



简介

NeigelItems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

安装

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NeigelItems

NeigelItems-自动构建

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

注意

! 不要尝试通过Plugman等热重载工具加载NeigelItems

默认配置

config.yml

```
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个
  # 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{i
  # 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
  # 对应ID物品已存在提示
  existedKey: §e[NI] §6已存在ID为 §a{itemID} §6的物品
  # 未知解析对象提示
  invalidParser: §e[NI] §6不能针对后台解析物品，请指定一个玩家
  # 错误发送者提示
  onlyPlayer: §e[NI] §6该指令仅可玩家使用
  # 保存成功提示
```



```
successSaveInfo: §e[NI] §6成功将 §f{name} §6以ID §a{itemID} §6保存至 §a{path}
# MM物品转换完毕提示

mmImportSuccessInfo: §e[NI] §6成功将所有MM物品保存至 §a{path}
# 物品到期删除提示

itemExpirationMessage: §e[NI] §6你背包中的 §f{itemName} §6已到期删除
# 物品列表内， 点击获取物品提示

clickGiveMessage: §e点击获取该物品
# 不要保存空气提示

airItem: §e[NI] §6请不要试图保存空气， 谢谢合作
# 输入无效数字提示

invalidAmount: §e[NI] §6无效数字
# 输入无效世界提示

invalidWorld: §e[NI] §6无效世界
# 输入无效坐标提示

invalidLocation: §e[NI] §6无效坐标
# 权限不足提示

insufficientPermissions: §e[NI] §6权限不足
# 物品冷却提示

itemCooldown: §e物品冷却中！ 请等待{time}秒
# 重载完毕提示

reloadedMessage: §e[NI] §6重载完毕
# 无效NBT提示

invalidNBT: §6[NI] §cNBT加载失败， 请勿在列表型NBT中混用键值对， 数字及字符串
# MM生物穿戴物品失败提示

equipFailed: §e[NI] §6在尝试给ID为 §f{mobID}§6 的MM怪物穿戴ID为 §f{itemID}§6
# 错误物品提示

invalidItem: '§6[NI] §c物品加载失败， 物品可能缺损数据， 物品ID: §6{itemID}'
# 给予失败提示

failureInfo: '§e[NI] §6物品给予失败， 可能原因： 物品未配置材质/玩家已下线'
# 缺少前置插件提示

invalidPlugin: '§e[NI] §6未发现前置插件： {plugin}'
# 位置物品材质提示

invalidMaterial: '§e[NI] §6物品 {itemID} 使用了未知的材质 {material}'
# 掉落物归属提示信息

invalidOwnerMessage: §6无法拾取该物品， 该物品的拥有者是 §f{name}
# 物品包掉落提示信息

dropPackSuccessInfo: §e[NI] §6成功在 §a{world} §6的 §a{x},{y},{z} §6掉落了
# 指令帮助信息

Help:
  prefix: |-
    §6=====§eNeigeItems§6=====
    §6=====[]为必填， ()为选填=====
  suffix: §6=====<< §e{prev} §f{current}§e/§f{total} §e{next} §
  amount: 10
```

format: "{command} §7> {description}"

prev: 上一页

next: 下一页

commands:

action:

command: §e/ni §faction [玩家ID] [动作内容]

description: 执行NI物品动作

edithand:

command: §e/ni §fedithand [玩家ID] [物品编辑函数ID] [函数内容]

description: 通过对应编辑函数编辑主手物品

editoffhand:

command: §e/ni §feditoffhand [玩家ID] [物品编辑函数ID] [函数内容]

description: 通过对应编辑函数编辑副手物品

editslot:

command: §e/ni §feditslot [玩家ID] [物品编辑函数ID] [函数内容]

description: 通过对应编辑函数编辑对应槽位物品

list:

command: §e/ni §flist (页码)

description: 查看所有NI物品

get:

command: §e/ni §fget [物品ID] (数量) (是否反复随机) (指向数据)

description: 根据ID获取NI物品

give:

command: §e/ni §fgive [玩家ID] [物品ID] (数量) (是否反复随机) (指向数据)

description: 根据ID给予NI物品

givePack:

command: §e/ni §fgivePack [玩家ID] [物品包ID] (数量)

description: 根据ID给予NI物品包

giveAll:

command: §e/ni §fgiveAll [物品ID] (数量) (是否反复随机) (指向数据)

description: 根据ID给予所有人NI物品

drop:

command: §e/ni §fdrop [物品ID] [数量] [世界名] [X坐标] [Y坐标] [Z坐标] [Z轴偏转角度]

description: 于指定位置掉落NI物品

dropPack:

command: §e/ni §fdropPack [物品包ID] [数量] [世界名] [X坐标] [Y坐标] [Z坐标] [Z轴偏转角度]

description: 于指定位置掉落NI物品包

save:

command: §e/ni §fsave [物品ID] (保存路径)

description: 将手中物品以对应ID保存至对应路径

cover:

command: §e/ni §fcover [物品ID] (保存路径)

description: 将手中物品以对应ID覆盖至对应路径

mm load:





```
command: §e/ni §fmm load [物品ID] (保存路径)
description: 将对应ID的MM物品保存为NI物品

mm cover:
command: §e/ni §fmm cover [物品ID] (保存路径)
description: 将对应ID的MM物品覆盖为NI物品

mm loadAll:
command: §e/ni §fmm loadAll (保存路径)
description: 将全部MM物品转化为NI物品

mm get:
command: §e/ni §fmm get [物品ID] (数量)
description: 根据ID获取MM物品

mm give:
command: §e/ni §fmm give [玩家ID] [物品ID] (数量)
description: 根据ID给予MM物品

mm giveAll:
command: §e/ni §fmm giveAll [物品ID] (数量)
description: 根据ID给予所有人MM物品

reload:
command: §e/ni §freload
description: 重新加载NI物品

help:
command: §e/ni §fhhelp
description: 查看帮助信息

# 物品列表格式
ItemList:
Prefix: §6=====§eNeigeItems§6=====
Suffix: §6=====<< §e{prev} §f{current}§e/§f{total} §e{next} §6>>=====
ItemAmount: 10
ItemFormat: §6{index}. §a{ID} §6- §f{name}
Prev: 上一页
Next: 下一页

# 物品拥有者提示信息显示方式
ItemOwner:
# 通过/ni get及/ni give获取物品时, 移除物品上的拥有者标签
removeNBTWhenGive: false
# actionBar / message
messageType: actionBar

# 掉落物颜色实现方式(protocol对应protocollib发包, vanilla为原版实现)
ItemColor:
# protocol / vanilla
type: protocol

ItemAction:
# 是否将1.6以下版本的ItemAction配置转换为1.7版本格式
upgrade: true
```

```
# 连击间隔(ms)
comboInterval: 500
```



GlobalSections/ExampleSection.yml

```
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
    - 1::第二行
global-js-1:
  # JavaScript节点
```

```
type: js
# 脚本路径
path: ExampleScript.js::main
```



Items/ExampleItem.yml

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
enchancements:
  ARROW_DAMAGE: 1
```



```
ARROW_KNOCKBACK: 1
# 物品隐藏标识
hideflags:
- HIDE_ATTRIBUTES
- HIDE_DESTROYS
# 物品颜色(适用于药水/皮革装备)
color: 65535
# 额外选项
options:
  charge: 10
  color: GOLD
# 物品NBT
nbt:
  # NBT中也可以随机调用节点
  <strings::文本1_文本2_文本3_文本4>: 114514
  # 可以在NBT中编辑物品的原版属性
  AttributeModifiers:
    - Amount: 10
      AttributeName: minecraft:generic.max_health
      Operation: 0
      UUID:
        - 0
        - 31453
        - 0
        - 59664
      Name: generic.maxHealth
# 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
sections:
  simple-1: <strings::text1_text2_text3>
  strings-1:
    type: strings
    values:
      - 测试文本1
      - 测试文本2
  number-1:
    type: number
    min: 1
    max: 2
```




```
      fixed: 3
calculation-1:
  type: calculation
  formula: 1+2+3<number-1>+<number-1>
  min: 1
  max: 100
  fixed: 3
weight-1:
  type: weight
  values:
    - 5::第一行
    - 1::第二行
js-1:
  type: js
  path: ExampleScript.js::main
papiString-1:
  type: strings
  values:
    - "player_"
papiString-2:
  type: strings
  values:
    - "name"
ExampleItem2:
  material: STONE
ExampleItem3:
  material: STONE
ExampleItem4:
  material: STONE
  name: "&f%neigeitems_nbt_NeigeItems.id%"
  lore:
    - '&f物品使用次数: %neigeitems_charge%/%neigeitems_maxCharge%'
  options:
    charge: 10

# Check节点测试
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
      actions:
        - "tell: 你得到了名为 test2 的物品"
      # value不为test2
    deny:
      # value为test3
      condition: value == "test3"
      # 通知玩家
      actions:
        - "tell: 你得到了名为 test3 的物品"
```

WhenTest:

```
material: STONE
name: <test> - <when>
sections:
  test: <number::0_100>
  when:
    type: when
    value: <test>
    conditions:
      - condition: value < 10
        result: E
      - condition: value < 30
        result: D
      - condition: value < 50
```



```
        result: C
    - condition: value < 70
        result: B
    - condition: value < 90
        result: A
    - condition: value <= 100
        result: S
```

ComboTest:

```
    material: DIAMOND
```

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"temp

templateItem3:

```
    inherit:
    - template1
    - template2
    name: §f物品继承测试
    sections:
        damage: 100
```



```
inheritSectionTest:
  material: STONE
  lore:
    - <templateTest>
    - <inheritTest>
    - <inherit::templateTest>
  sections:
    templateTest: <strings::text1_text2_text3>
    inheritTest:
      type: inherit
      template: templateTest
actionTest:
  material: STONE
  name: <test>
  nbt:
    test1: "666"
    test2:
      test3: "777"
    test4:
      - "888"
      - "999"
  sections:
    test: "yeah"
customSection:
  material: STONE
  lore:
    - '自定义节点测试: <test-1>'
    - '自定义节点测试: <test::test_test_test>'
  sections:
    test-1:
      type: test
      values:
        - test
        - test
        - test
        - test
eatTest:
  material: APPLE
eatTest2:
  material: APPLE
  options:
    charge: 10
dropTest:
```



```
material: STONE
dropTest2:
  material: STONE
  options:
    charge: 3
ownerTest:
  material: STONE
  name: 你捡我啊
  options:
    # 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
    # 你可以在此处填写"%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)
    owner: Neige
CustomAction:
  all:
    - "test"

# 物品时限测试
itemTimeTest:
  material: STONE
  name: 限时物品-到期时间-<js::ItemTime.js::main_<itemtime>>
  options:
    itemtime: <itemtime>
  sections:
    itemtime: 60

# join节点测试
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
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
    # 分隔符(默认为", ")
    separator: "-"
```

JoinTest3:

```
material: STONE
lore:
  # 结果: <1, 2, 3, 4, 5>
  - 'join节点: <test>'
sections:
```

```
  test:
    type: join
    list:
      - 1
      - 2
      - 3
      - 4
      - 5
    # 前缀
    prefix: "<"
    # 后缀
    postfix: ">"
```

JoinTest4:

```
material: STONE
lore:
  # 结果: 1, 2, 3
  - 'join节点: <test>'
sections:
```

```
  test:
    type: join
    list:
      - 1
      - 2
```



```
- 3
- 4
- 5
# 限制长度
limit: 3
JoinTest5:
  material: STONE
  lore:
    # 结果: 1, 2, 3, ...
    - 'join节点: <test>'
  sections:
    test:
      type: join
      list:
        - 1
        - 2
        - 3
        - 4
        - 5
      limit: 3
      # 超过长度的部分用该符号代替
      truncated: "..."/>
JoinTest6:
  material: STONE
  lore:
    # 结果: 2, 3, 4, 5, 6
    - 'join节点: <test>'
  sections:
    test:
      type: join
      list:
        - 1
        - 2
        - 3
        - 4
        - 5
      # 对列表中的每个元素进行一定操作
      # this.it代表当前元素
      # this.index代表当前序号(0代表第一个, 1代表第二个, 以此类推)
      # this.player代表玩家
      # this.vars(String string)用于解析节点
      # List<String> this.list代表节点中的list
      transform: |-
        // 尝试将当前元素转换为整数, 并加一, 然后保留整数
```

```
return (parseInt(this.it) + 1).toFixed(0)
```



JoinTest7:

```
material: STONE
lore:
  # 等同于:
  # - 第一行
  # - 第二行
  # - 第三行
  #
  # 这个节点应该单独占据一行
  # 不要在这行写其他文本(比如'join节点: <test>')
  # 具体请自行测试
  - '<test>'
sections:
  test:
    type: join
    list:
      - 第一行
      - 第二行
      - 第三行
    # 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: ''
```

GaussianTest:

```
material: STONE
lore:
  - '随机数: <test>'
  # <gaussian::基础数值_浮动单位_浮动范围上限_取整位数(默认为1, 可省略)_数值下限(可
  - '随机数: <gaussian::100_0.1_0.5_1>'
sections:
  test:
    type: gaussian
    # 基础数值
    base: 100
    # 浮动单位
    spread: 0.1
    # 浮动范围上限
    maxSpread: 0.5
    # 取整位数(默认为1)
    fixed: 1
    # 数值下限
```




```
min: 0
# 数值上限
max: 10000

# 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
    - 'repeat节点: <repeat>'
  sections:
    repeat:
      type: repeat
      content: "|"
      repeat: 20
      prefix: "§4"
      # 对列表中的每个元素进行一定操作
      # this.it代表content
      # this.index代表当前序号(0代表第一个, 1代表第二个, 以此类推)
      # this.player代表玩家
      # this.vars(String string)用于解析节点
      transform: |-
        if (this.index == this.vars("<number>")) {
          return "§f" + this.it
```



```

        } else {
            return this.it
        }
    number:
        type: number
        min: 0
        max: 20
        fixed: 0
RepeatTest3:
    material: STONE
    lore:
        # 随机1-4行"&4&l<红宝石槽>"
        - '<repeat>'
    sections:
        repeat:
            type: repeat
            content: '&4&l<红宝石槽>'
            repeat: <number::1_4_0>
            # 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: ''
RepeatTest4:
    material: STONE
    lore:
        # 形似"$4$l<★>-$4$l<★>-$4$l<★>", 随机1-4个
        - '<repeat>'
    sections:
        repeat:
            type: repeat
            content: '$4$l<★>'
            repeat: <number::1_4_0>
            separator: "-"

```

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"
  # 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
  #
  #           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
# 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"

ExampleItem3:
# 左键触发
left:
# 冷却时间(单位是ms)
cooldown: 3000
# 冷却组，同一冷却组的物品共享冷却时间
group: test2
# 消耗选项
consume:
# 物品消耗条件
condition: perm("item.ExampleItem3")
# 每次消耗物品数量
amount: 1
# 不满足条件时执行的动作
deny:
  - "tell: 你没有&e item.ExampleItem3 &f权限"
# 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 ExampleItem3 &f物品"

ExampleItem4:
all:
consume:
amount: 1
sync:
  - 'console: say He''s name is %player_name%'
  - 'command: say My name is %player_name%'

ActionTest:
all:
sync:
# 检测test1这条NBT的值是否等于"666"
condition: parseItem("<nbt::test1>") == "666"
actions:
  - '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>'

EatTest1:
```



```
eat:
  sync:
    - 'giveFood: 5'
    - 'giveHealth: 5'
EatTest2:
  eat:
    consume:
      amount: 1
    sync:
      - 'giveFood: 5'
      - 'giveHealth: 5'
DropTest1:
  drop:
    actions:
      - 'castSkill: SkillTest'
DropTest2:
  drop:
    consume:
      amount: 1
    actions:
      - 'castSkill: SkillTest'
CustomAction:
  all:
    sync:
      - test
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:
        # 进行对应操作
        - 'tell: &e连击 &bL'
  right:
```

sync:

```
# 在ComboTest组记录，触发了类型为right的连接
- "combo: ComboTest right"

# 检测ComboTest组是否完成了left-right连接
- condition: combo("ComboTest", ["left", "right"])
actions:
  # 进行对应操作
  - 'tell: &e连接 &bL &f+ &bR'
```

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.SectionM
  const CustomSection = Packages.pers.neige.neigeitems.section.impl.Cust
  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,
        return SectionUtils.parseSection("<number::0_1_2>", cache,
      }
      return null
    },
    /**
     * 用于即时节点解析
```



```

    * @param args List<String> 节点参数
    * @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(args, cache, player, sections) {
        return SectionUtils.parseSection("<number::0_1_2>", cache, pla
    })
// 节点注册
SectionManager.loadParser(customSection)
}

```

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.ActionMan
    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
            return true
        })
}

```



```
// 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.ItemE

    // 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
            if (itemStack.type != Material.AIR) {
                // 获取itemMeta
                const itemMeta = itemStack.itemMeta
                if (itemMeta != null) {
                    // 获取并设置lore
                    let lore = itemMeta.lore
                    if (lore == null) lore = new ArrayList()
                    lore.addAll(ChatColor.translateAlternateColorCodes('&'
                    itemMeta.lore = lore
                    // 将改动完成的itemMeta设置回去
                    itemStack.setItemMeta(itemMeta)
                    // 物品编辑都需要返回一个布尔量，判断你是否编辑成功
                    return true
                }
            }
        }
    )
    // 物品编辑都需要返回一个布尔量，判断你是否编辑成功
```



```

        return false
    }
)
}

```



ItemPacks/ExampleItemPack.yml

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

sections:

```
test:
  type: strings
  values:
    - ExampleItem 5 1
    - ExampleItem 10 1
```



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.
}
```

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



/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) > 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)



`/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`

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：[NI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) >` 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

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**

! 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**

! 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.MMIItemsPath

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

! 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.MMIItemsPath

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

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

! 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



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



reload

/ni reload > 重新加载NI物品

物品配置

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

配置

详见[默认配置](#)

编写你的物品

/ni save是万物起源

遇事不决，/ni save。如果不行，就/nl 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。

ID

所有物品都应该有一个ID，如下格式：



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

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`

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
```

```
lore:
```

- 我有lore
- 我真有lore
- 信我

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:

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



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。

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

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



有附魔的铁剑：

```
material: IRON_SWORD
enchancements:
  # 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

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

啥都看不到的铁剑：

```
material: IRON_SWORD
hideflags:
  # 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
```

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。

Neigeltems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:
  MMOITEMS_ATTACK_DAMAGE: (Double) 1000000
```

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

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

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

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
```

有关私有节点各个类型，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

配置

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:
  inherit:
    - template1
    - template2
  name: §f物品继承测试
  sections:
    damage: 100
```

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。

```
inherit: template1
```

代表这个物品将继承"template1"的全部内容

```
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

```
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。

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变量与即时节点）



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

以默认指令配置为例

```
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)
 - 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
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

冷却时间

冷却时间的单位是毫秒(ms), 支持解析动作变量及即时声明节点, 如:

```
# 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:
    cooldown: <number::5000_10000>
    actions:
      - "tell: 你好"
# 冷却时间为物品中名为 test1 的NBT的值
test3:
  left:
    cooldown: <nbt::test1>
    actions:
      - "tell: 你好"
# 冷却时间为 %player_level% 这个PAPI变量的解析值
test4:
  left:
    cooldown: <papi::player_level>
    actions:
      - "tell: 你好"
```

冷却组

冷却组默认为 **触发类型-物品ID** , 如:

```
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



```
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
      deny:
        - 'tell: 你没有 test1 权限'
    actions:
      - 'tell: 你持有 test1 权限, 成功消耗了 1 个 test1 物品'
```

其中, amount支持代入动作变量及即时声明节点, 如:

```
# 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:

```
test1:
  left:
    consume:
      amount: 2
    deny:
      - 'tell: 你真的太逊了, 连2个物品都拿不出来'
    actions:
      - 'tell: 物品消耗成功'
```

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: 所有非线性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,
#           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

全部动作支持papi变量, 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:

```
# 字符式
actions: 'tell: 你好'
```

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

```
# 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:

```
# 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)

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

发送文本

向玩家发送一条消息(不将&解析为颜色符号)

```
- '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)

```
- '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'



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:
```




```
        # 进行对应操作
        - 'tell: &e连击 &bL'

right:
  sync:
    # 在ComboTest组记录，触发了类型为right的连击
    - "combo: ComboTest right"
    # 检测ComboTest组是否完成了left-right连击
    - condition: combo("ComboTest", ["left", "right"])
    actions:
      # 进行对应操作
      - 'tell: &e连击 &bL &f+ &bR'
```

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'
```



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

! 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
```

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`

```
pers.neige.neigeitems.utils.ItemUtils
pers.neige.neigeitems.utils.SectionUtils
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?>()
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对应物品中体现为:
```

```
# test:
#   test: "666"
condition: 'getNBT("test.test") == "666"'
```

获取NBT值(getNBTTTag)

字符串记得用引号包起来

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

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

```
# 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: 'getNBTTTag("test").asString() == "666"'
```



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

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

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 && bedSpawnZ() == 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 && compassTargetZ() == 1'
```

本月日期(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'

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

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

```
condition: 'guilding(true)'
```

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)



```
# 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：

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

```
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

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.ActionMan
    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

通过 `/ni edithand``/ni editoffhand``/ni editslot` 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

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

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(解析即时声明节点)



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

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

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

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(解析即时声明节点)

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

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

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

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(解析即时声明节点)

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

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

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



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

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(解析即时声明节点)

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

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Se测试物品

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SeNeige

文本中的&将被自动替换为颜色符号\$

显示名(解析即时声明节点)

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Se测试物品1或Se测试物品2

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

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%-`

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

文本中的&将被自动替换为颜色符号§

显示名前缀(解析即时声明节点)

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



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

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

文本中的&将被自动替换为颜色符号§

显示名后缀(解析即时声明节点)

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`Se测试物品`, 函数执行后将随机变为`Se测试物品-后缀1`或`Se测试物品-后缀2`

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`"

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"

文本中的`&`将被自动替换为颜色符号`$`

替换显示名(替换一次, 解析即时声明节点)

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"

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"

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"

文本中的&将被自动替换为颜色符号§

替换显示名(替换全部, 解析即时声明节点)



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"

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"

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_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"

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, 解析即时声明节点)

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"

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"

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"

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, 解析即时声明节点)

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"

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

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

文本中的&将被自动替换为颜色符号\$, \n代表换行

添加Lore(解析其中的即时声明节点)

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

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

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

文本中的&将被自动替换为颜色符号\$, \n代表换行

设置Lore(解析即时声明节点)

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

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

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



XXXXXXX

拥有者: Neige

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, 解析即时声明节点)

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

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

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



XXXXXXX

拥有者: Neige

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, 解析即时声明节点)

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

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

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

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, 解析即时声明节点)

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

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

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

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, 解析即时声明节点)

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

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

! 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

! 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(解析即时声明节点)



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

! 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)

! 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)



! 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(解析即时声明节点)

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)

! 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)

! 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)

! 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(解析即时声明节点)

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

! 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)



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

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

适用于1.14+版本

CustomModelData(解析即时声明节点)

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

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(解析即时声明节点)

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

! 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

! 物品原有附魔将被移除

设置附魔(解析即时声明节点)

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



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

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

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(解析即时声明节点)

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

! 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)

! 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)

! 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, 解析即时声明节点)



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)

! 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(解析即时声明节点)



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(解析即时声明节点)

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(解析即时声明节点)

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

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(解析即时声明节点)

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

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`



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(解析即时声明节点)

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

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(解析即时声明节点)

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

通过.分隔NBTCCompound, 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, 解析即时声明节点)

为物品设置NBT(原有设置NBT将被覆盖)



函数ID: `setNBSection`

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: `setNBWithList`

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:

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: `setNBWithListPapi`

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(解析即时声明节点)

为物品设置NBT(原有设置NBT将被覆盖)

函数ID: `setNBWithListSection`

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:

```
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

! 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}

! 该函数将无视堆叠数量重构物品

重构物品(解析即时声明节点)

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}`

! 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

! 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)(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

! 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)(解析即时声明节点)



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

! 该函数将无视堆叠数量重构物品

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:

```
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

! 该函数将无视堆叠数量刷新物品



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`

! 该函数将无视堆叠数量刷新物品

刷新部分节点(解析即时声明节点)

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`

! 该函数将无视堆叠数量刷新物品

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)(解析即时声明节点)

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

自定义函数

自定义函数需要一定的 javascript 和 java 基础。



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.ItemE

    // 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
            if (itemStack.type != Material.AIR) {
                // 获取itemMeta
                const itemMeta = itemStack.itemMeta
                if (itemMeta != null) {
                    // 获取并设置lore
                    let lore = itemMeta.lore
                    if (itemMeta == null) lore = new ArrayList()
                    lore.addAll(ChatColor.translateAlternateColorCodes('&'
                    itemMeta.lore = lore
                    // 将改动完成的itemMeta设置回去
                    itemStack.setItemMeta(itemMeta)
                    // 物品编辑都需要返回一个布尔量，判断你是否编辑成功
                    return true
                }
            }
        }
    )
}
```

```
        // 物品编辑都需要返回一个布尔量，判断你是否编辑成功
        return false
    }
}
)
```



额外选项

以默认配置为例:

```
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

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

charge 该物品可使用的次数 (可触发物品动作的次数)

物品光效

```
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)

! 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)

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.
}
```

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被发包装替换

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_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

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 可以直接写数量

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



横向偏移表示物品向四周弹射的力度

纵向偏移表示物品向空中弹射的力度

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

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

以默认配置为例

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) (指向类)
- 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

```
sections:
  test:
    type: strings
    values:
      - ExampleItem 5 1
      - ExampleItem 10 1
```

具体调用指令如下

[givePack](#)

[dropPack](#)

全局/私有节点

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

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

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

节点ID:

```
type: gaussian
base: 100
spread: 0.1
maxSpread: 0.5
```

```
fixed: 1
min: 0
max: 10000
```



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
- **max** 结果的最大值
- **fixed** 小数保留位数

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
- **max** 结果的最大值
- **fixed** 小数保留位数



权重节点

节点ID:

```
type: weight
values:
- 5::第一行
- 1::第二行
```

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"

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



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"
```

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)

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:

```
# 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
      # 分隔符(默认为", ")
      separator: "-"
# 帮助理解prefix及postfix
JoinTest3:
  material: STONE
  lore:
    # 结果: <1, 2, 3, 4, 5>
    - 'join节点: <test>'
  sections:
    test:
      type: join
      list:
        - 1
        - 2
        - 3
        - 4
        - 5
      # 前缀
```



```
    prefix: "<"
    # 后缀
    postfix: ">"
# 帮助理解limit
JoinTest4:
    material: STONE
    lore:
        # 结果: 1, 2, 3
        - 'join节点: <test>'
    sections:
        test:
            type: join
            list:
                - 1
                - 2
                - 3
                - 4
                - 5
            # 限制长度
            limit: 3
# 帮助理解truncated
JoinTest5:
    material: STONE
    lore:
        # 结果: 1, 2, 3, ...
        - 'join节点: <test>'
    sections:
        test:
            type: join
            list:
                - 1
                - 2
                - 3
                - 4
                - 5
            limit: 3
            # 超过长度的部分用该符号代替
            truncated: "... "
# 帮助理解transform
JoinTest6:
    material: STONE
    lore:
        # 结果: 2, 3, 4, 5, 6
        - 'join节点: <test>'
```



```
sections:
  test:
    type: join
    list:
      - 1
      - 2
      - 3
      - 4
      - 5

    # 对列表中的每个元素进行一定操作
    # this.it代表当前元素
    # this.index代表当前序号(0代表第一个, 1代表第二个, 以此类推)
    # this.player代表玩家
    # this.vars(String string)用于解析节点
    # List<String> this.list代表节点中的list
    transform: |-
      // 尝试将当前元素转换为整数, 并加一, 然后保留整数
      return (parseInt(this.it) + 1).toFixed(0)

# 利用join节点插入多行lore
JoinTest7:
  material: STONE
  lore:
    # 等同于:
    # - 第一行
    # - 第二行
    # - 第三行
    #
    # 这个节点应该单独占据一行
    # 不要在这行写其他文本(比如'join节点: <test>')
    # 具体请自行测试
    - '<test>'

sections:
  test:
    type: join
    list:
      - 第一行
      - 第二行
      - 第三行

    # 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: ''
```



Repeat节点

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:

```
# 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
        - 'repeat节点: <repeat>'
sections:
    repeat:
        type: repeat
        content: "|"
        repeat: 20
        prefix: "&4"
        # 对列表中的每个元素进行一定操作
        # this.it代表content
        # this.index代表当前序号(0代表第一个，1代表第二个，以此类推)
        # this.player代表玩家
        # this.vars(String string)用于解析节点
        transform: |-
            if (this.index == this.vars("<number>")) {
                return "&f" + this.it
            } else {
                return this.it
            }
    number:
        type: number
        min: 0
        max: 20
        fixed: 0
RepeatTest3:
    material: STONE
    lore:
        # 随机1-4行"&4&l<红宝石槽>"
        - '<repeat>'
```



```
sections:
  repeat:
    type: repeat
    content: '&4&l<红宝石槽>'
    repeat: <number::1_4_0>
    # 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: ''
RepeatTest4:
  material: STONE
  lore:
    # 形似"$4$l<★>-$4$l<★>-$4$l<★>", 随机1-4个
    - '<repeat>'
  sections:
    repeat:
      type: repeat
      content: '$4$l<★>'
      repeat: <number::1_4_0>
      separator: "-"
```

渐变色节点

```
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
```

- **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



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：

```
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
actions:
- 'tell: 23333'
```

- **value** 待检查内容
- **actions** 执行动作

value将作为变量传入condition供你判断, 示例配置如下:

```
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
actions:
- "tell: 你得到了名为 test2 的物品"
# value不为test2
deny:
  # value为test3
  condition: value == "test3"
  # 通知玩家
  actions:
  - "tell: 你得到了名为 test3 的物品"

```

When节点

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

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:

```

WhenTest:
  material: STONE
  name: <test> - <when>
  sections:

```



```
test: <number::0_100>
when:
  type: when
  value: <test>
  conditions:
    - condition: value < 10
      result: E
    - condition: value < 30
      result: D
    - condition: value < 50
      result: C
    - condition: value < 70
      result: B
    - condition: value < 90
      result: A
    - condition: value <= 100
      result: S
```

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(V1.6.0添加)

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
- 参数3 公式结果最小值
- 参数4 公式结果最大值

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
- 参数3 公式结果最小值
- 参数4 公式结果最大值

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

!

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>
```

- 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.SectionM
    const CustomSection = Packages.pers.neige.neigeitems.section.impl.Cust
    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,
                return SectionUtils.parseSection("<number::0_1_2>", cache,
            }
            return null
        },
    ),
```



```
/**
 * 用于即时节点解析
 * @param args List<String> 节点参数
 * @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(args, cache, player, sections) {
    return SectionUtils.parseSection("<number::0_1_2>", cache, pla
    })
// 节点注册
SectionManager.loadParser(customSection)
}
```

16进制颜色

<#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

! 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:

```
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

```
globalsections:
  - stringTest
```

引用该节点

JavaScript

对象与函数

NeigelItems 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

NI物品掉落



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)
```

```
- itemId 1-5 0.5
```

```
test2:
```

```
Type: ZOMBIE
```

```
Health: 1
```

```
NeigeItems:
```

```
Drops:
```

```
# 50%掉落1个ID为"itemId"的NI物品(或MM物品、EasyItem物品)
```

```
- itemId 1 0.5
```

```
test3:
```

```
Type: ZOMBIE
```

```
Health: 1
```

```
NeigeItems:
```

```
Drops:
```

```
# 掉落5个ID为"itemId"的NI物品(或MM物品、EasyItem物品)
```

```
- itemId 5
```

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怪物配置掉落组

```
NeigeItems:
  DropPacks:
    - 物品包ID
    - 物品包ID
    - 物品包ID
```

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
```

多彩掉落

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

纵向偏移表示物品向空中弹射的力度

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

下面我写几个MM怪物配置实例：

```
test1:
  Type: ZOMBIE
  Health: 1
  NeigeItems:
    FancyDrop:
      offset:
        x: 0.1
        y: 1
      angle:
        # 转一圈弹出去
        type: round
```

```
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
```

NI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:
    Equipment:
```



```
# 头部50%几率穿戴ID为"Helmet1"的NI物品(或MM物品、EasyItem物品)
- 'Helmet: Helmet1 0.5'

# 胸部100%几率穿戴ID为"Chestplate1"的NI物品(或MM物品、EasyItem物品)
- 'Chestplate: Chestplate1'
- 'Leggings: Leggings1 0.5'
- 'Boots: Boots1 0.5'
- 'MainHand: MainHand1 0.5'
- 'OffHand: OffHand1 0.5'
```

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



```
test1:
  Type: ZOMBIE
  Health: 1
  NeigeItems:
    Equipment:
      - 'Helmet: Helmet1 0.5'
      - 'Chestplate: Chestplate1'
      - 'Leggings: Leggings1 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

通过节点插入多行lore

方法1



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

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

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

！ 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

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，会直接变成回车（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上面那个屁问题。这种狗东西你怎么写出来的，脑测吗？

回答：请善用debug。将plugin/NeigelItems/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

错误示范见下图

生成随机强度条

方法1

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

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



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

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: ''
```

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NeigelItems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NeigelItems.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NeigelItems-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NeigelItems-1.6.5.jar文件。别寄吧问我为什么没找到叫“NeigelItems-版本.jar”的文件，只看到了一个什么“NeigelItems-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NeigelItems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“NeigelItems-1.6.5.jar”。别jb用那个“NeigelItems-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，NeigelItems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“NeigelItems-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NeigelItems.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

NeigelItems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。

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

插件安装

本节中，我将以paper-1.16.5-794为例，介绍NeigelItems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NeigelItems.jar文件放入该文件夹。

开启服务端

开启服务端，NeigelItems理应正常加载。如果你遵照这三步安装NeigelItems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`：

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

NeigelItems是一个随机物品库，理应可以写出随机物品。

我们需要通过NeigelItems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

