可用的优先级有:
     * EventPriority.LOWEST (最先处理)
     * EventPriority.LOW
     * EventPriority.NORMAL
     * EventPriority.HIGH
```

相信你能通过注释了解监听器注册的全部内容，但为了兼顾你是个懒蛋的情况，我还要在此写一个最简单的监听器注册：

```
function enable() {
  listenerExample()
}

function listenerExample() {
  new Listener(Packages.org.bukkit.event.player.AsyncPlayerChatEvent.class)
    .setExecutor(
      function(event) {
        event.player.sendMessage(
          '你是不是打算说: " ' + event.message + ' " ?'
        )
      }
    )
    .register()
}
```

看不懂说明你有问题

PAPI变量注册

NI Expansion提供了链式PAPI注册，示例如下：

```
/**
 * 服务器开启后 及 ni reload后异步执行
 */
function enable() {
  // 调用PAPI变量注册
  placeholderExample()
}

/**
 * PAPI变量注册示例
 */
function placeholderExample() {
  /**
   * 新建一个名为test的papi变量
   * 你需要在最后通过.register()注册它
   * 在ni reload前, 所有脚本执行过disable后, 所有通过脚本注册的papi变量都会自动取消注册
   * 因此你需要注册就好, 不用担心后续的事情
   * 插件内部已经做过同步处理, 因此你可以放心异步执行注册
   * 插件内部已经做过兼容处理, 因此不安装PlaceholderAPI不会导致这段代码报错, 只会让它不生效
   */
  new Placeholder("test")
    // 设置变量作者, 默认为"unknown"(所以这行实际上不用写)
    .setAuthor("Neige")
    // 设置变量版本, 默认为"1.0.0"(所以这行实际上不用写)
    .setVersion("1.0.0")
    /**
```

相信你能通过注释了解PAPI注册的全部内容，但为了兼顾你是个懒蛋的情况，我还要在此写一个最简单的PAPI注册：

```
function enable() {
  placeholderExample()
}

function placeholderExample() {
  new Placeholder("test")
    .setExecutor(
      function(player, params) {
        // %test% 返回玩家名
        return player.name
      }
    )
    .register()
}
```

看不懂说明你有问题

脚本

JavaScript

对象与函数

JavaScript

对象与函数

NeigeItems 的 JavaScript 节点目前提供以下对象

- `this.player` 即 玩家本身
- `arguments` 你调用时传入的参数，以数组形式出现

提供以下函数

- `this.vars(String text)` 解析替换文本中的节点
- `this.papi(String text)` 解析替换文本中的papi变量

路径

所有脚本文件应存放于`plugins/NeigeItems/Scripts`文件夹

插件适配

mythicmobs

5 个项目

PlaceholderAPI

即时声明节点解析

Invero(俗称TrMenuV4)

这b插件的wiki指望不得，还是得我自己写

mythicmobs

物品掉落

关配置支持解析即时声明变量

多彩掉落

相关配置支持解析即时声明变量

物品穿戴

相关配置支持解析即时声明变量

让穿戴物品随机掉落

相关配置支持解析即时声明变量

📄 掉落物品触发技能

你可以让MM怪物死亡后掉落的NI物品触发指定MM技能，详见下方链接：

物品掉落

> 关配置支持解析即时声明变量

在MM怪物的配置中添加

```
NeigeItems:
 # 类似物品lore, Drops可以通过换行符"\n"换行
 Drops:
 - 物品ID 随机最低数量-随机最高数量 掉落概率 是否重复随机 指向数据
```

物品ID可以是NI物品ID或者MM物品ID、EasyItem物品ID，优先检测NI物品

`随机最低数量-随机最高数量` 可以直接写数量

`掉落概率` 不写的话默认为1

`是否重复随机` 默认重复随机(对于MM物品, 这个配置项不代表是否随机生成, 代表物品是否合并)

`指向数据` 想写的话正常写就行

下面我写几个MM怪物示例配置

```
test1:
 Type: ZOMBIE
 Health: 1
 NeigeItems:
  Drops:
  # 50%掉落1-5个ID为"itemId"的NI物品(或MM物品、EasyItem物品)
```

顺带一提，因为整体支持调用即时声明节点，你可以通过节点自定义你的掉落概率（可根据权限、变量、等级、生命等一系列因素决定掉落概率）。下面我写一个最简单的例子

```
test4:
  Type: ZOMBIE
  Health: 1
  NeigeItems:
    Drops:
    # 掉落玩家等级数量的ID为"itemId"的NI物品(或MM物品、EasyItem物品)
    - itemId <papi::player_level>
```

或者，你可以直接给MM怪物配置掉落物品包

```
NeigeItems:
  DropPacks:
  - 物品包ID 指向数据
  - 物品包ID 指向数据
  - 物品包ID 指向数据
```

其中指向数据可省略

插件将直接读取识别对应的物品组并添加掉落物及多彩掉落配置

下面我写几个MM怪物示例配置

```
test2:
  Type: ZOMBIE
  Health: 1
  NeigeItems:
    DropPacks:
    - Example
```

默认情况下只有玩家击杀才会掉落物品, 如果你希望自然死亡的怪物掉落物品, 可以将怪物

的`PlayerOnly`选项设置为`false`：

```yaml
test1:
  Type: ZOMBIE
  Health: 1
  NeigeItems:
    PlayerOnly: false
    Drops:
    - ExampleItem 1
```

多彩掉落

> 相关配置支持解析即时声明变量

在MM怪物的配置中添加

掉落物可以像无主之地一样喷射到空中，具体配置方法如下

```
NeigeItems:
  FancyDrop:
    offset:
      x: 横向偏移
      y: 纵向偏移
    angle:
      type: 旋转方式
```

`横向偏移` 表示物品向四周弹射的力度, 是一个数字

`纵向偏移` 表示物品向空中弹射的力度, 是一个数字

`旋转方式` 决定物品的弹射角度，是一个个绕一圈弹出去，还是随机弹出去(分为random和round)

下面我写几个MM怪物配置实例：

```
test1:
  Type: ZOMBIE
  Health: 1
  NeigeItems:
    FancyDrop:
```

```yaml
test1:
  Type: ZOMBIE
  Health: 1
  NeigeItems:
    FancyDrop:
      offset:
        # 随机偏移值
        x: 0-0.1
        # 随机偏移值
        y: 1-1.5
      angle:
        # 随机角度弹出去
        type: random
```

物品穿戴

> 相关配置支持解析即时声明变量

在MM怪物的配置中添加

```
NeigeItems:
  Equipment:
  - 穿戴位置: 物品ID 穿戴概率 指向数据
```

物品ID可以是NI物品ID、MM物品ID或EasyItem物品ID

可用的穿戴位置都有：

- `Helmet` 代表头部
- `Chestplate` 代表胸部
- `Leggings` 代表腿部
- `Boots` 代表脚部
- `MainHand` 代表主手
- `OffHand` 代表副手

穿戴概率默认为1

下面我写一个MM怪物示例配置

```
test1:
  Type: ZOMBIE
  Health: 1
  NeigeItems:
```

让穿戴物品随机掉落

> 相关配置支持解析即时声明变量

众所周知MM不能直接让怪物穿戴的装备掉落

如果需要这种功能，只能在掉落物里配置跟装备一样的东西

但是NI是一个注重随机的插件，你这样操作的话，最后怪物穿的跟怪物掉的很可能不是一个东西

所以NI特意提供了相关的配置，配置如下

在MM怪物的配置中添加

```
NeigeItems:
 DropEquipment:
 - 掉落位置 掉落概率
```

可用的掉落位置都有：

- `Helmet` 代表头部
- `Chestplate` 代表胸部
- `Leggings` 代表腿部
- `Boots` 代表脚部
- `MainHand` 代表主手
- `OffHand` 代表副手

掉落概率默认为1

下面我写一个MM怪物示例配置

```yaml
test1:
  Type: ZOMBIE
  Health: 1
  NeigeItems:
    Equipment:
    - 'Helmet: Helmet1 0.5'
    - 'Chestplate: Chestplate1'
    - 'Leggings: Leggins1 0.5'
    - 'Boots: Boots1 0.5'
    - 'MainHand: MainHand1 0.5'
    - 'OffHand: OffHand1 0.5'
    DropEquipment:
    # 头部NI装备50%掉落
    - Helmet 0.5
    # 胸部NI装备100%掉落
    - Chestplate
    - Leggings 0.5
    - Boots 0.5
    - MainHand 0.5
    - OffHand 0.5
```

掉落物品触发技能

你可以让MM怪物死亡后掉落的NI物品触发指定MM技能，详见下方链接：

掉落技能

PlaceholderAPI

即时声明节点解析

- `%ni_parse_内容%`

  例：`%ni_parse_<number::0_1_5>%` 返回0-1保留5位小数的随机数，如0.45784

Invero(俗称TrMenuV4)

> 这b插件的wiki指望不得，还是得我自己写

调用NI/MM物品

我写个例子:

```yaml
title: 'NI物品测试'

layout: '  *  '

items:
  '*':
    texture:
      source: ni
      value: ExampleItem
```

你看不懂我只能说你是个傻逼

其中的`source: ni`也可以写成`source: neigeitems`，没区别

`value:`后面跟物品ID，也可以加指向数据，比如`value: ExampleItem {"test1":"test1","test2":"test2"}`

物品ID优先检测NI物品，获取不到就会尝试获取MM物品

应用实例

生成多个非重复词条

等概率选取

MythicMobs怪物掉落实体金币

效果：

通过节点插入多行Lore

方法1

生成随机强度条

方法1

📄 生成随机数量宝石槽

方法

生成多个非重复词条

等概率选取

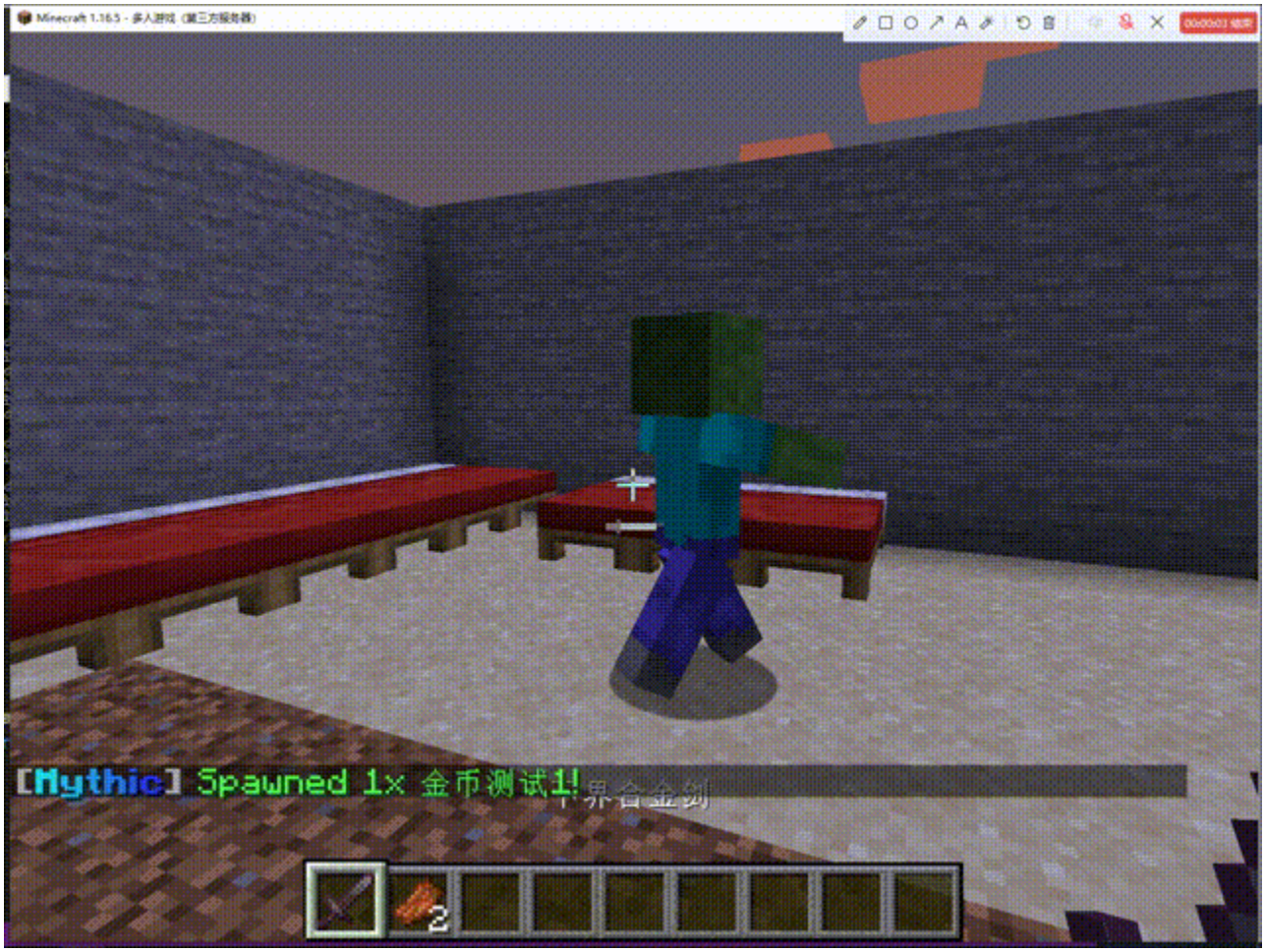
```yaml
# 等概率不重复词条
JoinTest8:
 material: STONE
 lore:
  # 等同于随机出现三行不重复词条
  - '<test>'
 sections:
  test:
   type: join
   list:
    - '攻击力: 100'
    - '攻击百分比: 10%'
    - '防御力: 100'
    - '防御百分比: 10%'
    - '生命值: 100'
    - '生命百分比: 10%'
    - '暴击率: 10%'
    - '暴击伤害: 10%'
   # 限制最多出3条
   limit: 3
   # 是否打乱顺序
   shuffled: true
   # 像下面这样写分隔符、前缀和后缀
   # 即可达到调用多行lore的效果
   separator: "\\n"
   prefix: '""'
   postfix: '""'
```

根据权重选取

```yaml
WeightJoinTest:
  material: STONE
  lore:
  - '<test>'
  sections:
    test:
      type: weightjoin
      list:
      # 权重::取值
      - '1::暴击率: 10%'
      - '1::暴击伤害: 10%'
      - '2::攻击力: 100'
      - '2::攻击百分比: 10%'
      - '3::防御力: 100'
      - '3::防御百分比: 10%'
      - '4::生命值: 100'
      - '4::生命百分比: 10%'
      # 选取3条
      amount: 3
      # 是否打乱顺序
      shuffled: true
      # 像下面这样写分隔符、前缀和后缀
      # 即可达到调用多行lore的效果
      separator: "\\n"
      prefix: '""'
      postfix: '""'
```

MythicMobs怪物掉落实体金币

效果：



需配置以下文件：

plugins\NeigeItems\Expansions\金币.js

```
function enable() {
  // 监听物品合并事件
  new Listener(Packages.org.bukkit.event.entity.ItemMergeEvent.class)
    .setPriority(EventPriority.LOWEST)
    .setExecutor(
      function(event) {
        async(function() {
          loadItem(event.getTarget())
        })
      }
    )
    .register()

  // 监听物品生成事件
  new Listener(Packages.org.bukkit.event.entity.ItemSpawnEvent.class)
    .setPriority(EventPriority.LOWEST)
    .setExecutor(
      function(event) {
        async(function() {
          loadItem(event.getEntity())
        })
      }
    )
    .register()
}

function getItemName(itemStack) {
  const ItemUtils = Packages.pers.neige.neigeitems.utils.ItemUtils
  const HookerManager = Packages.pers.neige.neigeitems.manager.HookerManager

  const itemInfo = ItemUtils.isNiItem(itemStack, true)
  if (itemInfo == null) return null

  const data = itemInfo.data
  if (data == null) return null
```

plugins\NeigeItems\ItemActions\金币.yml

```yaml
金币:
  pick:
    cooldown: 0
    consume:
      amount: 1
    sync:
    - 'giveMoney: <data::金币数量>'
    - 'tell: §e[NI] §6恭喜你获得了 <data::金币数量> 金币'
```

plugins\NeigeItems\Items\金币.yml

```yaml
金币:
  material: GOLD_NUGGET
  name: §e<金币数量> 金币
  sections:
    # 定义获取的金币数量
    金币数量: 100
```

plugins\MythicMobs\Mobs\金币.yml

```yaml
金币测试1:
  Type: ZOMBIE
  Health: 1
  NeigeItems:
    Drops:
    # 金币 掉落1个 100%概率掉落
    - 金币 1 1
金币测试2:
  Type: ZOMBIE
  Health: 1
  NeigeItems:
    Drops:
```

通过节点插入多行Lore

方法1

```yaml
多行Lore测试1:
  material: STONE
  lore:
  - <多行Lore>
  sections:
    多行Lore:
      type: join
      list:
        - 第一行
        - 第二行
        - 第三行
      separator: "\\n"
      prefix: ""
      postfix: ""
```

方法2

```yaml
多行Lore测试2:
  material: STONE
  lore:
  - <多行Lore>
  sections:
    多行Lore: '"第一行\n第二行\n第三行"'
```

方法3

```yaml
多行Lore测试3:
  material: STONE
  lore:
  - <多行Lore>
  sections:
    多行Lore:
      type: repeat
      content: 啦啦啦
      repeat: 3
      separator: "\\n"
      prefix: '"'
      postfix: '"'
```

值得一提

> **警告**
>
> 本段内容较为复杂，如果你没有打破砂锅问到底的闲心，请跳过本部分

> **信息**
>
> > 提问: 根据你写的配置，以方法1为例，`多行Lore`这一节点的返回值应该为`"第一行\\n第二行\\n第三行"`。你为什么要在两边加上双引号?你为什么要使用`"\\n"`?根据yaml语法，`"\\n"`应该代表形似`\n`的字符，`"\n"`才是换行符，你在搞什么，为什么最后这段配置运行正常？我想打死你:)
>
> > 回答: 世界比你想象的更加复杂，你先别急，让我先急:)
>
> 首先，`多行Lore`这一节点的返回值不是`"第一行\\n第二行\\n第三行"`，而是`"第一行\n第二行\n第三行"`。
>
> 对于这个join节点，节点返回值应该为`前缀`+`列表的第一项`+`分隔符`+`列表的第二项`+`分隔符`+`列表的第三项`+`后缀`。
>
> 这个过程是拼接出来的。所以`"\\n"`作为字符体现为`\n`
>
> 故结果为`"第一行\n第二行\n第三行"`

下面我来解释一下不在两边加上双引号，并直接用换行符做separator会发生什么

替换前:

```
多行Lore测试1:
  material: STONE
  lore:
  - <多行Lore>
```

替换后:

```
多行Lore测试1:
  material: STONE
  lore:
  - 第一行
第二行
第三行
```

是的，换行符不会以换行符形式出现，会真的换行（微笑）

所以我们需要形似`"第一行\n第二行\n第三行"`的返回值

替换后:

```
多行Lore测试1:
  material: STONE
  lore:
  - "第一行\n第二行\n第三行"
```

读取后刚好是正确的格式

> 提问: 我花一年时间理解了你上面那个屌问题。这种狗东西你怎么写出来的，脑测吗?

> 回答：请善用debug。将`plugin/NeigeItems/config.yml`中的`Main.Debug`设置为true即可开启debug模式。

开启后效果见下图

```
[22:41:32 INFO]: Neige issued server command: /ni get 多行Lore测试1
[22:41:32 INFO]: 多行Lore测试1:
[22:41:32 INFO]:   material: STONE
[22:41:32 INFO]:   lore:
[22:41:32 INFO]:   - "第一行\n第二行\n第三行"
[22:41:32 INFO]:   sections:
[22:41:32 INFO]:     多行Lore:
[22:41:32 INFO]:       type: join
[22:41:32 INFO]:       list:
[22:41:32 INFO]:       - 第一行
[22:41:32 INFO]:       - 第二行
[22:41:32 INFO]:       - 第三行
[22:41:32 INFO]:       separator: \n
[22:41:32 INFO]:       prefix: ''
[22:41:32 INFO]:       postfix: ''
```

警告

错误示范见下图

```
[22:43:09 INFO]: Neige issued server command: /ni get 多行Lore测试1
[22:43:09 INFO]: 多行Lore测试1:
[22:43:09 INFO]:   material: STONE
[22:43:09 INFO]:   lore:
[22:43:09 INFO]:   - 第一行
[22:43:09 INFO]: 第二行
[22:43:09 INFO]: 第三行
[22:43:09 INFO]:   sections:
[22:43:09 INFO]:     多行Lore:
[22:43:09 INFO]:       type: join
[22:43:09 INFO]:       list:
[22:43:09 INFO]:       - 第一行
[22:43:09 INFO]:       - 第二行
[22:43:09 INFO]:       - 第三行
[22:43:09 INFO]:       separator: |2+
[22:43:09 INFO]:
[22:43:09 WARN]: [NeigeItems] Plugin NeigeItems v1.5.7 generated an exception while executing task 4337
org.bukkit.configuration.InvalidConfigurationException: while scanning a simple key
 in 'string', line 5, column 1:
    第二行
    ^
could not find expected ':'
 in 'string', line 6, column 1:
    第三行
    ^

        at org.bukkit.configuration.file.YamlConfiguration.loadFromString(YamlConfiguration.java:59) ~[pa
tched_1.16.5.jar:git-Paper-794]
        at pers.neige.neigeitems.utils.ConfigUtils.loadFromString(ConfigUtils.kt:203) ~[?:?]
        at pers.neige.neigeitems.item.ItemGenerator.getItemStack(ItemGenerator.kt:159) ~[?:?]
        at pers.neige.neigeitems.manager.ItemManager.getItemStack(ItemManager.kt:137) ~[?:?]
        at pers.neige.neigeitems.command.Command.giveCommand(Command.kt:1073) ~[?:?]
        at pers.neige.neigeitems.command.Command.giveCommand(Command.kt:995) ~[?:?]
        at pers.neige.neigeitems.command.Command.access$giveCommand(Command.kt:45) ~[?:?]
        at pers.neige.neigeitems.command.Command$giveCommandAsync$1.invoke(Command.kt:1007) ~[?:?]
        at pers.neige.neigeitems.command.Command$giveCommandAsync$1.invoke(Command.kt:1006) ~[?:?]
        at pers.neige.neigeitems.taboolib.platform.BukkitExecutor$submit$6.run(BukkitExecutor.kt:89) ~[?:
?]
        at org.bukkit.craftbukkit.v1_16_R3.scheduler.CraftTask.run(CraftTask.java:101) ~[patched_1.16.5.j
ar:git-Paper-794]
        at org.bukkit.craftbukkit.v1_16_R3.scheduler.CraftAsyncTask.run(CraftAsyncTask.java:54) ~[patched
_1.16.5.jar:git-Paper-794]
        at com.destroystokyo.paper.ServerSchedulerReportingWrapper.run(ServerSchedulerReportingWrapper.ja
va:22) ~[patched_1.16.5.jar:git-Paper-794]
        at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1130) ~[?:?]
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:630) ~[?:?]
```

生成随机强度条

方法1

```yaml
强度条测试1:
  material: STONE
  lore:
    - '&4<强度条1>&f<强度条2>'
  sections:
    强度条1:
      type: repeat
      content: "|"
      repeat: <number>
    强度条2:
      type: repeat
      content: "|"
      repeat: <calculation::20-<number>>
    number:
      type: number
      min: 0
      max: 20
      fixed: 0
```

方法2

```yaml
强度条测试2:
  material: STONE
  lore:
    - '<强度条>'
  sections:
    强度条:
      type: repeat
      content: "|"
      repeat: 20
      prefix: " §4"
      transform: |-
        if (this.index == this.vars("<number>")) {
            return " §f" + this.it
        } else {
            return this.it
        }
    number:
      type: number
      min: 0
      max: 20
      fixed: 0
```

值得一提

警告

本段内容较为复杂，如果你没有打破砂锅问到底的闲心，请跳过本部分

信息

> 提问: 你为什么方法1的颜色符号用&，到了方法2里就用 § 了。你是不是歧视&

> 回答: 理解过程你可以开debug自己理解一下。 总的来说，如果用&，换完就变成了

```
多行Lore测试1:
  material: STONE
  lore:
  - &4||||||||||||||&f||||||
```

> 没有引号包裹，最前面的那个&4会被识别成yaml语法中的锚点, 所以用 § 而不是 &

生成随机数量宝石槽

方法

```yaml
宝石槽测试:
  material: STONE
  lore:
    - '<宝石槽>'
  sections:
    宝石槽:
      type: repeat
      content: '&4&l<宝石槽>'
      # 随机1-4个
      repeat: <number::1_4_0>
      separator: "\\n"
      prefix: ""
      postfix: ""
```

傻逼式教程

前言

在一般教程中，这部分一般被称为“傻瓜式教程”。

没头脑篇

没头脑篇对应没有大脑的人，在本节中，我将尽可能将过程详细描述，并辅以图片帮助理解，以尽量写出你能看懂的文字。

不高兴篇

不高兴篇应对不愿意看wiki的傻逼，在本节，我可能大量引用wiki内链接，告诉你这个傻逼，这个问题wiki里已经你妈的写了。

前言

在一般教程中，这部分一般被称为“傻瓜式教程”。

可当一个个傻逼进群提出一些wiki中有，亦或是不过大脑的问题后，我认为“傻瓜”不太能抒发我内心的愤懑，因此，这部分被我称为“傻逼式教程”，即：写给傻逼看的教程。

在正文部分，我将顺序介绍NeigeItems的使用。我将以我个人浅薄的想象力尽量介绍到每个方面，如果你看完还是不懂，请自杀。你可以在网页左侧点击标题进行快速跳转，别寄吧说什么太长翻起来太麻烦，请左转跳楼。

如果你看完wiki没看懂，你可以选择使用其他插件。

如果你wiki也不看帖子也不看，那我推荐你看看傻逼式教程。

没头脑篇

没头脑篇对应没有大脑的人，在本节中，我将尽可能将过程详细描述，并辅以图片帮助理解，以尽量写出你能看懂的文字。

插件下载

你有三种选择下载到NeigeItems.jar文件：

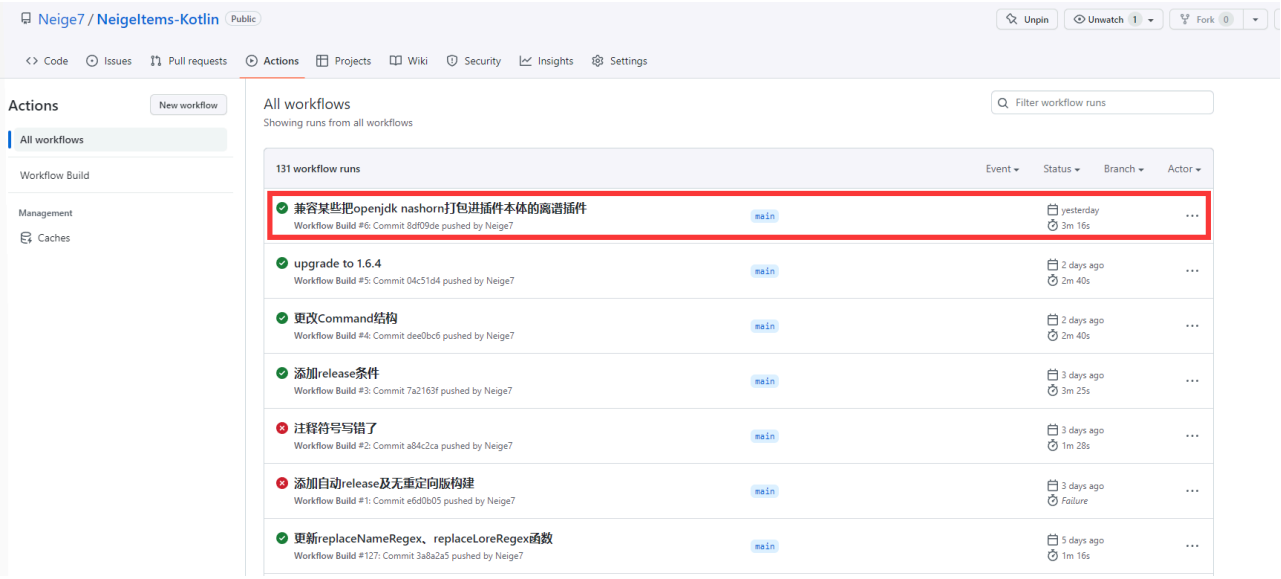
MCBBS插件贴

前往本插件的MCBBS插件贴，往下翻，翻到“插件下载”部分，下载里面的NeigeItems-版本.jar文件。其中“版本”代表当时的插件版本。比如本节wiki书写时，最新版本为1.6.5，你就应该下载NeigeItems-1.6.5.jar文件。别寄吧问我为什么没找到叫“NeigeItems-版本.jar”的文件，只看到了一个什么“NeigeItems-1.X.X.jar”，你敢问你就他妈的命不久矣。为防止观看者找不到“插件下载”部分，说我没提供，我在此截图：

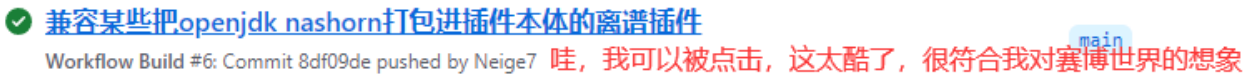
画红框的部分就是你要下载的文件

Github自动构建

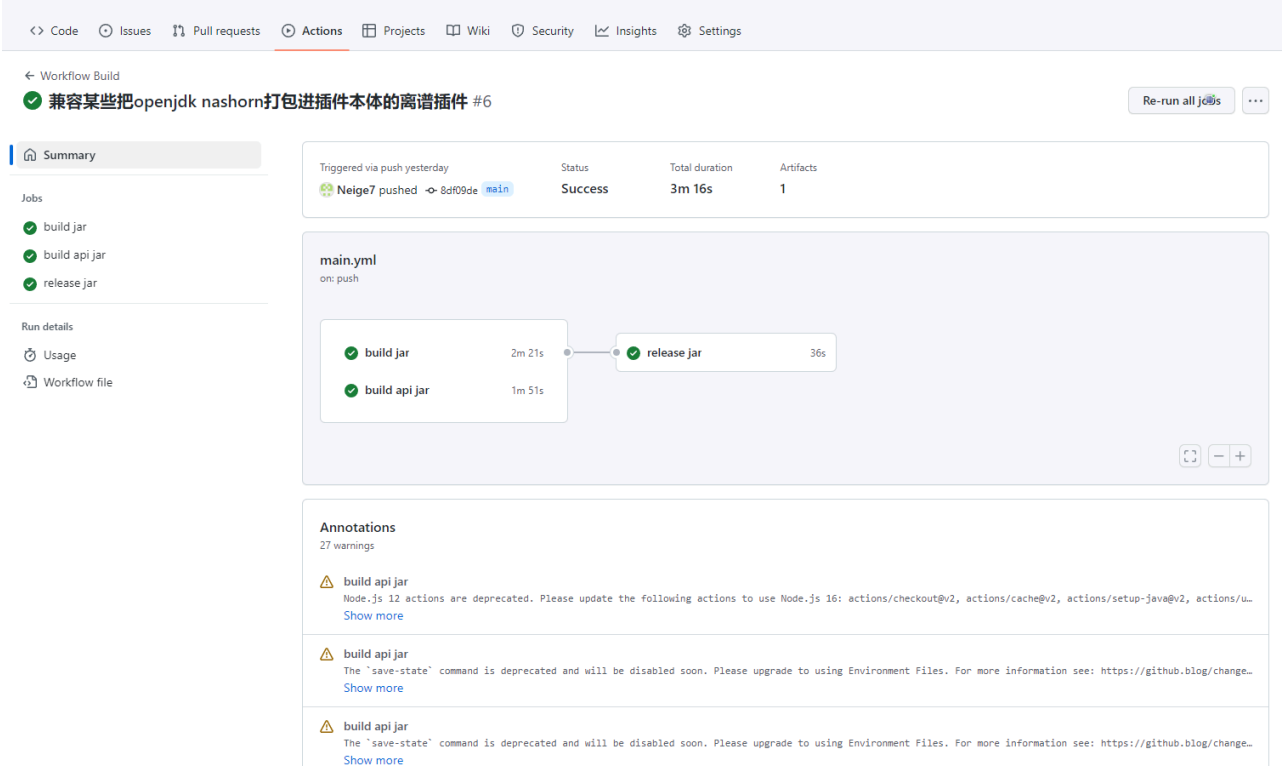
前往Github自动构建下载。首先，你需要登录Github账号。不登录Github账号是下载不了自动构建的。别寄吧问我Github怎么注册，这是NeigeItems的wiki，不是他妈Github的wiki。下面，我将通过图片展示自动构建页面：



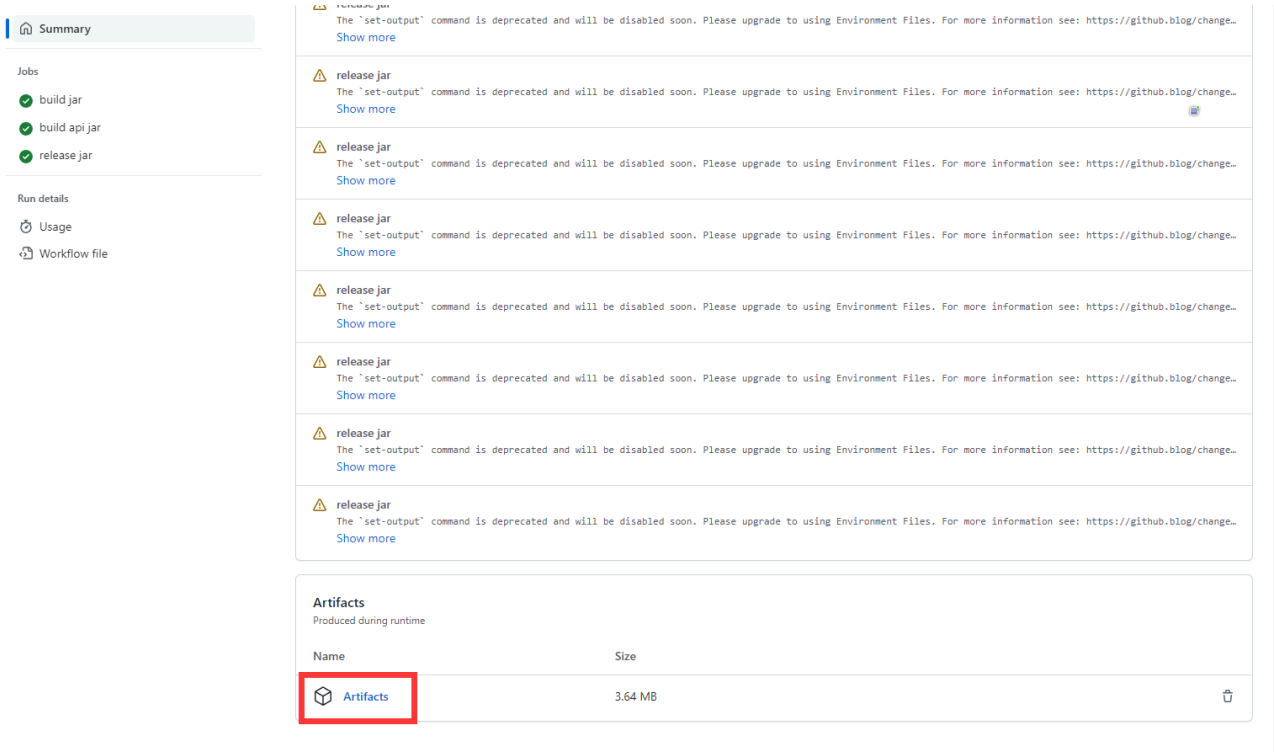
画红框的部分，即最上面一条自动构建，就是我们的目标。将鼠标移动到文字上后，我们会惊讶地发现，这行文字可以点击：



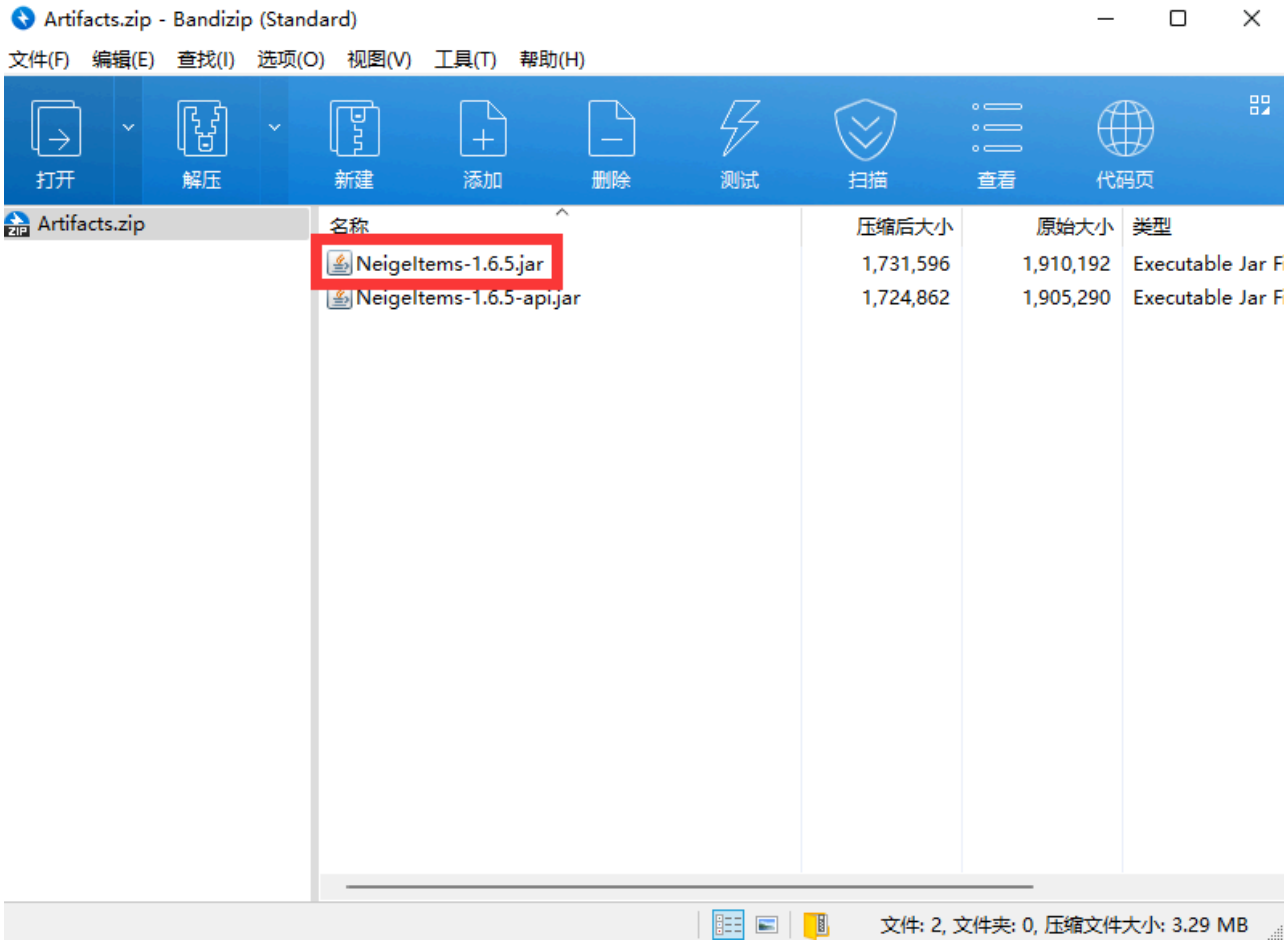
点击进入最新的自动构建，我们将看到如下界面：



此时你可能会问：你妈的，花花绿绿，我要下载的插件在哪里？这页面这么花，插件作者的人品一定有问题。但是，你先别急，让我先急。动动你的鼠标滚轮，翻到页面的最下面：



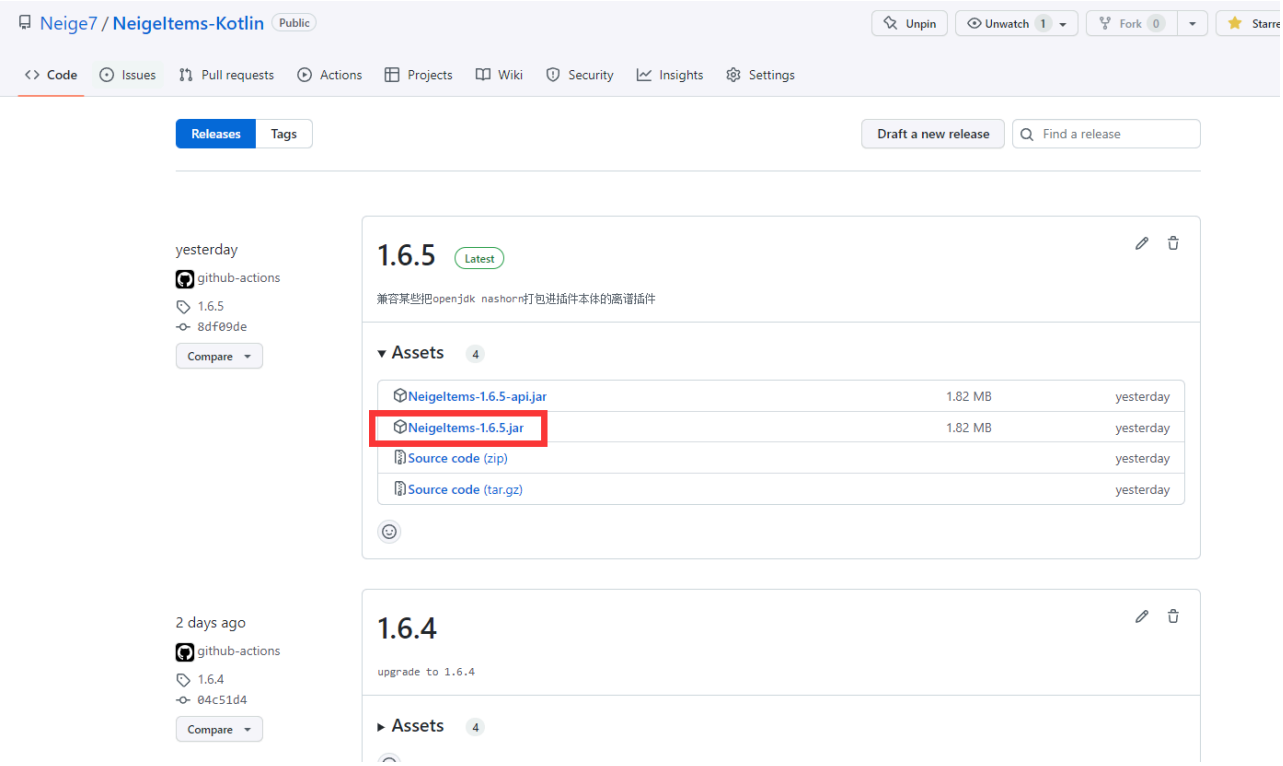
看到红框中的“Artifacts”了吗？点击他，你就可以开始下载自动构建了。你将通过下载得到Artifacts.zip文件，使用解压软件打开，你可以看到：



你需要的就是这个“NeigeItems-1.6.5.jar”。别jb用那个“NeigeItems-1.6.5-api.jar”，这玩意儿没有经过重定向，是用来在写插件的时候引作依赖的。

Github Releases

前往Github Releases下载。为防止有人不知道怎么下载自动构建，从1.6.2版本开始，NeigeItems会自动将自动构建发布为Release。Releases中的附件不会过期，且可以直接下载。下面看图：



红框中的“NeigeItems-1.6.5.jar”就是你需要的文件，别jb问我那个带-api的东西是什么，那是写插件的时候用来当依赖引用的。

无法下载?

我没有BBS账号/我的BBS账号等于小于3

没有MCBBS账号/MCBBS账号等级小于3会导致你下载的附件变为51KB大小的空文件，这种文件显然是不能正常当做插件使用的。对此我的建议是：创建一个MCBBS账号，然后慢慢升到3级以上。别jb再在这个问题上继续纠缠，除非你命不久矣，最后的遗愿是下载到NeigeItems.jar

我的网络环境无法连接到Github

对此我提供四种解决方案：

- 通过MCBBS下载

- 前往国外旅游，连接国外网络后登录Github下载
- 通过武林绝学连接国外网络后登录Github下载
- 欲练神功，必先自宫。自宫后口含内存条通过脑电波下载

服务端需求

NeigeItems基于BukkitAPI编写，因此你需要在拥有BukkitAPI的服务端上使用本插件。至本段wiki撰写时，以下服务端通过了用户测试：

- paper1.12.2-1.19.3
- arclight1.16.5
- spigot1.12.2
- catserver1.12.2

其中：catserver需要使用较新的版本，旧版本catserver可能导致插件无法加载。

> 信息
>
> 不在此列表的bukkit服务端不一定不能使用NeigeItems，你可以亲自尝试，下一节，我将介绍NeigeItems的安装。

插件安装

本节中，我将以paper-1.16.5-794为例，介绍NeigeItems的安装。对于bukkit服务端，插件安装过程都是一样的，此案例适用于绝大多数插件。

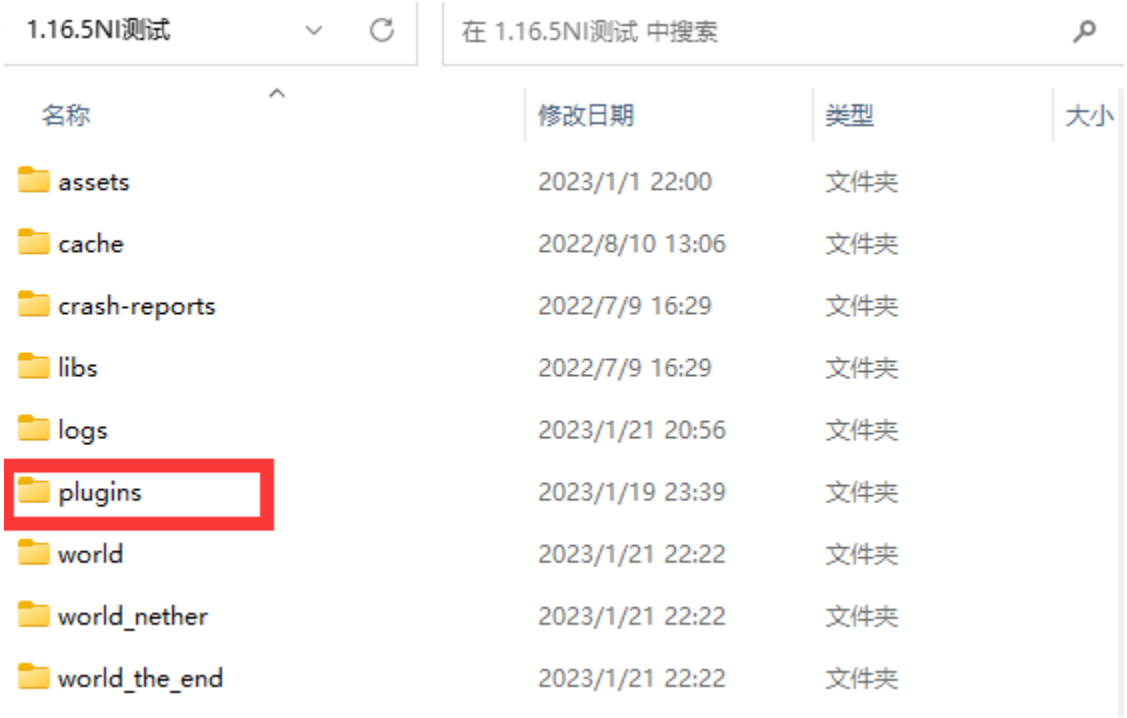
关闭服务端

安装插件前，你应该关闭服务端。所有热加载插件的行为均无法保证插件运行稳定性。热加

载插件后反馈插件无法运行的人应该被塞到马桶里溺死。同时我在此建议：卸载你服务器中的YUM，这是狗屎插件一个。

放置插件

打开你的服务端根目录，你可以看到一系列文件夹，请注意其中的“plugins”文件夹：



将你的NeigeItems.jar文件放入该文件夹。

开启服务端

开启服务端，NeigeItems理应正常加载。如果你遵照这三步安装NeigeItems遇到了插件不加载的情况，你可以加入QQ群，将logs文件夹中的日志文件发给群主Neige。Neige会根据你的服务器日志判断出现了什么错误。

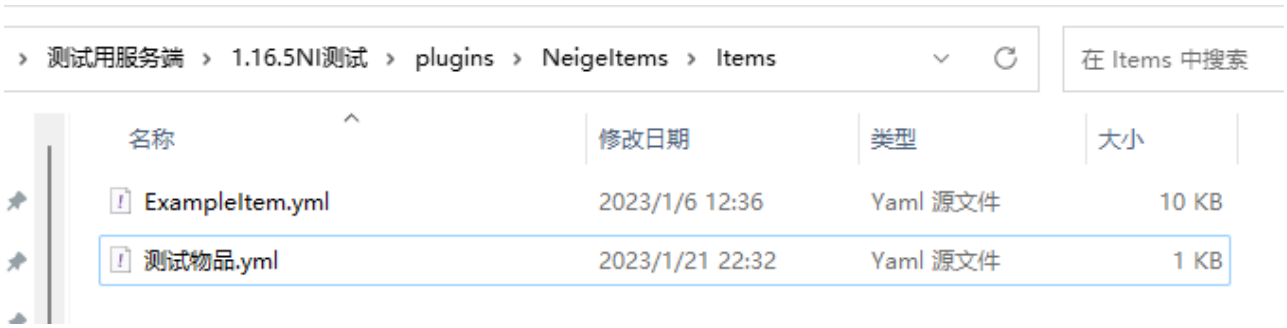
我的第一个物品

为照顾各个智商层次的人类（或者其他生物？），我将通过物品保存指令演示生成你的第一个物品配置。 现在，进入服务器，确保你是服务器OP，手持一个物品：

输入指令：`/ni save 测试物品`

我们可以看到，聊天框出现了以下文本：

现在打开服务端根目录，查看`plugins/NeigeItems/Items`文件夹：



我们可以看到，生成了`测试物品.yml`文件，因为我们没有指定物品保存路径，所以文件以物品ID命名。

相关内容可以查看物品保存指令。

现在我们打开`测试物品.yml`：

```
G: > Server > 测试用服务端 > 1.16.5NI测试 > plugins > NeigeItems > Items > ! 测试物品.yml > {} 测试物品
1 测试物品:
2   material: STONE
3
```

我们可以看到，我手中的石头成功保存了。

其中的`STONE`，就是石头对应的材质ID。你可以通过`/ni save`指令，得知所有物品的对应ID（包括mod物品）

其中的`测试物品`代表`物品ID`为测试物品，`material: STONE`代表该物品的`材质`为`石头`。

如果你想要编辑物品lore、附魔等属性，请查看物品配置

当然，你也可以在游戏内做出拥有相关属性的物品，然后通过`/ni save`将其保存，从而得知相关属性配置的编写方法，正如你得知material代表材质一样。

获取/给予物品

上一节中我们提到：其中的`测试物品`代表`物品ID`为测试物品

现在我们输入指令：`/ni get 测试物品`：

我们可以看到：

我们成功获取了`测试物品`。

你可以查看物品获取了解get、give、giveAll的用法。

类似的，`/ni give Neige 测试物品`代表给予Neige一个测试物品。

不高兴篇

不高兴篇应对不愿意看wiki的傻逼，在本节，我可能大量引用wiki内链接，告诉你这个傻逼，这个问题wiki里已经你妈的写了。

随机物品

NeigeItems是一个随机物品库，理应可以写出随机物品。

我们需要通过NeigeItems中的随机节点达成写出随机物品的目的。

随机节点分为私有节点和即时节点。由于这些东西我都写过，我直接把链接贴出来，看不懂就自杀吧。

私有节点配置

```
随机名称的铁剑:
  material: IRON_SWORD
  name: <weight-1>
  sections:
    weight-1:
      type: weight
      values:
      - 5::名字1
      - 4::名字2
      - 3::名字3
      - 2::名字4
      - 1::名字5
```

该物品的名字：

- `5/15` 概率为 `名字1`
- `4/15` 概率为 `名字2`
- `3/15` 概率为 `名字3`
- `2/15` 概率为 `名字4`
- `1/15` 概率为 `名字5`

即时节点配置

```
ExampleItem:
  material: LEATHER_HELMET
  lore:
  - '即时声明字符串节点测试: <strings::number-1_weight-1>'
```

该物品的Lore：

- `1/2` 概率为 `即时声明字符串节点测试: number-1`
- `1/2` 概率为 `即时声明字符串节点测试: weight-1`