




# Shimeji开源桌宠代码学习（1）

原创 置顶  AlanJager 于 2016-11-30 22:27:39 发布 阅读量1.5w 收藏 68 点赞数 23

分类专栏: [shimeji](#) [Java基础](#) 文章标签: [shimeji](#) [Github](#) [Java](#) [mac](#) [开源](#)

Shimeji在日语中本意为“蘑菇”。

我们这里的Shimeji是种可以在电脑桌面上四处走动，玩耍，分裂以及卖萌捣乱的桌面程序。

这种桌面程序具有高度可配置的特点。其运行方式是依靠xml文件来控制吉祥物的动作及动作频度。而吉祥物的形象和特殊动作可以通过替换图片来达到定制的效果。

Shimeji 程序由日本的Yuki Yamada开发制作，其官方网页为：

[www.group-finity.com/Shimeji/](http://www.group-finity.com/Shimeji/)

现在我们所见到的各式各样的shimeji形象，也均是由各个作者利用官方网站提供的原始程序修改而成的。

由于现在该网站似乎已经停止维护，所以这里使用的是一个Github上的作者开发的Shimeji4mac的项目作为本博客的学习内容

地址为：

<https://github.com/nonowarn/shimeji4mac>

我的fork地址为

<https://github.com/AlanJager/shimeji4mac>

原本考虑到作者太久没有上Github所以不好进行pull request，抱着试一试的心态给作者写了一封邮件，没想到收到了回复

Hello,

I'm not active on the GitHub for years, but I still alive and love Shimeji.

Please send me Pull Request.

Thanks,  
-nonowarn

所以有意向进行开发的朋友也可以放心了。接下来就进入正题了，

首先打开Main.java

```
1 private static final Logger log = Logger.getLogger(Main.class.getName());
2
3 static final String BEHAVIOR_GATHER = "マウスの周りに集まる";
4
5 static {
6     try {
7         LogManager.getLogManager().readConfiguration(Main.class.getResourceAsStream("/logging.properties"));
8     } catch (final SecurityException e) {
9         e.printStackTrace();
10    } catch (final IOException e) {
11        e.printStackTrace();
12    }
13 }
14
15 private static Main instance = new Main();
16
17 public static Main getInstance() {
18     return instance;
19 }
20
21 private final Manager manager = n
```



AlanJager

关注

23



```
23 |     private final Configuration configuration = new Configuration();
```

首先是util.logging.Logger类，大多我们都使用log4j进行日志记录，对这个难免陌生，下面一篇介绍的比较详细的文章[java.util.logging.Logger使用详解](#)

我也简短的说明一下，Logger类最大的特点就是对日志的级别分的非常详细，所有级别都定义在util.logging.Level类中，分别是：Severe, Warning, Info, Config, Fine, Finer, Finest，此外还有OFF级别用于关闭日志，All则是启动所有日志记录

基本的使用方式如下：

```
log.log(Level.INFO, "設定ファイルを読み込み({0})", "/動作.xml");
```

然后声明了一个静态常量，  
BEHAVIOR\_GATHER = "マウスの周りに集まる";  
这个变量用于响应聚集事件时，创建所有shimeji的动作。

之后的LogManager用于管理日志，详细可以参考[java.util.logging.LogManager](#)

对基本使用进行了说明。

然后创建了静态的Main类和获取Main类的方法，这样做的原因非常明显，这也是何时使用static的习惯

这里翻译一段Stack Overflow上面一个回答：有一个这样的法则，问自己不构建Obj而去直接调用方法是否有意义，如果有意义那么就用使它为static。比方说你的汽车类Car里有一个方法Car::convertMPgToKpl(double mpg)，可能有人需要使用转换的方法，但是并不需要构建一个Car实例。

这大致就是个人认同的使用static的原则。

然后继续创建了Manger和Configuration的实例。

紧接着就是主函数的执行

```
1 |     public static void main(final String[] args) {
2 |
3 |         getInstance().run();
4 |     }
5 |
6 |     public void run() {
7 |
8 |         // 設定を読み込む
9 |         loadConfiguration();
10 |
11 |        // トレイアイコンを作成する
12 |        createTrayIcon();
13 |
14 |        // しめじを一匹作成する
15 |        createMascot();
16 |
17 |        getManager().start();
18 |    }
```



AlanJager

关注

23

通过之前的static method获取main实例，然后调用定义的方法Main::run()

首先是调用Main::loadConfiguration()

```
1 private void loadConfiguration() {
2
3     try {
4         log.log(Level.INFO, "設定ファイルを読み込み({0})", "/動作.xml");
5
6         final Document actions = DocumentBuilderFactory.newInstance().newDocumentBuilder().parse(
7             Main.class.getResourceAsStream("/動作.xml"));
8
9         log.log(Level.INFO, "設定ファイルを読み込み({0})", "/行動.xml");
10    }
```



首先使用DocumentBuilderFactory来parse一个xml文件，如官方文档描述，DocumentBuilderFactory：

Defines a factory API that enables applications to obtain a parser that produces DOM object trees from XML documents.

这里使用的是w3c的DOM来解析xml文件，关于如何用 [DocumentBuilderFactory解析xml](#) 这篇文章里以Shimeji的部分代码为例子进行了说明。当然除了使用DOM之外，Java还有许多用于解析xml文 [java中四种操作（DOM、SAX、JDOM、DOM4J）xml方式详解与比较](#) 中对这几种方法进行了详细说明，并且给出比较其效率的代码。由于这个桌宠的xml文件本身并不是很大，所以使用DOM处理

接下来我们来看Configuration::load()这个方法，类似DocumentBuilderFactory解析xml，Shimeji中定义了Entry类作为几本节点，我们先来看Entry类的定义：

```
1 public class Entry {
2
3     private Element element;
4
5     private Map<String, String> attributes;
6
7     private List<Entry> children;
8
9     private Map<String, List<Entry> > selected = new HashMap<String, List<Entry>>();
10
11     public Entry(final Element element){
12         this.element = element;
13     }
14
15     public String getName() {
16         return this.element.getTagName();
17     }
18 }
```

定义了基本的属性和get方法，这里的Element实际上是Dom元素中的一个接口，它的super interface就是node，然后对属性，子节点进行了定义。然后类添加方法，根据解析xml管用思路可以知道，我们需要获取属性和获取节点的方法这里就不多赘述了。

我们看下面实现的一个方法：

```
1 public Map<String, String> getAttributes() {
2     if ( this.attributes!=null) {
3         return this.attributes;
4     }
5
6     this.attributes = new LinkedHashMap<String, String>();
7     final NamedNodeMap attrs = this.element.getAttributes();
8     for(int i = 0; i<attrs.getLength(); ++i ) {
9         final Attr attr = (Attr)a
10         this.attributes.put(attr,
```



AlanJager

关注

23

```

11         }12
13         return this.attributes;
14     }

```

作者使用了LinkedHashMap用于储存xml中节点的多个属性，通过 [LinkedHashMap](#)、[HashMap](#)以及[TreeHashMap](#)的比较实际上使用LinkedHashMap能够保证放入元素的顺序，我们先记录这个特点，在这里似乎还没体现出使用LinkedHashMap的优势。

在Configuration::load()方法里，对每一个xml文件中定义的动作进行了创建，

```

1         for (final Entry node : list.selectChildren("動作")) {
2
3             final ActionBuilder action = new ActionBuilder(this, node);
4
5             if ( this.getActionBuilders().containsKey(action.getName())) {
6                 throw new ConfigurationException("動作の名前が重複しています:"+action.getName());
7             }
8
9             System.out.println("action name is: " + action.getName());
10
11             this.getActionBuilders().put(action.getName(), action);
12         }

```

同时附上一个动作的xml内容：

```

1     <動作 名前="歩く" 種類="移動" 枠="地面">
2         <アニメーション>
3             <ポーズ 画像="/shime1.png" 基準座標="64,128" 移動速度="-2,0" 長さ="6" />
4             <ポーズ 画像="/shime2.png" 基準座標="64,128" 移動速度="-2,0" 長さ="6" />
5             <ポーズ 画像="/shime1.png" 基準座標="64,128" 移動速度="-2,0" 長さ="6" />
6             <ポーズ 画像="/shime3.png" 基準座標="64,128" 移動速度="-2,0" 長さ="6" />
7         </アニメーション>
8     </動作>

```

传入一个动作节点，然后交给ActionBuilder处理，ActionBuilder的constructor()如下：

```

1     public ActionBuilder(final Configuration configuration, final Entry actionNode) throws IOException {
2         this.name = actionNode.getAttribute("名前");
3         this.type = actionNode.getAttribute("種類");
4         this.className = actionNode.getAttribute("クラス");
5
6         log.log(Level.INFO, "動作読み込み開始({0})", this);
7
8         this.getParams().putAll(actionNode.getAttributes());
9         for (final Entry node : actionNode.selectChildren("アニメーション")) {
10             this.getAnimationBuilders().add(new AnimationBuilder(node));

```



通过读取一个动作下的アニメーション，并将其传递给AnimationBuilder，

接下来我们看AnimationBuilder的constructor()

```

1     public AnimationBuilder(final Entry animationNode) throws IOException {
2         this.condition = animationNode.getAttribute("条件") == null ? "true" : animationNode.getAttribute("条件");
3
4         log.log(Level.INFO, "アニメーション読み込み開始");
5
6         for (final Entry frameNode : animationNode.getChildren()) {
7
8             this.getPoses().add(loadP

```



AlanJager

关注

23



```

9         }10
11         log.log(Level.INFO, "アニメーション読み込み完了");
12     }

```

对于读入的一个节点判断其是否满足发生条件，如果满足则对Pose进行读取，每一个Pose对应一张png图片

```

1     private Pose loadPose(final Entry frameNode) throws IOException {
2
3         final String imageText = frameNode.getAttribute("画像");
4         final String anchorText = frameNode.getAttribute("基準座標");
5         final String moveText = frameNode.getAttribute("移動速度");
6         final String durationText = frameNode.getAttribute("長さ");
7
8         final String[] anchorCoordinates = anchorText.split(",");
9         final Point anchor = new Point(Integer.parseInt(anchorCoordinates[0]), Integer.parseInt(anchorCoordinates[1]));
10

```

同时对图像的锚点，持续时间，移动长度进行了设置，最后返回一个Pose实例。整个流程结束后，一整个アニメーション将会存储在poses这个ArrayList里则使用一个ArrayList来存储一系列这样的AnimationBuilder。即通过ActionBuilder管理所有的AnimationBuilder，然后每一个AnimationBuilder管理一组Poses。

继续看ActionBuilder的constructor，这里还使用了ActionRef这个类，这里主要处理的是复合动作的实现，此处仅仅将该对应的复合动作节点放在了ArrayList内，到此ActionBuilder则完成了工作。简单动作和复合动作分别对应了一个ActionBuilder。

然后Configuration::load()进行了Behavior的加载

```

1         for (final Entry list : configurationNode.selectChildren("行動リスト")) {
2
3             log.log(Level.INFO, "行動リスト...");
4
5             loadBehaviors(list, new ArrayList<String>());
6         }

```

类似对动作的读取，通过定义BehaviorBuilder来描述一个Behavior，

```

1         <行動 名前="マウスの周りに集まる" 頻度="0">
2             <次の行動リスト 追加="false">
3                 <行動参照 名前="座ってマウスのほうを見る" 頻度="1" />
4             </次の行動リスト>
5         </行動>

```

主要负责对发生的频度等进行读取。

最后Main::loadConfiguraion()在加载完两个配置文件后调用了如下方法：

```
this.getConfiguration().validate();
```

这个确认有效的方法分别去检查ActionBuilder和BehaviorBuilder是否存在问题，即重新确认当前存储的节点是否确实存在于配置文件中的安全检测。Main::loadConfiguration()结束。



AlanJager

关注

23



文章知识点与官方知识档案匹配，可进一步学习相关知识

Java技能树 首页 概览 147280 人正在系统学习中

Shimeji开源桌宠代码学习（3）Alan Jager sp

之前的文章里，我们讨论了在Main::run()中调用的Main::loadConfiguration()方法， public void run() { // 設定を読み込む loadConfiguration(); // トレイアイコンを作成する on(); // しめじを一匹作成する createMascot(); getManage

用 Python 制作一个桌面宠物，好玩 最新发布 y1282037271的

今天，我们来分享一个宠物桌面小程序，全程都是通过 PyQt 来制作的，对于 Python GUI 感兴趣的朋友，千万不要错过哦！我们先来看看最终的效果，对于一个小小的说，还是不错的！

python桌面宠物 热门推荐 HI\_REY的

亲测有效！

Shimeji开源桌宠代码学习(2)\_会满屏跑的手机桌宠开源码

在Shimeji开源桌宠代码学习(1)中描述了整个配置文件加载的过程,其中很多地方使用了Builder这一概念,首先我们先来看看这些Builder都有什么特征,我认为我们从Animation下层的Builder来看起,应该更容易找到其特点,首先是AnimationBuilder的定义,它主要有两个实例变量 privatefinalString condition; privatefinalList<Pose...

写一个linux平台的桌面宠物\_在centos6中养桌宠

name=url.split('/')[1]+''.zip'print('正在下载资源',name)data=requests.get(url).contentwithopen(name,'wb')asf:f.write(data)print('下载完成!')path=os.environ['HOME']+'/shimeh.exists(path):os.makedirs(path)print('正在解压资源...')z=zipfile.ZipFile(name,'r')z....

“相关推荐”对你有帮助么？

非常没帮助 没帮助 一般 有帮助 非常有帮助

关于我们 招贤纳士 商务合作 寻求报道 400-660-0108 kefu@csdn.net 在线客服 工作时间 8:30-22:00

公安备案号11010502030143 京ICP备19004658号 京网文〔2020〕1039-165号 经营性网站备案信息 北京互联网违法和不良信息举报中心 家长监护 网络110报警服务 中国互联网举报中心 Chrome商店下载 账号管理规范 版权与免责声明 版权申诉 出版物许可证 营业执照

©1999-2024北京创新乐知网络技术有限公司



AlanJager  
码龄9年 暂无认证

34	19万+	98万+	10万+	
原创	周排名	总排名	访问	等级
1377	21	49	0	84
积分	粉丝	获赞	评论	收藏



私信

关注

大额流量券免费送

发布一篇就可获得！

去查看

流量券

搜博主文章



热门文章

Shimeji开源桌宠代码学习（1） 15402

Mac下 ImportError: No module named cv2 问题的解决 8117

DocumentBuilderFactory解析xml 7434

Yii2.0 RESTful风格的Controller与ActiveController 6948

Yii2.0 初识 RESTful Serializer 5297




AlanJager

关注

23



分类专栏

	虚拟化	1篇
	Swift	5篇
	SpriteKit swift	
	iOS App development	3篇
	iOS app 开发	2篇
	PHP	13篇

大家在看

- 【C++】程序设计期末复习 day01（连更）  
基础知识点和题目
- 如何使用python发邮件
- gstreamer+qt5实现简易视频播放器 345
- royale
- 如何设计一个秒杀系统

最新文章

- QEMU FT方案介绍
- 关于虚拟化（virtualization）的一些知识
- 关于Qcow2特性的一些总结

2020年 1篇	2017年 4篇
2016年 28篇	2015年 6篇



AlanJager

关注

23