# BothSideScript规范

## 风险警告！

BothSideScript(下文简称BSS)在客户端和服务端各自独立运行，并且可以起到近似于模组开发的效果，

这可能会对一些 *模组黑名单/客户端模组检测* 的功能带来绕过的可行性。

这种风险目前是未知的，潜在的，可能的，如果您对此介意，我有如下两个建议：

·不安装SSL模组——一旦安装SSL模组，没有100%的可能性避开BSS作弊

·安装模组版本在1.18及以下的SSL模组——这些版本没有提供BSS

！但也没有提供NeoPuppet 框架

·访问SSL源代码并进行手动修改——我已将该模组上传Github，请自行搜索查看

## 启用BSS

请查阅SheepScriptLit的配置文件，BSS由于其危险性，默认为关闭，使用必须手动打开。

## 脚本文件

在游戏根目录下（对于无版本隔离为 .minecraft/，对于版本隔离为 版本/）

文件夹SheepBothScripts内的文件将会在游戏初始化阶段被自动识别，在该目录下，或该目录的任何子目录下，后缀名为”.ssl.js”的文件将会被识别为脚本文件，自动加载。

如果出现乱码，请确保你的脚本文件编码和配置文件一致。

## 运行，订阅事件，编辑脚本信息

游戏初始化阶段加载时，脚本文件将会被执行一次，执行时有两个可用参数，

“script”和“station”

下面来看一段例子：

var System = Java.type("java.lang.System");

System.out.println("run successfully.");

station.sbs = "SheepBothScript test";

script.setContainerName("测试容器1");

script.enableFMLServerEvents();

script.listenNpcsEvent("listener1",Java.type("noppes.npcs.api.event.PlayerEvent.InteractEvent").class,function(event){

    event.player.message("事件监听1");

})

function fmlServerFired(event){

    System.out.println("SheepBothScript test server events.");

}

script是脚本容器对象，该对象有七个可用方法（其中有几个是重复方法）

setContainerName(String name) 设置容器名字，这会便于你识别自己的脚本文件

如果你没有设置容器名字，它默认会是这个文件的相对路径，以日志信息为准

enableFMLServerEvents() 启用FML服务端事件

disableFMLServerEvents() 禁用FML服务端事件

如果你启用了FML服务端事件，那么当FML服务端事件发生时，它将会触发你脚 本中名为fmlServerFired(event)的函数

FML服务端事件有五个：

AboutToStart,Starting,Started,Stopping,Stopped

对该方面，我不细细解释。

订阅事件有四个方法：

listen(String listenerName, @Nullable EventBus bus, String eventType, function(event))

listen(String listenerName, @Nullable EventBus bus, Class eventType, function(event))

这两项是常规的事件订阅方法，当bus未指定，默认为Forge事件总线

listenNpcsEvent(String listenerName, String eventType, function(event))

listenNpcsEvent(String listenerName, Class eventType, function(event))

这两项是订阅CNPC事件，即默认bus为NPC事件总线

取消订阅事件

unlisten(String listenerName)

## 错误与处理

如前面的那一个例子，在成功加载之后，会出现如下文字

[12:16:26] [Client thread/INFO] [sheepscriptlib]: Loading both side scripts.

[12:16:26] [Client thread/INFO] [STDOUT]: [jdk.nashorn.internal.scripts.Script$1$\^eval\\_::program:2]: run successfully.

[12:16:26] [Client thread/INFO] [sheepscriptlib]: Loading script, file name:test.ssl.js, container name:测试容器1, script state:running

[12:16:26] [Client thread/INFO] [sheepscriptlib]: Successfully loading both side scripts.

第一句意味着SSL开始加载脚本

第二句是例子中的语句，向外输出一句话

第三句是输出载入的脚本状态

第四句意味着SSL完成加载脚本

其中第三句，如果加载出错，那么state将会变成error。

事实上，无论是监听事件，FML服务端事件，运行过程中出错，该脚本状态都会变成error，状态为error时将不会运行任何事件！

如果你的脚本出错，你可以使用指令重加载脚本，或者用如下方法获取容器进行处理

var ScriptLoader = Java.type("saltsheep.ssl.script.ScriptLoader");

var container = ScriptLoader.loader.getContainer(containerName);

在获取到容器对象之后，你可以使用以下两个方法：

getLastError() 获取错误

resetLastError() 清除错误，这会使容器状态恢复到running

## 热重载

指令

/sslreloadscript 这将会重新加载所有脚本，并且重置它们的错误

## 跨服务端-客户端

这一部分内容涉及到相对复杂的功能，且客户端NPCAPI不受用，因此没有模组开发基础的人可以对此不做尝试，我有提供相关的库，请自行查阅源码

## 跨服务端-客户端 例子：

/\*\*

 \* @Author Saltfish\_Sheep (咸鱼羊)

 \*/

script.setContainerName("网络测试1");

var System = Java.type("java.lang.System");

var PacketSender = Java.type("saltsheep.ssl.network.PacketSender");

var PacketScript = Java.type("saltsheep.ssl.network.PacketScript");

var SPacketTest = Java.type("saltsheep.ssl.network.SPacketTest");//\*内置测试类

var SPacketScriptTest = Java.extend(PacketScript,{

    readBuffer:function(buf,data){

        var num = buf.readInt();

        data.num = num;

    },

    writeBuffer:function(buf,data){

        buf.writeInt(data.num);

    },

    invoke:function(data){

        var MC = Java.type("net.minecraft.client.Minecraft");

        //\*在日志看到这一条消息

        System.out.println(""+MC.getMinecraft().player+","+data.num);

    },

    getRegisterName:function(){

        return "network test";

    }

});

var packetFactory = function(){

    var packet = new SPacketScriptTest();

    packet.data = new Object();

    return packet;

};

PacketSender.INSTANCE.registerPacket("network test",packetFactory);

script.listenNpcsEvent("listener1",Java.type("noppes.npcs.api.event.PlayerEvent.InteractEvent").class,function(event){

    event.player.message("正在发送数据包");

    //\*SPacketTest会在日志输出5000

    var packetA = new SPacketTest();

    var packetB = packetFactory();

    packetB.data.num = event.player.world.getTotalTime();

    PacketSender.INSTANCE.sendToAll(packetA);

    PacketSender.INSTANCE.sendToAll(packetB);

})