**Mobs 技能拓展方案开发教程**

介绍:Mobs是一款开源的,高扩展型怪物应用插件.主要用于服务器RPG等各种需要对怪物进行属性修改,添加技能,添加刷新的地方.

本教程面向:

对Mobs插件有使用意向/需要对MC的生物进行扩展的技术/服主

开发条件:

·Java入门级以上水准---会使用Java语系的基本规范.

·使用BukkitAPI对MC进行插件开发的经验---可有可无,要求极低

·Mobs最新版本的文件主体---本文中一概称为Mobs.jar

·BukkitAPI\_1.7.2,1.7.10,1.8.3---本文提供SpigotAPI,对初学者基本同理

·JDK\_1.8---Java开发环境

·Java IDE---本文中以Eclipse为例(Eclipse神教大法好!!!)

工具链接:

·[Mobs原帖地址](http://www.mcbbs.net/thread-494926-1-1.html)

·J[ava基础教程](http://www.taobao.com/)

·[SpigotAPI下载](http://www.mcbbs.net/thread-372893-1-1.html)

·[BukkitAPI-JavaDoc](http://jd.bukkit.org/)

·[JDK\_1.8下载](http://www.xp510.com/xiazai/Application/program/23626.html)

·[Eclipse下载](http://www.newasp.net/soft/73001.html)

现在,我们创建好了Mobs 技能构造的开发环境,请大家跟我一起部署.

注:以我用的中文版Eclipse为例

·打开Eclipse

·文件->新建->Java项目

·项目名可随意填,如我起名为TestSkills

·右键该项目->构建路径->配置构建路径->库->添加外部Jar->选择下载的Mobs.jar和BukkitAPI->确定

·点开包资源管理器中的项目,我们要在Src中创建一个包,存放编写的技能(嘛,把不是技能的类和技能类都混乱的放在一起而且不建任何包也不是不可以.不过我非常不建议这么做.)

·包名按照一般格式起为cn.rpgmc.skills用于存放写好的技能(包名可以起任意名称).

·为了分类清晰,在建立一个包叫做cn.rpgmc.utils储存不是技能的工具类

·在skills包下创建一个类,我起名为Skill\_Damage (意为一个伤害技能).

·右键项目,新建一个文件叫做main.info,在里面书写作者信息(就是你想写点啥,也可以不建立)

以上做完就完成了对一个技能包结构的创建.

在写代码之前请确保你有一些bukkit插件的编写基础,

然后看一下我下面对本插件经常接触的几个类的介绍.

·ConfigurationSection

引入自BukkitAPI,是配置文件的节点,在本插件的技能扩展内代表你当前写的技能的节点,直接进行操作即可.

常用方法:

get(String) return Object:

获取你的技能配置中的一个参数项的值,没有返回NULL,可以是大部分类型.

set(String,Object) :

设置一个参数项的值,设置为NULL则删除该项

createSection(String) return ConfigurationSection:

在该技能配置节点中创建子节点.

getConfigurationSection (String)

return ConfigurationSection:

获取一个子节点,没有则返回NULL

getKeys (boolean) return Set<String> :

获取该节点的所有参数项的名字.

关于该类更多信息请详见BukkitAPI

·Mob

该类表示Mobs中的一个怪物.使用getE()方法可以获得怪物的Entity对象.

下一步请打开Skill\_Damage,我们来写一个可以被识别的技能类.

·让该类继承与Skill类

这时Skill下面出现红线,鼠标放在Skill上,点击导入Skill.

·继承三种构造方法

**public** Skill\_Damage() {

}

**public** Skill\_Damage(ConfigurationSection cfg) {

**super**(cfg);

}

**public** Skill\_Damage(String s, ConfigurationSection cfg) {

**super**(s, cfg);

}

如果擅自修改构造方法格式可能会造成不可预知的错误.

·使用eclipse实现所有未实现的方法

**然后来跟我来让这个技能实现功能:**

**·首先要知道Damage需要知道造成多少伤害**

**private** String dmg;

**※务必不要在这里对变量进行任何赋值操作.**

**如private String** dmg **=null;**

**创建这么一个字段**

**使用String型是因为我想让这个技能包含公式,通过替换变量和运算算出最终会造成的伤害**

**·对自动生成出来的需要重写的方法进行修改**

@Override

**public** String getType() {

**return** "Damage";

}

该方法返回的是技能的类型,并且作为该技能的一切标识.

@Override

**public** String help() {

**return** "技能类型:伤害技能\n"

+ "技能介绍:对目标进行伤害.\n"

+ "指令:\n"

+ " /mobs skill modify dmg [伤害公式] (设置造成伤害的公式,%m\_h%代表施放者血量,%e\_h%代表被施放者血量,例子:%e\_h%/10 就是造成10/1血量的伤害)";

}

该方法返回技能独有的帮助命令,请大概按照这个格式书写

@Override

**protected** **void** skillNext(ConfigurationSection arg0) {

**this**.dmg = arg0.getString("dmg");

//从配置中读取名叫dmg的项值赋值给dmg变量.

}

该方法内书写在对技能配置进行读取时进行的操作.

@Override

**protected** **void** newSkillNext() {

// **TODO** 自动生成的方法存根

**this**.dmg="";

}

该方法内书写在创建一个该类型技能时,各项属性的默认值.

@Override

**protected** **void** saveNext() {

getCfg().set("dmg", dmg);

}

该方法书写在保存属性到配置时进行的操作(getCfg()方法用于获取本技能的配置).

@Override

**public** String seeNext() {

**return** " 伤害公式:\n" + dmg + "\n";

}

该方法返回本技能独有的参数属性,大概按照我的规格书写.

@Override

**protected** **boolean** cmdElse(String[] arg0, Player arg1) {

**if** (arg0.length != 2)

**return** **false**;

**if** (arg0[0].equalsIgnoreCase("dmg")) {

**this**.dmg = arg0[1];

} **else**

**return** **false**;

**return** **true**;

}

该方法用于处理技能独有指令,比如这个dmg就会在玩家选定一个Damage技能并且输入 /Mobs skill modify dmg XXXX 时被触发并且把XXXX保存到dmg变量.

最后书写该技能用于实现功能的方法之前,因为本技能需要对字符串进行运算,新建一个类放入cn.rpgmc.utils包下内容如下：

public class Calc {

private static ScriptEngine jse = new ScriptEngineManager()

.getEngineByName("JavaScript");

public static Double calc(String str) {

try {

return Double.parseDouble((String) jse.eval(str));

} catch (NumberFormatException e) {

e.printStackTrace();

} catch (ScriptException e) {

e.printStackTrace();

}

return null;

}

}

**public** **boolean** canRangeToNearby();

重写该方法返回这个技能是否支持触发对象为附近实体

**public** **boolean** canTriggerToDying();

重写该方法返回这个技能是否支持触发类型为怪物死亡

以此类推.

@Override

**public** **void** run(Mob mob, Entity[] es, Event event) {

for(int i=0;i<es.length;i++)

{**if** (es[i] **instanceof** LivingEntity) {

LivingEntity e = (LivingEntity) es[i];

**if** (dmg != **null**)

{

String str = "";

str = dmg.replaceAll("%m\_h%",String.*valueOf*(((LivingEntity) arg0.getE()).getMaxHealth()));

str = str.replaceAll("%e\_h%", String.*valueOf*(e.getMaxHealth()));

e.damage(Calc.*calc*(str));

}

}

}

}

该方法处理一个怪物对其他生物施放技能的时候

注:mob为该怪物,es为被释放技能的生物数组,LivingEntity是生物实体,event是该技能触发的事件,比如这个技能就会触发EntityDamageByEntity事件,具体请参见BukkitAPI(大部分技能都用不到这个参数,当技能为周期触发或者产生时触发则为null).

@Override

**public** String[] getCanExecuteVersion() {

**return** **new** String[] { "1.7", "1.8" };

}

#这一步不是必要的

这个方法返回的是本技能支持的版本号,如:1.8

忽略这步默认为支持1.7-1.8所有版本.

写完这些我们就完成对第一个技能的构建了,导出就可以使用了.如果想继续写其他技能,可以继续在这个工程内进行构建即可.

·导出Jar文件(放在 安装Mob的 服务端/plugin/Mobs/Skills/内即可载入)

右键工程->导出->选择导出目标->Java->Jar文件->下一步->选择要导出的资源勾选左边工程下的Src,右边的全部打挑,下面选项只选择导出生成的类文件和资源.->选择导出目标(随便选一个位置生成)->完成.