# Task使用规范

## 常规API介绍：

常规情况下，你使用且仅需使用“saltsheep.ssl.api.Task”类

**创建新的Task：**

Task.run(()=>void 执行任务)

Task.delay(int 延迟时间)

Task.repeat(()=>Boolean 执行任务, int 事件间隔)

Task.repeatUnique(IEntity/Entity 实体, String 任务ID,

()=>Boolean 执行任务, int 事件间隔)

注：repeat和repeatUnique的执行任务必须返回布尔值，且返回true时Task将会提前结束，不再继续重复执行任务。调用repeatUnique将会覆盖掉同实体同ID的任务

**连接Task：**

对于两个Task对象，task1先运行，task2后运行，你应该：

task1.append(task2)

该方法返回task1，这是为了便于开发，采用链式调用的设计，

对于任何Task对象使用append方法，新Task会追加到Task链的尾部

这意味着你可以链式调用（下为链式调用例子）

task1.append(task2).append(task3)

***Warning:***

在SSL1.19.4-，该方法返回task2

由于这个特性，你应该在创建task1保存它，而不是获取方法的返回值

否则你仅仅只会执行最后一个任务

***语法糖(1.22.2+)：***

你可以使用append(()=>void)取代append(Task.run(()=>void))

其他由SheepScriptLib提供的Task也同样适用于这个语法糖

这个糖由Task的append方法重载实现

**开始运行Task：**

对于一个Task对象，它可以使用start方法开始运行，

当该任务完成，将会继续完成其后接的Task

如下：

task.start();

假设有三个Task对象，task1后接task2但不运行，task2后接task3且运行

该假设情况的代码如下：

task1.append(task2).append(task3);

task2.start();

**为Task添加中断任务：**

当存档（或者服务器）关闭时，Task链的运行会被中断，

如果你想要进行一些末尾处理，你可用该方法，为单个Task追加一个中断运行：

task.setStopRun(function()->void 执行任务)

这个方法返回task自己，便于链式调用

值得一提的是，当Task链被中断，将会从最后一个尚未运行结束抛出的Task开始，

按顺序执行其与其后每一个Task的中断任务。

*这其中涉及到了Task的一个实现：*

*你可以查看源码，当Task的invoke方法返回true，Handler的Task指针会后挪，*

*Handler会处理接下来的Task，*

*如果invoke方法全部返回true，整个Task链会在一刻内执行完毕。*

*这意味着，中断任务只在那些 前或己 有暂缓功能（invoke返回false）的Task*

*从这个Task往后开始执行中断任务。*

**提前终止Task：**

一般情况下，Task只会在执行完毕，或者服务器关闭时停止，但从mod版本1.24.1起，SSL提供了一个新的方法。

task.stop()

这会触发中断任务

## 非常规API介绍：

task.next，通过调用这个字段，你可以直接获取和设置Task对象的下一个Task

task.init()，通过调用这个方法，你可以手动为单个Task对象初始化

task.invoke()，通过这个方法，你可以手动调用单个Task对象的执行（不建议）

Task.custom(Runnable init, Callable<Boolean> invoke, Callable<Task> clone)，高度自定义Task，具体查阅源码

## 注意的事情

1. 任何一个Task对象，其后接的Task不能够为其前面的Task，这会导致死循环
2. 你可以复用一条Task链，但如果其中有delay，你应该等上一条链执行完毕再进行复用，因为Task链开始运行时，会对链上所有Task进行初始化，而delay的实现基础是计时器，在初始化时会清空计时器。
3. Task的run和stopRun都是直接在MC主线程运行的，由于没有实体参数传入run和stopRun，任何报错信息都会输送到日志，你应该时刻查看日志——当然，你也可以使用try-catch包裹手动报错——如果你没用手动包裹报错，那么在报错信息输送到日志后，会默认跳过这一个出错的Task而执行下一个，对于stopRun则是跳过这个stopRun到下一个。

## 特性利用提示

①通过修改next和利用init，你可以做到临时变更后续Task链的操作，这会很有用

②你可以将Task链的每一个Task按顺序添加入数组，这样便于后继读取

③你可以利用中断任务记录最后一个执行到的Task，并借此在重新开始Task链时找到终止位置开始。

## 相对于SheepAI

优势：

1. 更加简易，轻量级
2. 在决定执行起点时非常方便

劣势：

1. 灵活性不足
2. 报错方式仅限于日志
3. 需要手动管理任务的启动

## 完整例子：

/\*\*

 \* @Author 咸鱼羊

 \* 该脚本会出现的错误以及原因：

 \*  中断任务概率不能成功保存storeddata

 \*  原因未查明，个人猜测是因为中断任务执行的时间点是ServerStopping，

 \*  而这时候实体可能已经完成保存，新添加的storeddata没有被保存

 \* 为此，此脚本仅供参考使用，不推荐使用该方式在中断任务保存数据

 \*/

var Task = Java.type("saltsheep.ssl.api.Task");

var System = Java.type("java.lang.System");//\*用于向日志输出消息，确认stopRun状态

var taskChain;

var isRunning = false;

function initTasks(npc){

    taskChain = new Array();

    taskChain.push(Task.run(function(){

        npc.say("开始启动！");

        isRunning = true;

    }));//0

    taskChain.push(Task.delay(10));

    taskChain.push(getTask(npc,1));//2

    taskChain.push(Task.delay(10));

    taskChain.push(getTask(npc,2));//4

    taskChain.push(Task.delay(10));

    taskChain.push(getTask(npc,3));//6

    taskChain.push(Task.delay(10));

    taskChain.push(getTask(npc,4));//8

    taskChain.push(Task.delay(10));

    taskChain.push(getTask(npc,5));//10

    taskChain.push(Task.delay(10));

    taskChain.push(getTask(npc,6));//12

    taskChain.push(Task.delay(10));

    taskChain.push(getTask(npc,7));//14

    taskChain.push(Task.delay(10));

    taskChain.push(getTask(npc,8));//16

    taskChain.push(Task.delay(10));

    taskChain.push(getTask(npc,9));//18

    taskChain.push(Task.delay(10));

    taskChain.push(getTask(npc,10));//20

    taskChain.push(Task.run(function(){

        npc.say("四字游戏，启动！");

        isRunning = false;

    }));

    var endTask = taskChain[0];

    for(var i=1;i<taskChain.length;i++){

        endTask = endTask.append(taskChain[i]);

    }

}

function getTask(npc,process){

    return Task.run(function(){

        npc.say("进度"+process+"0/100");

    }).setStopRun(function(){

        if(!npc.getStoreddata().has("Task process")){

            System.out.println("stopRun:"+process);

            npc.getStoreddata().put("Task process",process\*2);

        }

    })

}

function init(e){

    initTasks(e.npc);

}

function interact(e){

    if(!isRunning){

        var hasProcess = e.npc.getStoreddata().has("Task process");

        var process = e.npc.getStoreddata().get("Task process");

        e.npc.say(hasProcess+";"+process);

        if(!hasProcess||process==null){

            taskChain[0].start();

        }else{

            e.npc.getStoreddata().remove("Task process");

            var startTask = taskChain[process];

            e.npc.say("等一下，我这里好像有你的启动记录，让我们读取记录吧！");

            isRunning = true;

            startTask.start();

        }

    }

}

//\*用于重置storeddata

function damaged(e){

    e.npc.getStoreddata().remove("Task process");

}