# StateMachine使用规范

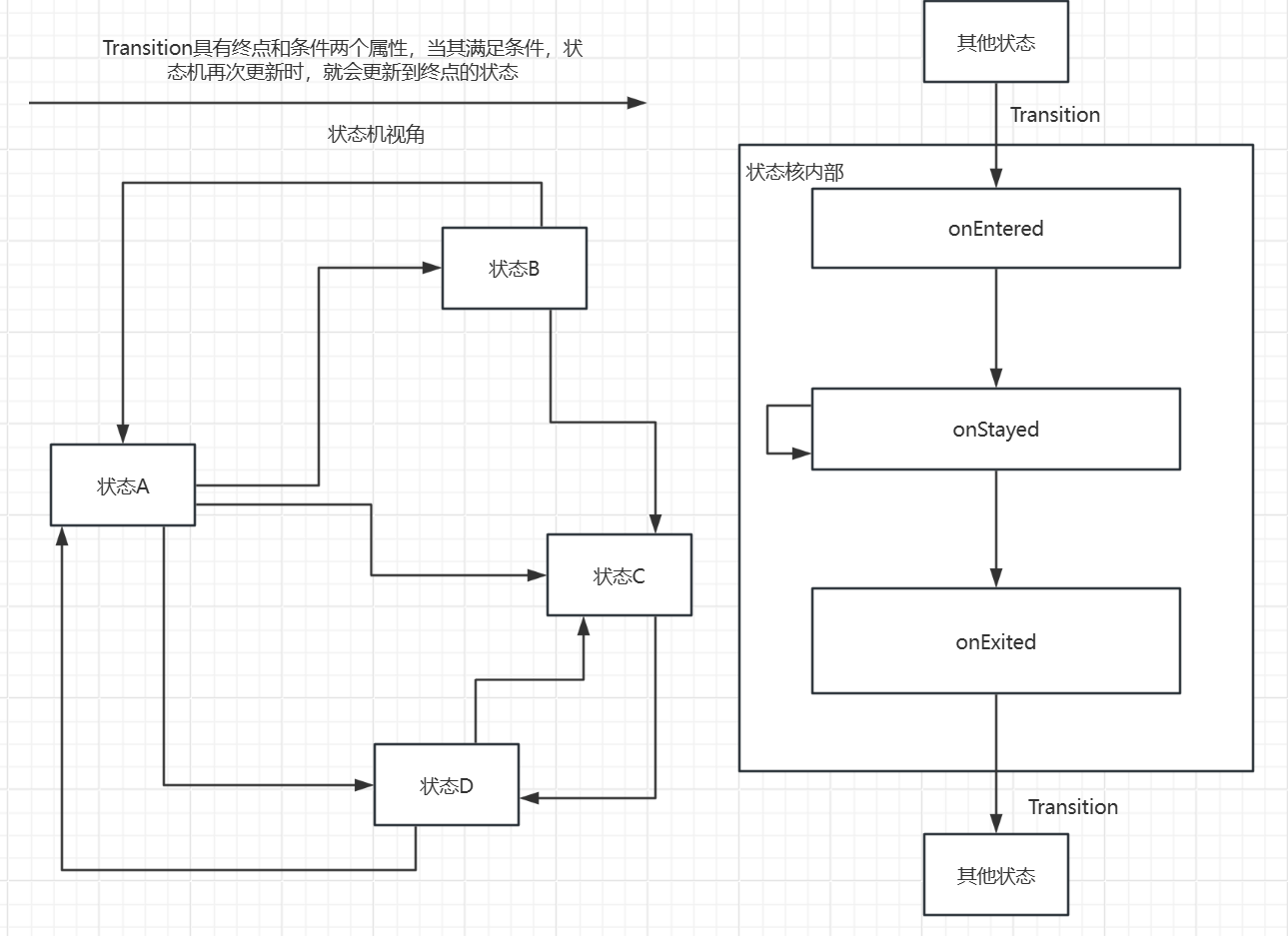
## 功能介绍：

显而易见地，这是个状态机，标准的有限状态机

此状态机实现有三个成分：

负责管理状态的状态机（StateMachine）

状态处理核（State）

状态转换箭头（Transition） 

## 如何使用：

导入saltsheep.ssl.api.StateMachine类，并实例化

实例方法有

getState(String stateName),

addState(String stateName),

addState(String stateName, State state),

addState(String stateName, Task task),

delState(String stateName),

setDefault(String defaultState),

setDefault(String defaultState. State state),

update()

addState和getState返回State，delState返回状态机本身，setDefault无返回值

**基于Task的快速State实现包装：**

addState(String stateName, Task task)

如果使用这一个语法糖，添加的状态将会在进入状态时启动Task，退出状态时会尝试提前终止Task

**对于State的用法：**

state.onEntered = function(){}

state.onStayed = function(){}

state.onExited = function(){}

onEntered, onStayed, onExited 为Runnable类型的字段，应设置其为无参无返回值的函数，也可设置为空（null），为空时不起效

state.addTransition(String destination, Callable<Boolean> condition)

添加状态转换箭头，当condition返回true时，状态就会切换至destination

addTransition(String destination, Callable<Boolean> condition, long duration, IGetter<Long> tickGetter) 添加的状态转换箭头带有计时功能，时间达标才能转换

state.clearTransitions() 清除所有状态转换箭头

**对于Transition的扩展：**

StateMachine.Transition提供了三个默认实现为空的方法

onStateEntered(), onStateStayed(), onStateExited()

你可以继承Transition类，重写方法，从而实现扩展。

**TransitionAnytime（从SSL 1.25.0 起）：**

StateMachine新增了三个方法：

addAnytimeTransition(String destination, Callable<Boolean> condition)

addAnytimeTransition(String destination, Callable<Boolean> condition, long duration, IGetter<Long> tickGetter)

setAnytimeEnabled (boolean anytimeEnabled)

功能是添加基于整个状态机的Transition，当anytimeEnabled为true时，状态机每次更新会优先考虑anytimeTransition，之后才会考虑当前状态的transition

## 实例：

// 这段代码受到攻击时进入another状态，对话时进入default状态

var StateMachine = Packages.saltsheep.ssl.api.StateMachine;

var Task = Packages.saltsheep.ssl.api.Task;

var machine;

var isDamagedRecently = false;

function init(e) {

    machine = new StateMachine();

    var taskStart = Task.run(function() {

        e.npc.say("TaskStart");

    });

    taskStart.append(50);

    var taskEnd = Task.run(function() {

        e.npc.say("TaskEnd");

    });

    taskEnd.setStopRun(function() {

        e.npc.say("TaskStop");

    });

    taskStart.append(taskEnd);

    machine.addState("default", taskStart);

    var defaultOne = machine.setDefault("default");

    var defaultOnesOldEntered = defaultOne.onEntered;

    defaultOne.onEntered = function() {

        e.npc.say("Enter");

        defaultOnesOldEntered.run();

    }

    defaultOne.onStayed = function() {

        e.npc.say("Staying");

    }

    var defaultOnesOldExited = defaultOne.onExited;

    defaultOne.onExited = function() {

        e.npc.say("Exit");

        defaultOnesOldExited.run();

    }

    defaultOne.addTransition("another", function() {

        return isDamagedRecently;

    });

    var anotherOne = machine.addState("another");

    anotherOne.onEntered = function() {

        e.npc.say("another enter");

    }

    anotherOne.onExited = function() {

        e.npc.say("another exit");

    }

    anotherOne.onStayed = function() {

        e.npc.say("another stay");

    }

    anotherOne.addTransition("default", function() {

        return !isDamagedRecently;

    });

}

function interact(e){

    isDamagedRecently = false;

    machine.update();

}

function damaged(e){

    isDamagedRecently = true;

    machine.update();

}