链滴

# java 简单代码生成器

作者：umeone

原文链接：https://ld246.com/article/1514214688024

来源网站：链滴

- **一个简单的代码生成器实现。**

---

- 代码如下:

```
/**
 * 测试
 * Created by GN on 2016/11/27.
 */
public class GnTest {

    @Test
    public void testGn() throws IOException {
        GnContext gnContext = new GnContext();
        gnContext.put("domain","User");
        gnContext.put("lowerDomain","user");
        String path = "E:\\WorkSpace\\Idea\\ztx\\gn-cc\\src\\main\\resources\\DaoImpl.java";
        Template template = GnUtil.getTemplate(path);
        String target = "G:\\DaoImpl.java";
        File file = new File(target);
        template.merge(gnContext,file);
    }

}

/**
 * 定义模板插入值
 * Created by GN on 2016/11/27.
 */
public class GnContext {

    /**
     * 存储模板值
     */
    private Map<String, String> context = new HashMap<>();

    public GnContext() {

    }

    public GnContext(Map<String, String> context) {
        this.context = context;
    }

    public void put(String key, String value) {
        context.put(key, value);
    }

    public String get(String key) {
        return context.get(key);
    }

}
```

原文链接：java 简单代码生成器

```
/**
 * Created by GN on 2016/11/27.
 */
public abstract class GnUtil {

    /**
     * 获取模板信息
     * @param filePath 模板路径
     * @return
     */
    public static Template getTemplate(String filePath) throws IOException {
        if (!StringUtil.hasLength(filePath)){
            throw new IllegalArgumentException("模板路径不能为空");
        }
        File file = new File(filePath);
        if (!file.isFile() || !file.exists()){
            throw new IllegalArgumentException("无法找到指定文件:"+filePath);
        }
        return new Template(file);
    }

}

/**
 * 模板处理
 * Created by GN on 2016/11/27.
 */
public class Template {

    private static final String TAG_PREFIX = "${";
    private static final String TAG_POSTFIX = "}";

    /**
     * 模板信息
     */
    private BufferedReader reader;
    private File file;
    /**
     * 模板标签,eg:${domain}:  key=domain,value=${domain}
     */
    private Map<String, String> tag = new HashMap<>();

    public Template(File file) throws IOException {
        this.file = file;
        //初始化模板输入流
        initReader();
    }

    private void initReader() {
        if (file == null) {
            throw new IllegalArgumentException("获取模板失败");
        }
        try {
            InputStreamReader inputStreamReader = new InputStreamReader(new FileInputStre
```

```
m(file), "UTF-8");
            this.reader = new BufferedReader(inputStreamReader);
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }
    }

    /**
     * 处理标签到tag中
     *
     * @param text
     */
    private void initTag(String text) {
        if (!StringUtil.hasLength(text)) {
            return;
        }
        String tagFlag = getTagFlag(text);
        String tagText = getTagText(text);
        this.tag.put(tagText, tagFlag);
        //递归解析一行中的多个标签
        int prefix = text.indexOf(TAG_PREFIX);
        String subText = text.substring(prefix + 2 + tagText.length());
        if (hasTag(subText)) {
            initTag(subText);
        }
    }

    /**
     * 获取标签，${domain}
     *
     * @param text
     * @return
     */
    private String getTagFlag(String text) {
        int prefix = text.indexOf(TAG_PREFIX);
        int postfix = text.indexOf(TAG_POSTFIX);
        return text.substring(prefix, postfix + 1);
    }

    /**
     * 获取标签中的内容，${domain} --> domain
     *
     * @param text
     * @return
     */
    private String getTagText(String text) {
        String tagFlag = getTagFlag(text);
        return tagFlag.substring(2, tagFlag.length() - 1);
    }

    /**
     * 判断文本中是否含有标签
```

```
     *
     * @param text
     * @return
     */
    private boolean hasTag(String text) {
        if (!StringUtil.hasLength(text)) {
            return Boolean.FALSE;
        }
        return text.indexOf(TAG_PREFIX) > 0 && text.indexOf(TAG_POSTFIX) > 0;
    }

    /**
     * 生成模板
     *
     * @param context 模板值定义
     * @param target  目标生成文件
     */
    public void merge(GnContext context, File target) throws IOException {
        if (reader == null) {
            throw new IllegalArgumentException("读取模板失败");
        }
        if (context == null) {
            throw new IllegalArgumentException("获取GnContext失败");
        }
        if (target == null) {
            throw new IllegalArgumentException("获取输出流失败");
        }
        String temp;
        FileWriter writer = new FileWriter(target, true);
        while ((temp = reader.readLine()) != null) {
            if (hasTag(temp)) {
                //解析出所有的标签
                List<String> tagTextList = findTagTextList(temp);
                if (!tagTextList.isEmpty()) {
                    for (String text : tagTextList) {
                        String value = context.get(text);
                        if (StringUtil.hasLength(value)){
                            temp = temp.replace(TAG_PREFIX + text + TAG_POSTFIX, value);
                        }
                    }
                }
            }
            writer.write(temp + "\n");
            writer.flush();
        }
        writer.close();
    }

    /**
     * 获取文本中的标签
     *
     * @param lineText
     * @return
     */
```

```
    private List<String> findTagTextList(String lineText) {
        List<String> tagTextList = new ArrayList<>();
        if (StringUtil.hasLength(lineText) && hasTag(lineText)) {
            String tmp = lineText;
            while (hasTag(tmp)) {
                String tagText = findTagText(tmp);
                if (!tagTextList.contains(tagText)) {
                    tagTextList.add(tagText);
                }
                int begin = tmp.indexOf(tagText);
                tmp = tmp.substring(begin + tagText.length()+1);
            }
        }
        return tagTextList;
    }

    /**
     * 获取文本中的单个标签
     *
     * @param text
     * @return
     */
    private String findTagText(String text) {
        if (!StringUtil.hasLength(text)) {
            return null;
        }
        return getTagText(text);
    }

}
```

---

- 一下为测试使用模板：

```
/**
 * Created by GNon 2016/9/8.
 */
@Service
@Transactional
public class ${domain}ServiceImpl implements I${domain}Service {

    @Autowired
    private ${domain}Dao dao;

    @Override
    public void save(${domain} ${lowerDomain}) {
        dao.save(${lowerDomain});
    }

    @Override
    public void update(${domain} ${lowerDomain}) {
        dao.update(${lowerDomain});
    }
```

```
    @Override
    public void delete(Serializable id) {
        dao.delete(id);
    }

    @Override
    public ${domain} findById(Serializable id) {
        return dao.findById(id);
    }

    @Override
    public List<${domain}> findAll() {
        return dao.findAll();
    }

    @Override
    public PageData findListByPage(PageData pageData) {
        return dao.findListByPage(pageData);
    }
}
```

- 以下是通过上述代码实现的简单生成器

```
/**
 * 代码生成器
 * Created by GN on 2016/11/28.
 */
public class GnCreator {

    /**
     * 实体
     */
    private List<String> domains = new ArrayList<>();
    /**
     * 模板基础路径
     */
    private String baseTemplatePath = "E:\\WorkSpace\\demo\\src\\main\\resources\\";
    /**
     * 生成文件保存基本路径
     */
    private String baseSavePath = "E:\\WorkSpace\\demosrc\\main\\java\\com\\gn\\demo\\";
    /**
     * 模板
     */
    private List<String> templates = new ArrayList<>();

    private final String DAOIMPL = "Dao.java";
    private final String SERVICE = "Service.java";
    private final String SERVICEIMPL = "ServiceImpl.java";

    public GnCreator() {
        //初始化生成器信息
        //初始化实体集合
```

```
        String[] domainList = {"User"}; //如果实体很多，可以通过注解扫描的方式去获取，不用一个个的去写
        List<String> asList = Arrays.asList(domainList);
        domains.addAll(asList);
        //初始化模板
        templates.add(DAOIMPL);
        templates.add(SERVICE);
        templates.add(SERVICEIMPL);
    }

    public void create() throws IOException {
        System.out.println("代码生成器开始执行>>>>>>>>>>>>>>>>>>>>");
        for (String domain : domains) {
            System.out.println("开始生成实体【"+domain+"】代码.....");
            //模板值定义
            GnContext gnContext = new GnContext();
            gnContext.put("domain",domain);
            gnContext.put("lowerDomain",domain.toLowerCase());
            //根据模板生成代码
            for (String tp : templates) {
                String templatePath = baseTemplatePath+tp;
                Template template = GnUtil.getTemplate(templatePath);
                //根据不同的模板保存到不同的目录
                String targetFilePath = "";
                String targetFileName = "";
                if (tp.equals(DAOIMPL)){
                    targetFilePath = baseSavePath+"dao\\impl\\";
                    targetFileName = domain+DAOIMPL;
                }else if (tp.equals(SERVICE)){
                    targetFilePath = baseSavePath+"service\\";
                    targetFileName = "I"+domain+SERVICE;
                }else if (tp.equals(SERVICEIMPL)){
                    targetFilePath = baseSavePath + "service\\impl\\";
                    targetFileName = domain+SERVICEIMPL;
                }
                //判断目录是否存在，如果不存在则创建
                File file = new File(targetFilePath);
                if (!file.exists()){
                    file.mkdirs();
                }
                File targetFile = new File(targetFilePath+targetFileName);
                template.merge(gnContext,targetFile);
            }
            System.out.println("生成实体【"+domain+"】代码完成.....");
        }
        System.out.println("代码生成器执行完毕>>>>>>>>>>>>>>>>>>>>>>>");
    }

}
```

---

● 以上为所有代码，仅为适用自己的需求，如果能帮助到其他人更好，说下简单思路：

代码主要实现功能是获取模板中的 ${domain} ,然后通过替换的方式把传入的 GnContext 所对应的

，最终实现代码生成。替换一处我用的是直接查询替换，如果你愿意也可以用正则表达式去处理，这也许会简洁很多。