

12 | 组织好代码文件，要有“用户思维”

2019-01-30 范学雷



上一讲中，我们讲了如何组织代码段，今天我来讲下，如何组织代码文件。

最开始接触一个项目代码时，我们最渴望的，就是快速揭开项目的面纱。这个项目是干什么的？是怎么做的？该怎么使用？

有很多这样的问题，排着队等我们处理。我们需要从一个点开始，先撕破一点点皮，然后，像剥洋葱一样，一层一层地阅读，一层一层地了解。

刚拿到一个项目的代码时，你最想找哪一个文件？面对大量的文件，该从哪里入手？创建一个项目时，各式各样的文件该怎么规整？

如果一个项目很小，只有三五个文件，我们不用担心上述的问题。

但事实上，一个典型的软件项目，拥有上百个文件是寻常的事情。如果这些文件组织混乱，就会让整个项目乱糟糟的，我们很难入手去查找、阅读和测试。

其实文件的组织是一个简单的事情，但这些简单的事情如果不能做得清晰、明了，就会变成一个效率的黑洞。

文件的组织要层次分明、易于检索、一目了然。要做到这一点，我们可以从用户思考问题的逻辑入手。

逻辑之一：软件是干什么的？

无论我们开始一个软件项目，还是阅读一个软件的代码，第一个遇到的问题就是，这个软件是干什么的？

可以回答这个问题的文件叫做**README**，它的命名全部使用大写字母。需要被放在一个软件工程的根目录里，方便人或者机器第一时间找到，从而轻而易举地找到并进行阅读。

“软件要用来干什么？”是一个软件工程的启动问题。

一个软件项目开始时，这个问题的答案可以不是很丰满，但是，最基本的思想一定要有。随着软件的研发进程，它的描述可以越来越清晰。软件成型之前，这个问题必须干脆地解决掉，得到明确的答案。

这个问题的描述，要记录在代码的工程里。可以让代码的阅读者轻而易举地找到并阅读。

由于机器的参与，尤其是在线代码管理服务（比如**GitHub**、**Maven**）的广泛使用，**README**的名字和位置也就慢慢地形成了共识。

逻辑之二：软件可以拿来用吗？

如果我们看到了**README**，想使用这个软件，那么紧接着的问题就是，这个软件我们可以使用吗？

所有的软件，都有归属，都受版权的保护。谁拥有这个软件的版权？这是我们需要关注的一个问题。

有时候，一个软件包含很多贡献者，不同的贡献者有不同的版权诉求。软件的不同部分，就有不同的版权。

这种情况下，**版权描述一般放在每一个源文件的头部**。不同的源文件可以有不同的版权，同一个源文件也可以有一个以上的版权所有者。

如果版权来源只有一个，而且源文件头部没有版权描述，我们就需要把版权描述放到最显眼的地方。**这个地方就是软件工程的根目录，命名为COPYRIGHT，全部使用大写字母。**

没有版权描述的软件，并不是没有版权保护。如果一个软件没有版权描述或者版权描述不清晰，使用起来有很多法律风险。如果这个软件依赖外部的版权，那么问题就会变得更为复杂。

有了版权保护，不代表我们就不能使用这个软件了。我们能不能使用、怎么使用，是由软件的许可证确定的。

许可证文件是**LICENSE**，全部使用大写字母，放在软件工程项目的根目录下。

当使用软件的时候，不能超越许可证约定的范围。一个没有许可证的软件，我们是不能使用

的，因为不知道许可的范围，也不知道应承担的义务。同样，如果一个软件的许可证不清晰，或者我们不了解，那么使用起来也会有很多法律问题。

逻辑之三：软件是怎么实现的？

作为程序员，代码是我们看软件世界的语言。我们关心的下一个问题就是，这个软件是怎么实现的？

代码的位置在现在的软件工程里有了一定的共识。通常来说，源代码存放在根目录下的**src**目录下。

当看到**src**目录的时候，我们就可以确认，这里面是源代码文件。当我们要查找源代码的时候，也是在软件工程文件里查找**src**目录。我不建议在这里搞创新，不要使用其他的名字或者位置。

但源代码并不能一股脑地堆在**src**这一个目录里。**src**目录下面可以有多个子目录。一般来说，稍具规模、相对正规的软件，都需要有命名空间的区隔。使用命名空间的区隔至少有三个好处：

1. 可以把一个组织的代码和另外一个组织的代码区隔开；
2. 可以把一个项目的代码和另外一个项目的代码区隔开；
3. 可以把一个模块的代码和另外一个模块的代码区隔开。

每一个命名空间的节点，都要对应一个文件目录。比如，我们常用的**java.lang**这个命名空间，就要相应地使用“**java/lang**”这两级目录。

如果软件项目把所有的源代码文件不加分别地放在同一个目录下，说明软件的开发人员并没有厘清代码之间的逻辑关系。纠缠在一起的代码越多，代码越难以维护，代码的安全越难以保证。

逻辑之四：软件该怎么测试？

如果要使用或者发布一个软件，最值得关注的还有软件的质量。软件的质量，首先要通过软件测试这一关。那么软件该如何测试呢？这是我们要面对的另一问题。

有很多传统的软件项目，测试代码和功能代码是放在同一个目录下的。如果一个项目比较小，那么这样做也许没什么大问题。一旦软件工程变得复杂，这样做就会让项目变得难以管理，尤其是在测试人员和开发人员分离的情况下。

让测试人员看着实现代码写测试，会误导测试的用例选择和测试效果；让开发人员看着测试代码改实现，也会影响开发的质量和效率。

既然要分工，不同的代码就要区隔开来。

如果开发和测试是一个人，或者是同一个小组成员，是不是就可以混在一起了呢？当然不是，

因为看代码的人依然可能是分工的。区隔实现和测试，可以尽可能照顾到每个工程师，方便他们工作。

幸运的是，现在的很多软件项目，大都采用了分离的组织方式。通常来说，源代码要存放在根目录下的`test`目录下。

仅仅放在对应的目录下还不够，测试文件本身还有一个需要注意的地方，一个测试文件，最好执行一个独立的任务。如果测试出错误，我们就能够快速定位错误。这也要求测试的目标要小，测试的结果要清晰，测试的组织要照应功能代码的组织。

比如说，对`java.io`测试的文件，要放在`java/io`测试目录下；对`java.util`测试的文件，要放在`java/util`目录下。这种互相照应的组织方式，从目录名就可以看出测试的范围。这样既可以快速从功能代码找到测试代码，也可以快速地从测试代码找到功能代码。

软件该怎么使用？

使用软件工程项目文件的，不仅仅只有程序员，还有软件的用户。

要是只能通过阅读软件代码来揣测软件该怎么使用，这样既没有效率，也容易犯错，更偏离了软件设计者的初衷。

一个好的软件，要尽可能降低使用门槛。编写使用指南和代码示例是两个常用的办法。一份好的用户文档，应该让软件的用户快速入门，然后再逐步深入地了解整个软件的使用细节，以及潜在的问题。

软件的文档，需要随着软件的变更不断升级维护。有很多项目会把文档和代码分开管理。

但实际上，这样做有一些缺陷，它会让文档变得难以维护。

我们经常看到很多文档和软件脱节的软件，很大一部分是由于项目组织和管理方面的问题导致的。如果代码的变更，也需要相应地变更文档，那么文档和代码统一管理，是一个更高效的组织方式。

如果你留意，就会注意到现在的很多项目，在根目录下，有一个名字为`docs`或者`doc`的目录。这个目录就是存放软件文档的。

作为程序员，我们不仅要熟悉源代码，还要熟悉文档。当需要更直观的用户指南或者代码示例时，就要写作这样的软件文档。对于每一行的代码变更，我们都要问，需不需要文档变更？如果代码和文档一致的话，就会节省我们大量的维护时间和维护成本。

下面的例子，是一个常见的软件组织形式，也是我们对上述讨论的小结。

Project directory layout:

```
src/  
  java/  
    lang/  
    io/  
  javax/  
    net/  
      ssl/  
test/  
  javax/  
    net/  
      ssl/  
doc/  
make/  
README  
COPYRIGHT  
LICENSE
```

小结

我们的日常软件开发工作，有很多都依赖于集成开发环境（IDE）。主流的IDE有缺省的文件组织形式。一般情况下，我们可以使用缺省的组织形式，然后添加进缺失的内容。

也有很多软件开发，不依赖于IDE。这就需要我们自己规划好文件的组织原则和基本形式。不同的语言，不同的项目，文件的组织方式差别可能很大。

如果你需要自己制定组织形式，我建议参考一些成功项目的组织方式。比如，如果你要做一个中间件，为客户提供类库，就可以参考OpenJDK的文件组织方式。

如果没有什么现成的项目可以参考借鉴的，请记住以下两点：

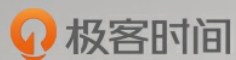
1. 文件的组织要一目了然，越直观，越有效率；
2. 可维护性要优先考虑。这要求文件组织要层次分明，合理区隔、照应、使用不同的空间。

一起来动手

由于项目的多样性，项目文件组织的具体形式有很多差异。借这个机会，我也想学习一下大家的项目文件组织经验。你做的项目，是怎么组织文件的？为什么选择这种组织的形式？你最欣赏的这种形式的哪几点？你阅读一个项目代码时，是怎么一步一步深入进去的？欢迎你把你的经验公

布在讨论区，我们一起来学习，一起进步。

欢迎你把这篇文章分享给你的朋友或者同事，一起来探讨吧！



代码精进之路

你写的每一行代码都是你的名片

范学雷

Oracle 首席软件工程师
Java SE 安全组成员
OpenJDK 评审成员



新版升级：点击「 请朋友读」，10位好友免费读，邀请订阅更有**现金**奖励。

精选留言



MOV AX,0

公司项目模块的大致结构：

```
build/  
src/  
main/  
java/domain.parent.module/  
constants/  
entity/  
dao/  
mapper/  
service/impl/  
facade/impl/  
job/  
utils/  
resources/  
META-INF/  
dubbo/
```

3

spring/
app.properties
log4j.properties
/test/java/domain.parent.module/
dao/
service/
facade/
job/
utils/
TestBase.java
deploy.properties
jenkinsfile
pom.xml

不经意间已经习惯这种结构了。很多东西用的时候很顺，以致于都察觉不到它的好处。看老师的文章感觉非常亲切，也收获了很多！

2019-01-31

作者回复

谢谢你分享这个典型的代码文件组织结构。

2019-02-01



Being

👍 3

doc/

需求文档（按功能模块分子目录）、各成员的日志记录（按项目节点分子目录）、UI设计

Source/

VS工程下的多个项目相关Project工程目录、基础库、test工程、Tools工程，.sln文件一般放在该目录，主要是VS IDE的组织方式

每个工程下就是会细分像老师说的src

Product/

一般还要分Debug/Release Win32/x64, 存放相应的exe生成文件以及依赖库

主要方便发布与测试

Versions/

就是每个节点后发布的版本了，并记录依赖的相关部署环境等

2019-01-30

作者回复

嗯，看起来像是代码的上一层的组织方式，不仅包括代码，还包括编译后的文件，以及支持文件。

2019-01-31



Lindroid

👍 1

总结：

README：命名大写字母，放在根目录下，用于介绍软件的使用；

COPYRIGHT：命名大写字母，放在根目录下，用于解释软件的版权；

源代码文件：**src**目录下，可再细分目录；

测试代码：需要和功能代码分开存放，可放在根目录的**test**目录下，一个测试文件最好执行一个独立任务；

文档：可以放在根目录的**docs** 或者 **doc** 的目录下。

2019-04-19



苦行僧

👍 1

一般使用**maven**的工程结构

2019-02-07



Y024

👍 1

作为 **Javaer**，分享一些自己接触到东西。目前很多都有脚手架工具，可以帮你快速初始化/规划项目代码组织，比如：**Java** 里面的 **Maven**，可以帮你快速初始化一个项目（**Maven** 生成的工程目录划分，具体细节移步官方文档：<https://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html>）。

利用脚手架初始化完一个项目之后，涉及到了分层，一般会有类别（**controller/web**、**service**、**dao** 等）、业务模块（**user**、**order** 等）两个维度，看是先按照类别再按业务分层（如：**controller/user**、**controller/order**；**service/user**，**service/order**），还是先按照业务再按照类别分层（**user/controller**，**user/service**；**order/controller**，**order/service**），但一定不要两种风格混合。

2019-01-31

作者回复

好经验，谢谢分享！

类似于**Maven**的脚手架工具很有价值，我也建议大家了解了解。我个人特别喜欢它对代码依赖关系的自动化处理。第二部分，我们会使用一下这个小工具（不会讲，就是用一下）。

2019-01-31



Robic

👍 1

上文图中那个**make/**是什么含义

2019-01-30

作者回复

放置**makefile**的目录

2019-01-30



老吴

👍 0

说到这个，老师能否加开一篇领域驱动的课

2019-02-26

作者回复

这是一个好话题，但是我现在真的不懂领域驱动。要加油学习了！

2019-02-26



pyhhou

👍 0

自己使用的就是最简单的 MVC 的代码结构，写的是 node JS，不需要编译结构也是一些个工具帮忙生成的：

```
bin/  
node_modules/  
public/  
scripts/  
src/  
controllers/  
helpers/  
models/  
routes/  
tests/  
.env  
app.js  
package.json  
package-lock.json
```

这里比较不好的是 **tests** 很难和 **src** 里面的东西一一对应，有时写着写着会把自己绕进去，不知老师有没有什么好的建议。还想请问老师的就是即便是在一个好的文件框架下，很多子文件的层级结构也还是得由我们程序员自己来设定，就比如一个功能可以由一个文件实现，也可以由多个文件协同实现，这里面有没有一些个比较好的实践，或者是可以借鉴的思想方法，使得即便是不看半句代码，光看文件名加文件结构就可以把设计思想以及代码功能看出个十之八九？

2019-02-24

作者回复

对应好源代码和测试案例，是一个省时省力的思路。JDK一般按照源代码的目录结构组织测试。比如，测试接口规范`javax.net.ssl.SSLSocket`的代码，放在`test/jdk/javax/net/ssl/SSLSocket`目录下。测试接口实现`sun.security.ssl.SSLSocketImpl`的代码，放在`test/jdk/sun/security/ssl/SSLSocketImpl`目录下。我们经常需要翻找测试用例，来检查测试覆盖是不是足够，这种对应的组织方式的确省了不少时间。

文件的安排，首先要做好模块的划分，功能类似的、目标类似的放到一个命名空间里（比如Java的包）；然后，分割接口规范和内部实现；接口规范和内部实现放到不同的命名空间里；然后，再划分依赖关系，公开接口的文件，一个类一个文件；内部实现的代码，尽量减少文件间的依赖关系，只被一个类依赖的代码，都放到那个类里去。

比如，`javax.net.ssl`是TLS协议的公开接口规范，单独使用一个公开的包。`sun.security.ssl`是TLS接口规范的一个实现，单独使用另外一个包。`sun.security.ssl.ClientHello`类里，还包含了支撑这个类实现的其他几个内部类。

```
package sun.security.ssl;
```

```
final class ClientHello {  
    static final class ClientHelloMessage extends HandshakeMessage {  
        // snipped  
    }  
  
    private static final  
    class ClientHelloKickstartProducer implements SSLProducer {  
        // snipped  
    }  
    // snipped  
}
```

另外，就是做好命名空间和类的命名，名字清晰了，从文件名和文件结构的确可以看到更多的东西。

2019-02-24



小文

👍 0

很多网上的代码都没有版权说明，那也会受法律保护吗？

2019-02-18

作者回复

是的。这种代码的版权风险很大的，不知期限，不知版权归属，不知许可方式。有些法律默认没有版权说明，就是作者保留所有权利。

2019-02-18