

# 目录

前言	1.1
软件开发的生命周期	1.2
需求	1.3
设计	1.4
生命周期	1.4.1
log日志级别	1.4.2
UUID和GUID	1.4.3
开发	1.5
README文档	1.5.1
行宽	1.5.2
代码注释的格式	1.5.3
调试	1.6
Debug的通用操作逻辑	1.6.1
发布	1.7
版本号命名	1.7.1
不用版本	1.7.2
协议类型	1.7.3
更新日志	1.7.4
发布频率	1.7.5
静态库vs动态库	1.7.6
下载	1.8
不同平台和位宽	1.8.1
历史版本	1.8.2
附录	1.9
参考资料	1.9.1

# 软件全生命周期管理

- 最新版本: 1.0.0
- 更新时间: 20250605

## 简介

介绍软件开发的全生命周期的相同通用知识。包括需求、设计、开发、调试、发布、下载等；其中需求中整理了常见的图解需求的重要性；以及设计中包含生命周期、log日志级别、UUID和UDID等；而开发中包含README文档、行宽、代码注释的格式等；调试中包含Debug的通用操作和逻辑；发布中包含版本号命名规范、不同的版本、开源协议类型、更新日志写法、发布的频率策略、以及静态库和动态库；下载中包含不同的平台和位宽、历史版本等。

## 源码+浏览+下载

本书的各种源码、在线浏览地址、多种格式文件下载如下：

### HonKit源码

- [crifan/software\\_whole\\_lifecycle\\_manage](#): 软件全生命周期管理

### 如何使用此HonKit源码去生成发布为电子书

详见：[crifan/honkit\\_template](#): demo how to use crifan honkit template and demo

### 在线浏览

- [软件全生命周期管理 book.crifan.org](#)
- [软件全生命周期管理 crifan.github.io](#)

### 离线下载阅读

- [软件全生命周期管理 PDF](#)
- [软件全生命周期管理 ePUB](#)
- [软件全生命周期管理 MOBI](#)

## 版权和用途说明

此电子书教程的全部内容，如无特别说明，均为本人原创。其中部分内容参考自网络，均已备注了出处。如发现有侵权，请通过邮箱联系我 [admin 艾特 crifan.com](mailto:admin@crifan.com)，我会尽快删除。谢谢合作。

各种技术类教程，仅作为学习和研究使用。请勿用于任何非法用途。如有非法用途，均与本人无关。

## 鸣谢

感谢我的老婆陈雪的包容理解和悉心照料，才使得我 [crifan](#) 有更多精力去专注技术专研和整理归纳出这些电子书和技术教程，特此鸣谢。

## 其他

## 作者的其他电子书

本人 crifan 还写了其他 150+ 本电子书教程，感兴趣可移步至：

[crifan/crifan\\_ebook\\_readme: Crifan的电子书的使用说明](#)

## 关于作者

关于作者更多介绍，详见：

[关于CrifanLi李茂 – 在路上](#)

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新： 2025-06-05 23:06:33

# 软件开发的生命周期

关于软件开发方面的生命周期，大概有下面这些部分：

- 需求
- 设计
  - 生命周期
- 开发
  - 具体写代码去开发
- 调试
  - 日志
  - 调试通用逻辑
- 发布
  - 开源协议选择
  - 版本管理
  - 更新日志
- 下载
  - 打包不同平台
  - 放在不同镜像中供下载

---

说明：

此处软件是广义上的，包含：

- 一般意义上的普通用户能安装使用的软件
- 也包含普通的各种库library

crifan.org, 使用[署名4.0国际\(CC BY 4.0\)协议](#)发布 all right reserved, powered by Gitbook最后更新：2025-06-05 23:05:08

## 需求

### 要充分理解需求，才能真正做出用户想要的东西

软件开发过程中，最常出现的，也是最严重的问题，也是非常重要一个环节就是：需求即搞清楚客户想要什么，然后多快好省的实现对应功能。

而如果需求理解不充分，则会出现这种情况：

你以为你实现的客户端是很好的，但是用户眼中看到的却是别样感受

即，你以为你理解了客户的需求，其实各自想要的以为东西不一样，有时候还有很大的偏差

以及公司内部不同部门和职位的人员之间的沟通，都会有理解的差异

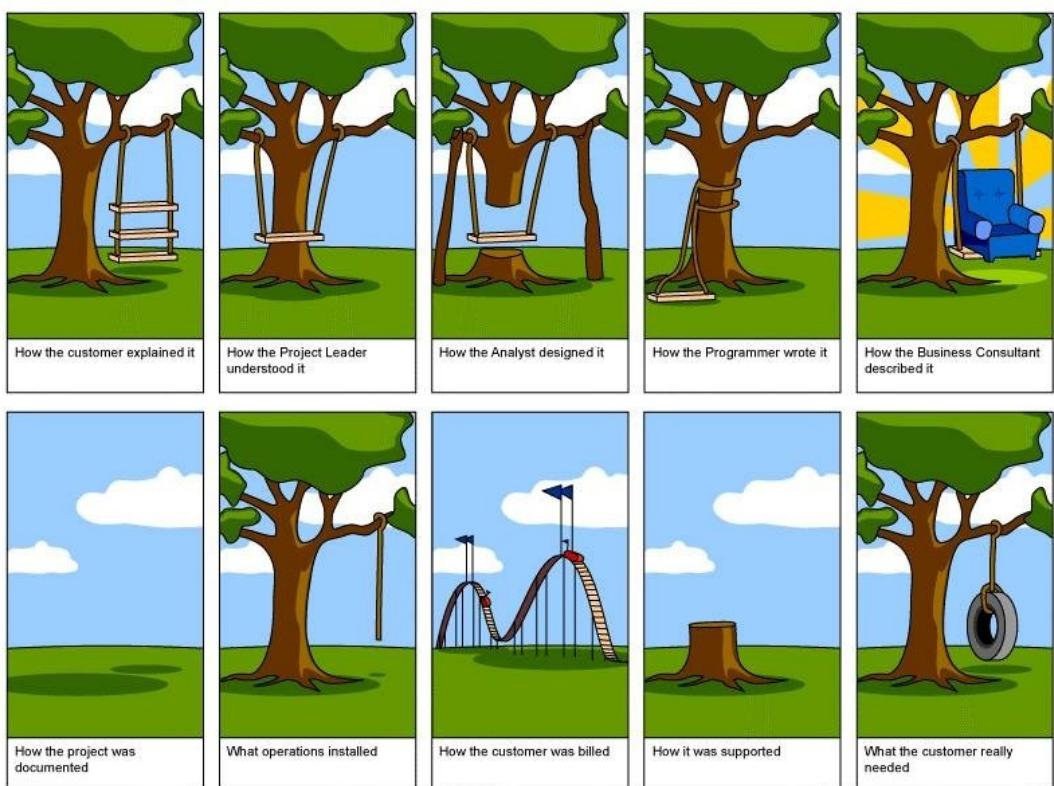
加上具体技术实现和产品原型本身存在差异，就会导致这种不同的理解

网上有很多相关的经典的例子：

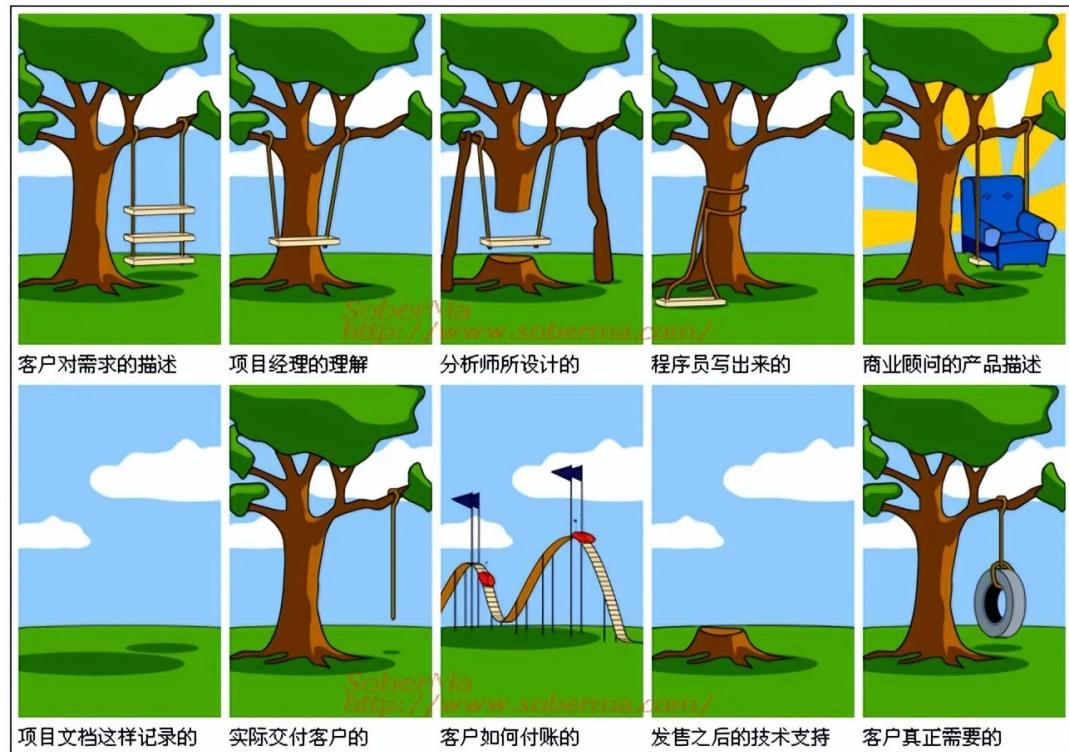
- 不同阶段，对于需求的理解，都不同：简易秋千

◦ 单图

▪ 英文



▪ 中文

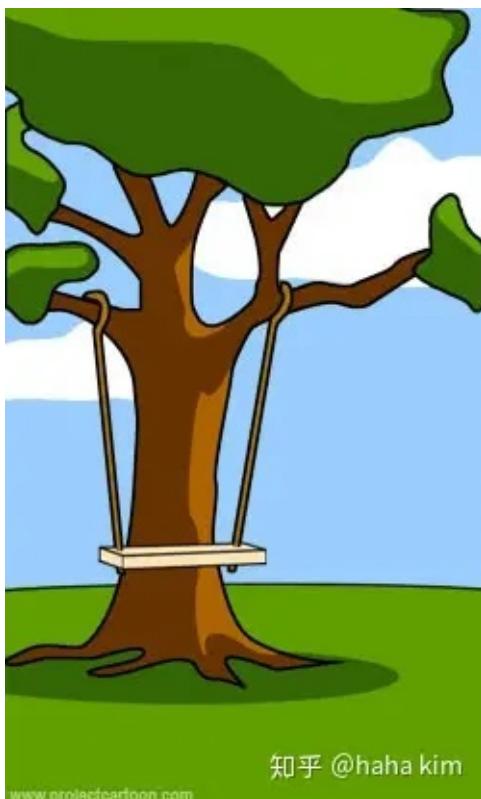


- 多图

- 客户是这样描述需求的



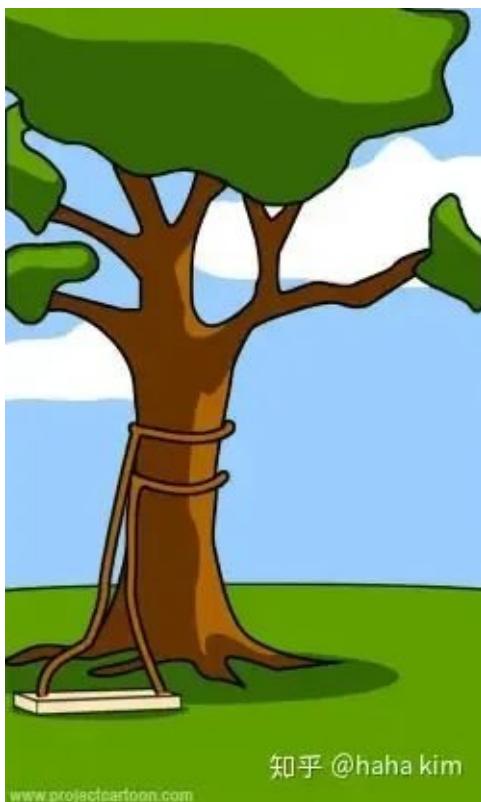
- 项目经理是这样理解的



- 设计出来是这样的



- 开发出来的是这样的



知乎 @haha kim

■ www.projectcartoon.com

- 测试的时候是这样的



知乎 @haha kim

■ www.projectcartoon.com

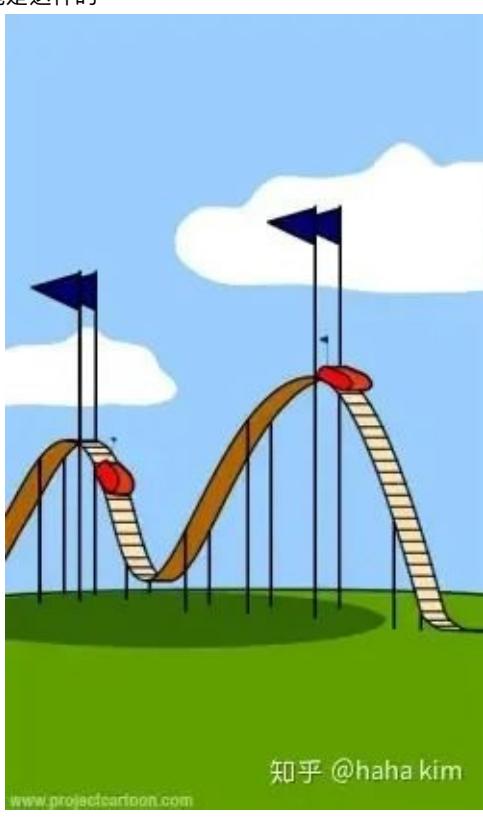
- 顾问是这样形容的



知乎 @haha kim

■ www.projectcartoon.com

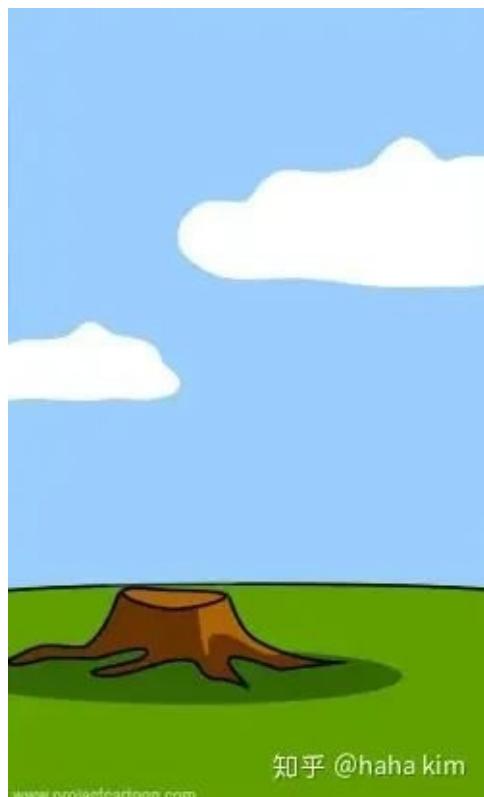
- 实施是这样的



知乎 @haha kim

■ www.projectcartoon.com

- 顾客得到的是这个样子的



知乎 @haha kim

- 顾客真正想要的是这个样子的



知乎 @haha kim

- 你眼中的儿童的有趣的动物玩具，但儿童眼中其实是一堆动物屁股

- - 不同阶段的小龙女的效果

- - 不同阶段对马的需求的理解的差异

◦

-》软件开发中的沟通很重要

-》如果需求都不对，后续代码实现，更不可能对，所谓 上梁不正下梁歪 的感觉。

## 希望实现需求理解充分和一致

-》而我们最希望达到的效果是这种：

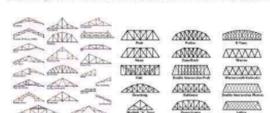
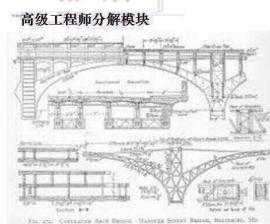
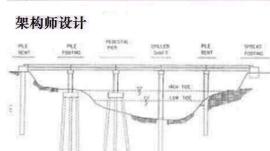
- 不同阶段对需求的理解都很一致

◦

## 开发时间不足，导致结果很差

如果时间被压缩的到不合理程度，会导致结果很差：

- 需求和实现差异很大：一座桥
  -



技术团队时间评估需要1.2年

领导发话：

“互联网精神—先上线，再迭代  
给技术团队一个月的时间造桥”



对外宣讲

## 预算不足，导致结果很差

当然也有预算原因导致的：

- 需求和实现差异很大：马

◦

## 其他相关

### 确保用户使用方式正确

以及，即使你把东西（软件）做出来，但是你的用户（可能）永远不会按照你的想法使用产品的，而是有自己的想法=特殊的使用方式：

- 特殊的使用方式：猫吃猫粮

◦

**系统非常稳定，所有代码不要随便乱动**



系统非常稳定，  
所有代码不要随便动

工作地点的变化



crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新: 2025-06-04 22:49:13

# 设计

软件，代码的框架，逻辑，概念，规则设计期间：

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新：2025-06-04 22:49:43

# 生命周期

很多框架和技术中，都有 `生命周期 = lifecycle = life cycle` 的概念。

## 举例

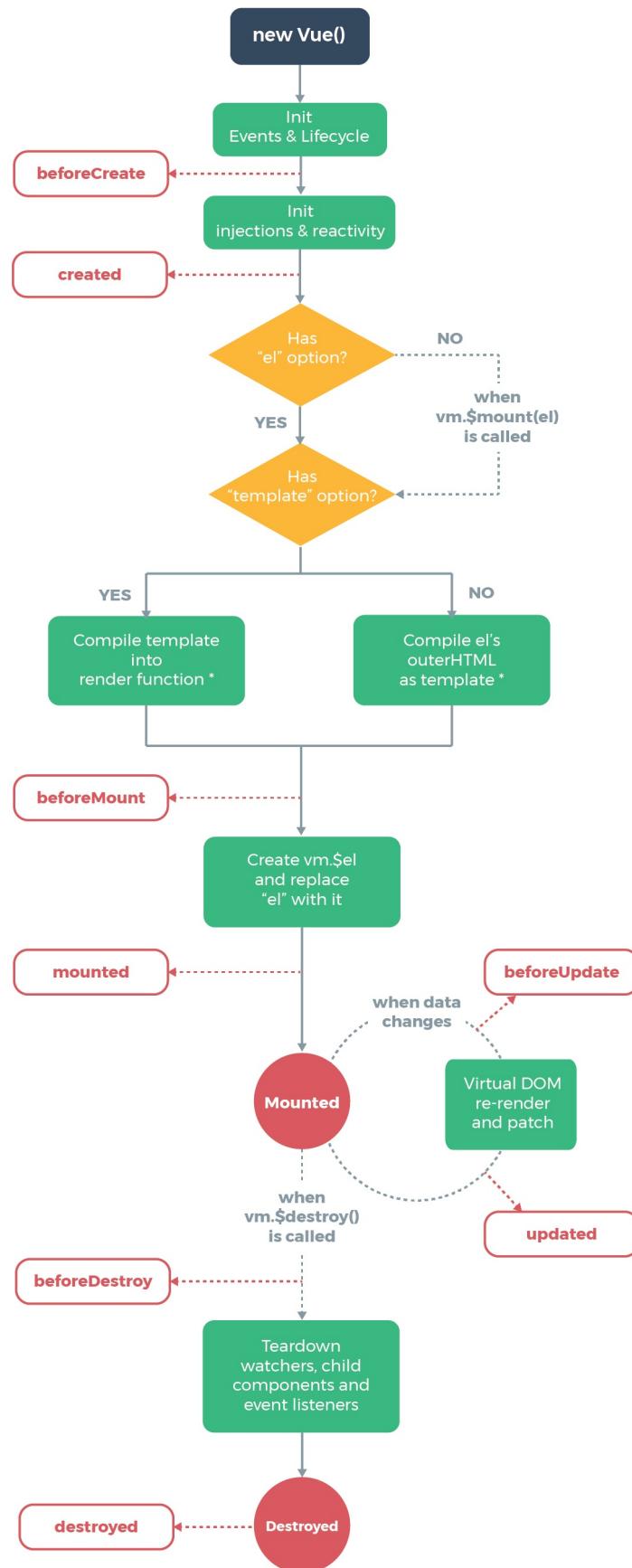
### React的生命周期

- React的生命周期
  - 包含
    - Mounting
      - `constructor()`
      - `getDerivedStateFromProps()`
      - `componentWillMount() / UNSAFE_componentWillMount()`
      - `render()`
      - `componentDidMount()`
    - Updating
      - `componentWillReceiveProps() / UNSAFE_componentWillReceiveProps()`
      - `getDerivedStateFromProps()`
      - `shouldComponentUpdate()`
      - `componentWillUpdate() / UNSAFE_componentWillUpdate()`
      - `render()`
      - `getSnapshotBeforeUpdate()`
      - `componentDidUpdate()`
    - Unmounting
      - `componentWillUnmount()`
    - Error Handling
      - `componentDidCatch()`
  - 详见：
    - State and Lifecycle - React
      - <https://reactjs.org/docs/state-and-lifecycle.html>
    - React.Component - React
      - <https://reactjs.org/docs/react-component.html>

### Vue.js 的生命周期

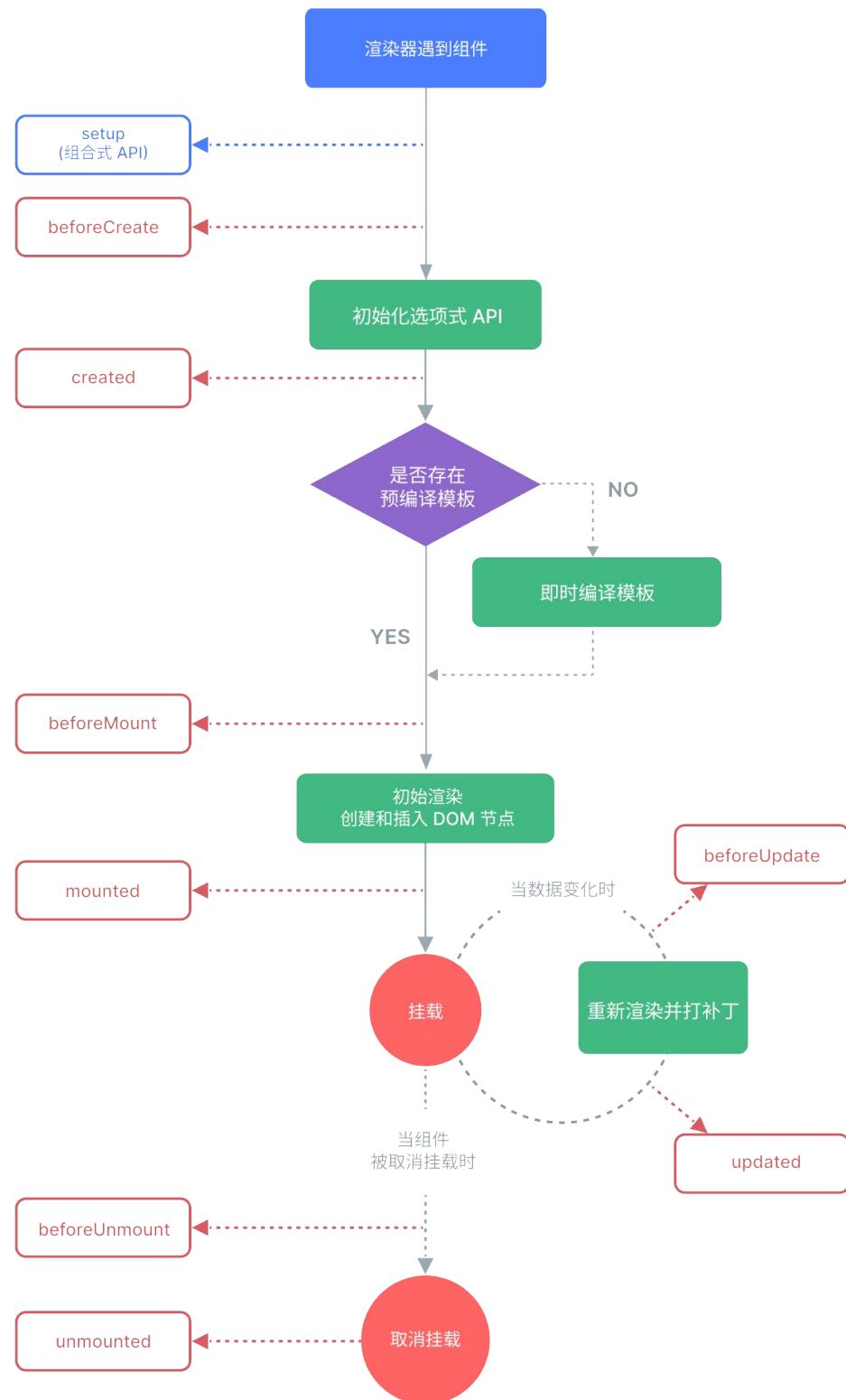
- `vue.js` 的生命周期

- - [Vue.js 实例生命周期的图表](#)
    - 英文



\* template compilation is performed ahead-of-time if using a build step, e.g. single-file components

- 中文



## Page的Lifecycle

- Page Lifecycle

◦

## 生命周期的作用和逻辑

生命周期：

- 就像人的生命，可以（根据不同分类标准，分成）大致几个周期

比如：

根据时间点和年龄段划分，可以分为：

- 出生
- 成长
  - 不同阶段
    - 幼年
    - 青年
    - 中年
    - 老年
  - 异常
    - 疫病
    - 事故
    - 导致受伤、去世
- 死亡

-》

而类似的，比如上述提到的React的生命周期中，对于Component组件来说，也是类似的时间点或时间段去划分的，和人类生命周期有点像的部分周期是：

- Mounting：从出生进入幼年 / 青年 / 中年
  - constructor()：出生阶段开始
  - componentWillMount()：将要出生
  - componentDidMount()：已经出生
- Updating：开始展开/活好自己的一生
  - componentWillReceiveProps()：从外部吸收营养==传递参数进来
  - render()：有参数变化就更新显示 -》 活出自己丰富多彩的一生

- `getSnapshotBeforeUpdate()`
- `componentDidUpdate()`
- `Unmounting` : 人的死亡
  - `componentWillUnmount()`
- `Error Handling` : 人的生病需要关注和处理和治疗
  - `componentDidCatch()`

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新: 2025-06-05 23:05:47

# log日志级别

软件内部要打印log日志，需要设计不同的日志级别。

## 举例

### Scrapy

参考[Logging — Scrapy 0.24.1 文档](#)中就有：

```
Log levels

Scrapy提供5层logging级别：
1. CRITICAL - 严重错误(critical)
2. ERROR - 一般错误( regular errors )
3. WARNING - 警告信息(warning messages)
4. INFO - 一般信息(informational messages)
5. DEBUG - 调试信息(debugging messages)

scrapy.log.CRITICAL
严重错误的Log级别
scrapy.log.ERROR
错误的Log级别 Log level for errors
scrapy.log.WARNING
警告的Log级别 Log level for warnings
scrapy.log.INFO
记录信息的Log级别(生产部署时推荐的Log级别)
scrapy.log.DEBUG
调试信息的Log级别(开发时推荐的Log级别)
```

## Supervisor

supervisor的log的日志级别：

[Logging — Supervisor 3.3.4 documentation](#)

Config File Value	Output Code	Description
critical	CRIT	Messages that indicate a condition that requires immediate user attention, a supervisor state change, or an error in supervisor itself.
error	ERRO	Messages that indicate a potentially ignorable error condition (e.g. unable to clear a log directory).
warn	WARN	Messages that indicate an anomalous condition which isn't an error.
info	INFO	Normal informational output. This is the default log level if none is explicitly configured.
debug	DEBG	Messages useful for users trying to debug process configuration and communications behavior (process output, listener state changes, event notifications).
trace	TRAC	Messages useful for developers trying to debug supervisor plugins, and information about HTTP and RPC requests and responses.
blather	BLAT	Messages useful for developers trying to debug supervisor itself.



# UUID和GUID

内部设计系统时，对于内容的管理，比如用户的id，其策略除了普通的id递增外，往往也会采用uuid。

举例：

之前用于用户的id，采用了 `UUID`，其中一个用户的id是：

```
user-4045bf61-9075-4403-b614-0aaaa7d19418
```

## 好的设计原则，设计规则

设计模式基础-SOLID

- `s` 代表着单一职责原则
  - <https://realm.io/cn/news/donn-felker-solid-part-1/>
- `o` 代表着开闭原则
  - <https://realm.io/cn/news/donn-felker-solid-part-2/>
- `L` 代表着里氏替换原则
  - <https://realm.io/cn/news/donn-felker-solid-part-3/>
- `I` 是接口隔离原则
  - <https://realm.io/cn/news/donn-felker-solid-part-4/>
- `D` 代表依赖倒置原则
  - <https://realm.io/cn/news/donn-felker-solid-part-5/>

crifan.org，使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新：2025-06-04 23:07:05

# 开发

软件总体逻辑设计完毕后，进入实际的开发期间，一些常见的规则和逻辑：

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新：2025-06-05 21:28:21

# README文档

折腾嵌入式开发时遇到很多项目源码根目录中都有 `README`

折腾web领域很多库的源码根目录中也有一个：`README` 或 `README.md`

## README的作用

- `README` = `read me` =读我= (项目作者希望你在使用其项目之前) 读我 (这个说明文件) = (项目/代码的基本的) 使用说明 (+版权声明等等)

换句话说：你使用该项目/库/代码 之前，最好要好好读一读。

-》可以了解到项目的背景，来源，版权声明，等等信息。

而README的格式，一般就是普通的文本文件。

如果是 `README.md`，则是[Markdown](#)的格式，更加易写易读。

## README的支持

对于 `md` 格式的 `README.md`，很多代码托管平台，比如：

- [github.com](#)
- [gitee.com](#)

都默认自带支持，在web端打开项目首页时，自动加载 `README.md`，显示出友好的格式化后的内容：

- [gitee.com](#) 显示 `README.md` 的效果

◦

- [github.com](#) 显示 `README.md` 的效果
  - [https://github.com/crifan/crifan\\_ebook\\_readme](https://github.com/crifan/crifan_ebook_readme)

The screenshot shows a GitHub repository page for 'crifan\_ebook\_readme'. The 'Branch: master' dropdown is set to 'master'. There are buttons for 'New pull request', 'Create new file', 'Upload files', 'Find file', and 'Clone or download'. A red box highlights the 'README.md' file in the list of files. The file details are as follows:

File	Commit	Date
.gitignore	1. add html and pdf	17 days ago
Crifan的电子书的使用说明.pdf	1. rename pdf name	17 days ago
README.html	1. update content	17 days ago
README.md	1. update content	17 days ago

Below the file list, the content of the README.md file is displayed:

## Crifan的电子书的使用说明

### 为何要有：Crifan的电子书

我之前折腾过很多方面的内容，包括技术类的，非技术类的。

关于技术类中又分很多种，其中都放到了我的网站 crifan.com 上，目前已有7000+的技术帖子，但多数都是零散的某个细节的知识点。

当我在某个技术领域有了一定的心得和经验后，就会抽空整理成相对成系统的内容，用(`Docbook`、`Gitbook`等)工具去变成电子书，方便别人参考。

### 为何要有：Crifan的电子书的使用说明

对于这些电子书，之前虽然已经分类列出来了，但是缺少了基本的介绍，所以对于很多人来说，不方便快速找到自己所需要看哪些。

换句话说：

如果你想要给项目写基本的说明和注意事项等内容，则可以在项目根目录中，填写 `README.md` 文件，然后用 `markdown` 格式去写内容，上传到 `gitee.com` 或 `github.com` 后，页面上自动帮你生成项目说明文档了。很是方便。

crifan.org, 使用[署名4.0国际\(CC BY 4.0\)协议](#)发布 all right reserved, powered by Gitbook最后更新：2025-06-05 21:36:23

# 行宽

- 行宽 = line width
  - 一般建议是: 80个字符
    - 根据[这里](#), 也可以改为 100 或 120 , 甚至 150

crifan.org, 使用[署名4.0国际\(CC BY 4.0\)协议](#)发布 all right reserved, powered by Gitbook最后更新: 2025-06-05 21:37:52

## 代码注释的格式

写软件代码期间：

各种语言都有自己的常见的写注释的约定俗成的方式：

有了大家一起都常用的，都遵守的格式，作用是：

就可以通过第三方的工具，去将代码中的注释提取出来，形成代码的文档

即：从代码中生成文档

比如：

- Java
  - javadoc
- Python
  - doc string
- Xcode
  - 和普通的C/C++类似的函数注释
- R
  - 从：
  - [childe-db: API](#)
  - 知道有：
    - [r-lib/pkgdown: Generate static html documentation for an R package](#)
    - [Make Static HTML Documentation for a Package • pkgdown](#)

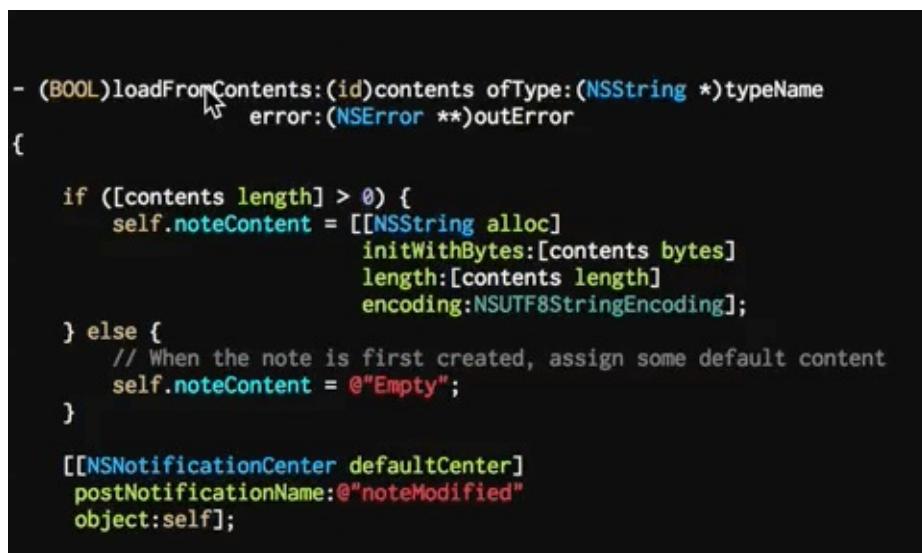
## 举例

### Xcode

有对应的插件，帮你自动生成格式：

给Xcode添加支持给iOS代码写注释和文档

[GitHub - onevcat/VVDocumenter-Xcode: Xcode plug-in which helps you write documentation comment easier, for both Objective-C and Swift.](#)



A screenshot of the Xcode code editor showing annotated Objective-C code. The code is annotated with various symbols and descriptions, such as @param, @return, and @note, indicating specific parts of the code like parameters, return values, and notes. The annotations are color-coded and placed directly next to the corresponding code lines.

```
- (BOOL)loadFromContents:(id)contents ofType:(NSString *)typeName
{
    if ([contents length] > 0) {
        self.noteContent = [[NSString alloc]
            initWithBytes:[contents bytes]
            length:[contents length]
            encoding:NSUTF8StringEncoding];
    } else {
        // When the note is first created, assign some default content
        self.noteContent = @"Empty";
    }

    [[NSNotificationCenter defaultCenter]
    postNotificationName:@"noteModified"
    object:self];
}
```

```
static var searchBundle: NSBundle = NSBundle.mainBundle()

init(frames: [Frame], size: CGSize, scale: CGFloat, bitDepth: Int, repeatCount: Int, firstFrameHidden: Bool) {
    self.frames = frames
    self.internalSize = size
    self.scale = scale
    self.bitDepth = bitDepth
    self.repeatCount = repeatCount
    self.firstFrameHidden = hidden
    dataOwner = nil
}

init(apng: APNGImage) {
    // The image init from this method will share the same data trunk with the other apng obj
    dataOwner = apng

    self.bitDepth = apng.bitDepth
    self.internalSize = apng.internalSize
    self.scale = apng.scale
    self.repeatCount = apng.repeatCount
    self.firstFrameHidden = apng.firstFrameHidden
}
```

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新: 2025-06-05 21:40:56

# 调试

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新: 2025-06-04 22:20:18

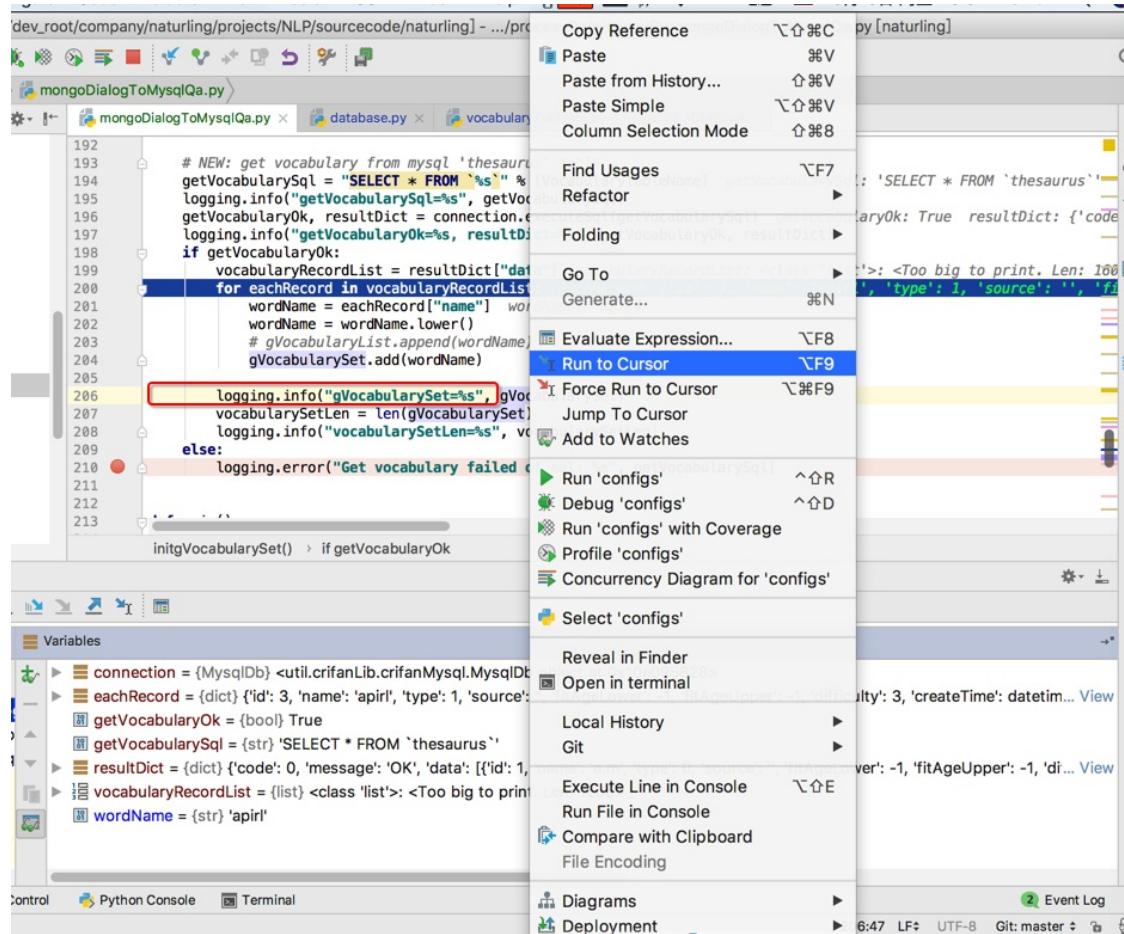
# Debug的通用操作逻辑

关于调试代码、程序时，有些通用的操作和含义，现解释如下：

## Debug的通用操作

常见的和调试Debug有关的操作：

- 开始调试
- 继续运行 = Continue：一直运行，知道代码运行完毕了，或者遇到了断点而停下来
- Stop：停止运行
- Step Over：单步执行
  - 如果不希望进入每行代码内部，比如函数的内部，往往最常用这个单步执行
- Step Into：单步执行，但是遇到函数的话，就进入函数内部执行
- Step Out：从函数内部跳出来
- Run To Cursor：运行到鼠标位置
  - 指的是，鼠标点击到某行代码后，然后让代码开始运行，直到鼠标所在的行，再停下来
  - 比如，调试到一段for循环，调试了前几次循环，觉得没问题了，接下来的循环此处就不需要继续调试了，希望运行到for循环后面的代码，则可以鼠标点击到对应行，然后用Run to Cursor：



- 即可直接（运行完剩下的循环）跳出for，到对应的行：

```

192
193     # NEW: get vocabulary from mysql 'thesaurus' table
194     getVocabularySql = "SELECT * FROM `%(VocabularyTableName)s`" % (VocabularyTableName)
195     getVocabularySql: 'SELECT * FROM `thesaurus`'
196     logging.info("getVocabularySql=%s", getVocabularySql)
197     getVocabularyOk, resultDict = connection.executeSql(getVocabularySql)
198     getVocabularyOk: True
199     resultDict: {'code': 0, 'message': 'OK', 'data': [...] }
200     logging.info("getVocabularyOk=%s, resultDict=%s", getVocabularyOk, resultDict)
201     if getVocabularyOk:
202         vocabularyRecordList = resultDict["data"]
203         for eachRecord in vocabularyRecordList:
204             eachRecord: {'id': 1608, 'name': 'wise', 'type': 3, 'source': '', ...}
205             wordName = eachRecord["name"]
206             wordName = wordName.lower()
207             # gVocabularyList.append(wordName)
208             gVocabularySet.add(wordName)
209     else:
210         logging.error("Get vocabulary failed of sql: %s", getVocabularySql)
211
212
213
    initgVocabularySet() > if getVocabularyOk
  
```

tables

- connection = {MysqliDb} <util.crifanLib.crifanMysqli.MysqliDb object at 0x10e095828>
- eachRecord = {dict} {'id': 1608, 'name': 'wise', 'type': 3, 'source': '', 'fitAgeLower': -1, 'fitAgeUpper': -1, 'difficulty': 0, 'createTime': dat... View
- getVocabularyOk = {bool} True
- getVocabularySql = {str} 'SELECT \* FROM `thesaurus`'
- resultDict = {dict} {'code': 0, 'message': 'OK', 'data': [...] }
- vocabularyRecordList = {list} <class 'list'>: <Too big to print. Len: 1608> View
- wordName = {str} 'wise'

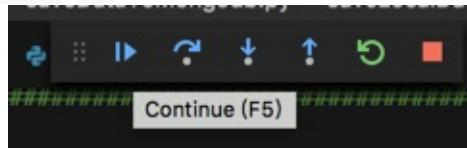
## VSCode

[【已解决】VSCode中调试Python代码](#)

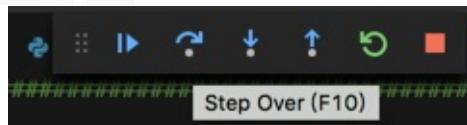
的按钮中的解释，很形象：

- VSCode中的Deubug通用逻辑

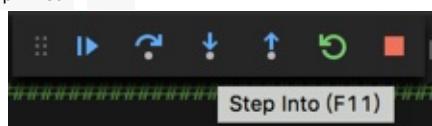
- Continue = F5 : 继续运行 -》 直到运行完毕或者遇到断点停下来



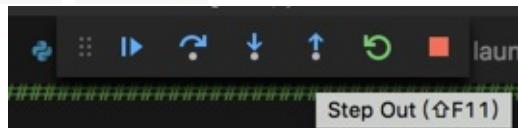
- Step Over = F10



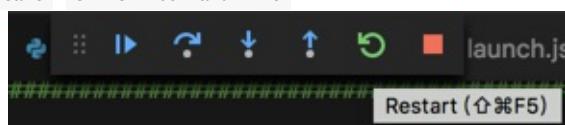
- Step Into = F11



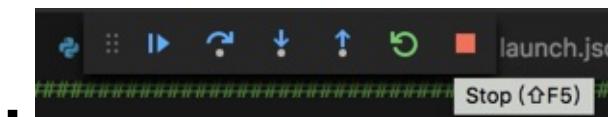
- Step Out = Shift + F11



- Restart = Shift + Command + F5



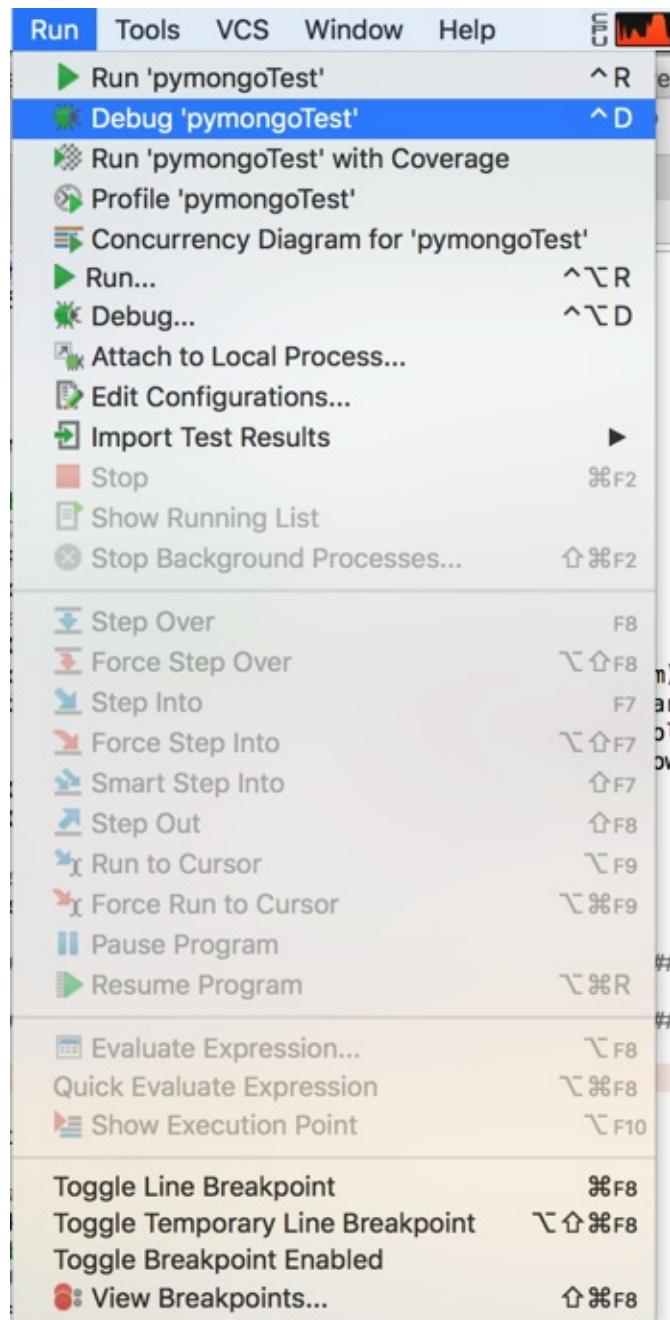
- Stop = Shift + F5



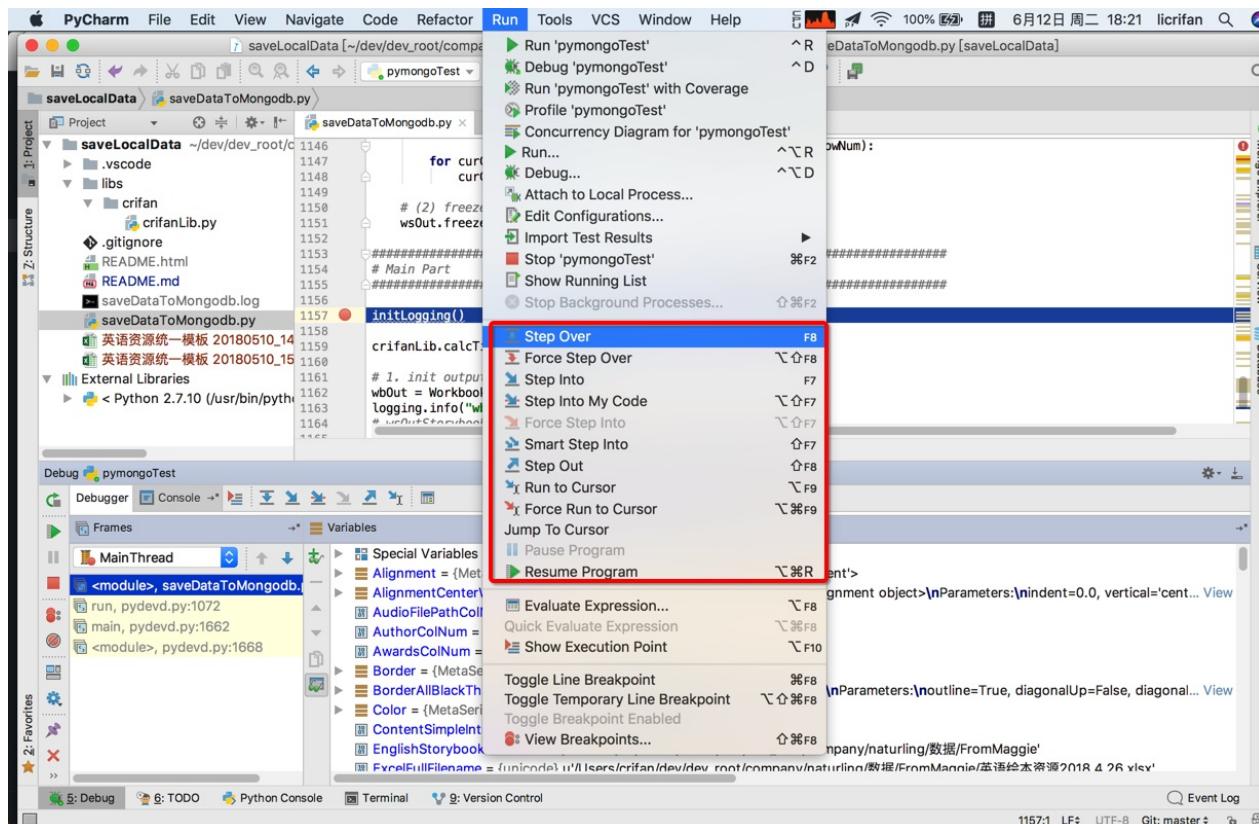
## PyCharm

比如：

PyCharm中的按钮： Command+D 开始 Debug 后：



接着就可以看到各个Debug调试按钮可用了：



## Watch监视

监视：有啥作用？何时使用？

举例说明：

折腾：

【未解决】Python的wda中从搜索结果的页面的xml源码中解析提取公众号的中文名称

期间，本以为：

```
idParentPrev = idParent previous_sibling
```

就是我们希望的元素：

```
<XCUIElementTypeOther type="XCUIElementTypeOther"
    enabled="true" visible="true" x="88" y="120"
    width="271" height="23">
    <XCUIElementTypeStaticText
        type="XCUIElementTypeStaticText"
        value="动卡空间" name="动卡空间" label="动卡空间"
        enabled="true" visible="true" x="88" y="120"
        width="70" height="22" />
</XCUIElementTypeOther>
```

结果实际上却是：

```
\n
```

即一个空行，不是我们要的内部有公众号中文名的XCUIElementTypeOther

此时就希望：能尽快搞清楚，具体代码应该怎么写，才能获取到上面的真正希望的节点

同时却又不希望重新启动调试去测试不同的代码的效果

此处，就可以利用上：监视 的功能了

即：想办法去写一些简答的代码，去看看获取到的节点，是不是我们要的

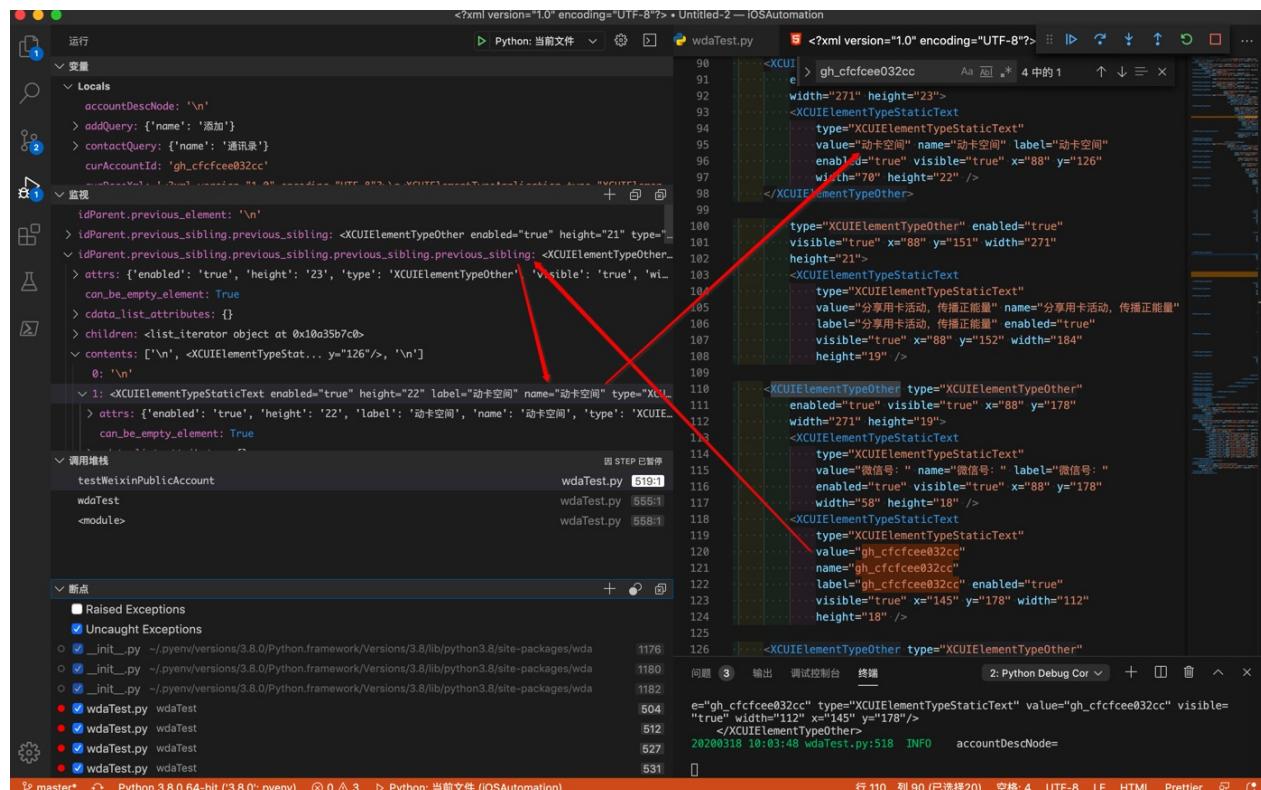
此处，对于BeautifulSoup来说，就可以继续用previous\_sibling

通过尝试发现：

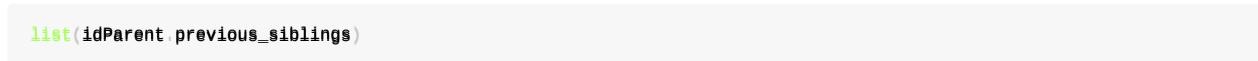
```
idParent.previous_sibling.previous_sibling.previous_sibling.previous_sibling
```

The screenshot shows the PyCharm debugger interface with the 'wdaTest.py — iOSAutomation' file open. The 'Watch' tab is selected, displaying a complex expression: `idParent.previous_element: '\n' > idParent.previous_sibling.previous_sibling: <XCUIElementTypeOther enabled="true" height="21" type="..." > idParent.previous_sibling.previous_sibling.previous_sibling.previous_sibling`. This expression is highlighted in blue, indicating it is being evaluated. The right side of the screen shows a vertical list of line numbers from 504 to 525, with several red dots and a yellow dot highlighting specific lines, likely corresponding to the stack frames or breakpoints in the code.

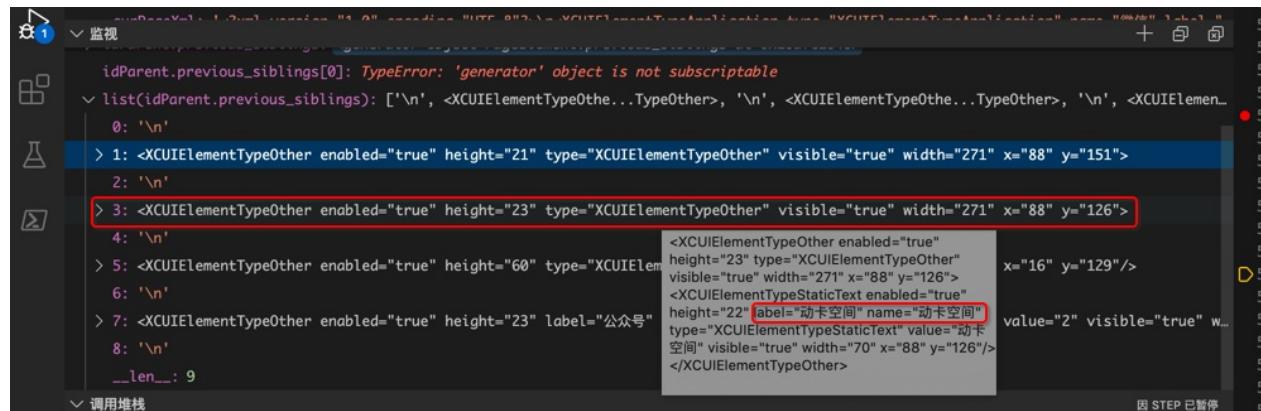
即4次的前一个兄弟节点，才找到我们要的内容：



以及想要实现更好的定位逻辑，经深入分析后，发现：



中的index=3，就是我们要找的元素：



由此实现了我们的目的：搞清楚代码怎么写，才能定位找到我们要的节点

引申：

在合适的时候，通过监视，输入代码（表达式），即可实时看到输出结果

-> 从而实现实时的调试（虽然相对有限的）代码，实现你要的效果：比如此处是找到我们要的节点

其他典型情况：输出变量的值，看看是否符合你的预期等等

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新：2025-06-05 21:58:47

# 发布

除了普通的版本号和版本策略外，往往还有很多不同类型的版本：

作为独立的软件产品的版本设计策略

crifan.org, 使用[署名4.0国际\(CC BY 4.0\)协议](#)发布 all right reserved, powered by Gitbook最后更新：2025-06-05 21:59:26

# 版本号命名

## 版本号命名规范

软件开发期间的版本号命名，往往都用这个标准：

- semver标准
  - [Semantic Versioning 2.0.0 | Semantic Versioning](#)

概述起来就是：

- 三位数字：`x.x.x = major . minor . misc`
  - major=主版本号：主要更新，新功能
  - minor=次版本号：次要更新，次功能
  - misc=revision=细节更新

比如：

每次发版本的app，如果只是小功能更新，则只是第二位加1（或根据功能数量决定），否则大的版本的改进，则是第一位加1

详细解释：

Given a version number MAJOR.MINOR.PATCH, increment the:

- MAJOR version : when you make incompatible API changes,
- MINOR version : when you add functionality in a backwards-compatible manner, and
- PATCH version : when you make backwards-compatible bug fixes.

Additional labels for pre-release and build metadata are available as extensions to the MAJOR.MINOR.PATCH format.

中文版翻译是：

### 语义化版本 2.0.0

版本格式：主版本号.次版本号.修订号，版本号递增规则如下：

1. 主版本号：当你做了不兼容的 API 修改，
2. 次版本号：当你做了向下兼容的功能性新增，
3. 修订号：当你做了向下兼容的问题修正。

先行版本号及版本编译信息可以加到“主版本号.次版本号.修订号”的后面，作为延伸。

现在个人的常见做法是：

1. 主版本：有重大功能更新
  - 比如多加了一个大的功能模块
2. 次版本：有一些重要更新
  - 比如部分功能有重大优化
3. 补丁版本/小版本：细节的优化
  - 比如一些小功能的优化，修复了一些小bug等等

## 版本发布的一些实践和做法

对于发布新版本，去写更新日志时，其形式可以借鉴[jeesite](#)的做法：

### /// V1.0.3

Release Date: Friday, April 19, 2013

改进 问题 添加批量保存菜单排序功能

改进 实体增加@DynamicInsert和@DynamicUpdate注解，只插入或更新需要更新的字段



Opened by think-gem, issue closed on Friday, May 31, 2013

#### // 实体增加@DynamicInsert和@DynamicUpdate注解，只插入或更新需要更新的字段

改进 使用ERMaster数据建模，DbUnit数据文件将xml类型更改为xls类型，分模块存放

改进 界面改为紧凑风格

改进 增加单元测试公共基类；Hibernate升级到4.2；Hibernate Validator升级到5.0.1

改进 系统缓存调成，减少cms与sys模块的耦合。

新增 增加DataEntity数据实体，包含：备注、创建者、创建时间、更新者、更新时间、删除标记字段

新增 增加sql server驱动及建表脚本

新增 集成Activiti5.12工作流引擎，提供简单请假的例子

新增 新增数据集权限控制

新增 登录3次错误后，输入验证码

缺陷 评论模块，查看文档

其效果很醒目，且点击每条可以展开显示详情

但是建议关键字还是用个人目前常用的做法：

- 新增：增加了新功能
- 修复：修复了之前存在的bug和问题
- 优化：改进，优化了已有的东西

# 不用版本

很多软件发布后，往往会区分出多个版本：

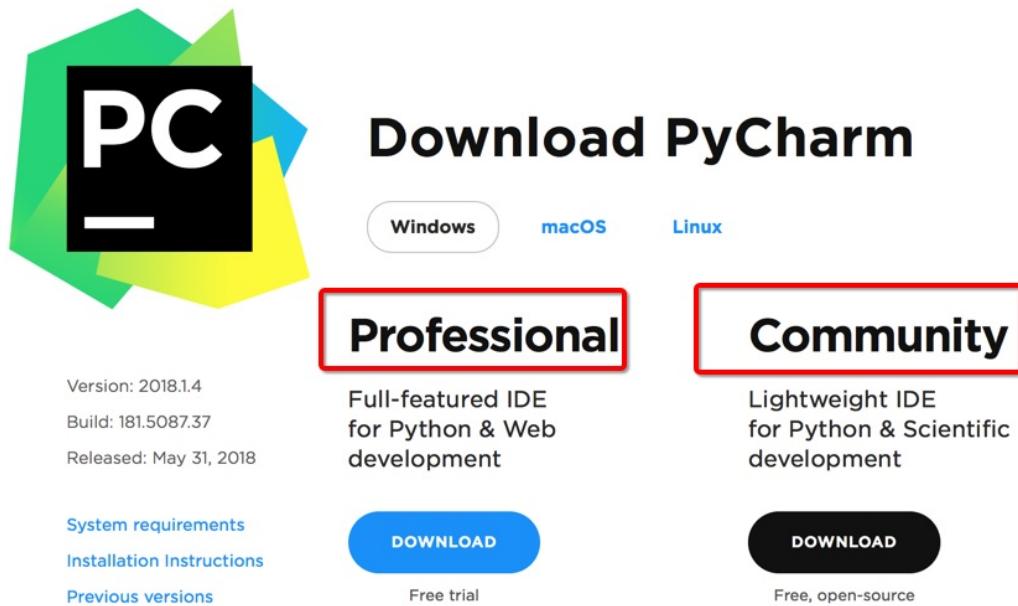
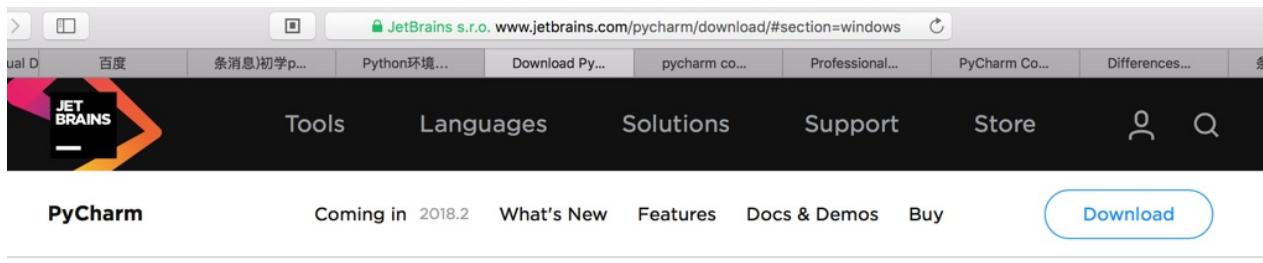
- 社区版 = Community Version = 免费版
  - 免费给（自己的）社区（的用户）使用，一般来说功能对于普通人够用了
- 专业版 = Professional Version = 收费版
  - 要花钱去买专业版才能用上包含了全部高级功能的版本

## 举例

### PyCharm

PyCharm中也是：

[Download PyCharm: Python IDE for Professional Developers by JetBrains](#)



- Professional : Full-featured IDE for Python & Web development
- Community : Lightweight IDE for Python & Scientific development

详见：

[版本选择 · 最智能的Python的IDE：PyCharm](#)

## Mongo Management Studio

MMS = Mongo Management Studio 中也是

Mongo Management Studio - the professional MongoDB GUI

The screenshot shows a web browser window for the Mongo Management Studio website. The URL bar shows 'mms.litixsoft.de/#'. The page has a navigation bar with links like '\$meta', 'Metadata an...', 'MongoDB as...', 'Consider Mo...', 'Adding a fiel...', 'mongofiles...', 'mongo comp...', 'Creating Co...', '20 Best Mon...', 'Mongo Mana...', and a search bar with 'gridfs' and a search icon. Below the navigation is a green header bar with 'Mongo Management Studio' and a menu with 'LITIXSOFT', 'TOP', 'FEATURES', 'VERSIONS', 'DOWNLOAD', 'TECHNOLOGIES', 'OTHER PRODUCTS', 'CONTACT', 'DOCS', 'BLOG', and language icons for German and English. The main content area has a heading 'Purchase and download'. A sub-instruction says 'Please choose a version and then press the purchase button. In the following pop-up, please select your operating system, you are then forwarded to our store.' There are three main pricing options:

Community (1, 3)	Professional (1)	Enterprise (2)
no support included, may not be used for professional applications <b>for FREE</b> incl. 1 user licence <a href="#">download</a>	incl. 1 year support and updates <b>25€</b> incl. 1 user licence <a href="#">purchase</a>	incl. 1 year support and updates <b>65€</b> incl. 5 user licences <a href="#">purchase</a>

<sup>1</sup> These versions are installed locally on your computer.

## Gitlab

- Gitlab的不同版本
  - 社区版 ( CE = Community Edition )
  - 企业版 ( EE = Professional Edition )

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新: 2025-06-05 22:17:07

## 不同协议类型的选择

- 不同协议类型的选择 = 如何选择 开源 协议

软件发布时，往往会根据自身情况，选择合适的协议

如果是开源的，往往选择某个开源协议：

可以参考：

- [Choose an open source license | Choose a License](#)
- [Licenses | Choose a License](#)

以及：

- [开源协议 整理 对比](#)

◦

◦

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新: 2025-06-05 22:22:46

# 在发布新版之前，更新日志如何写

对于提交代码，提交产品更新日志等，如何写注释：

其逻辑和Git中代码改动后的思路是一致的：

改动可以分为几种：

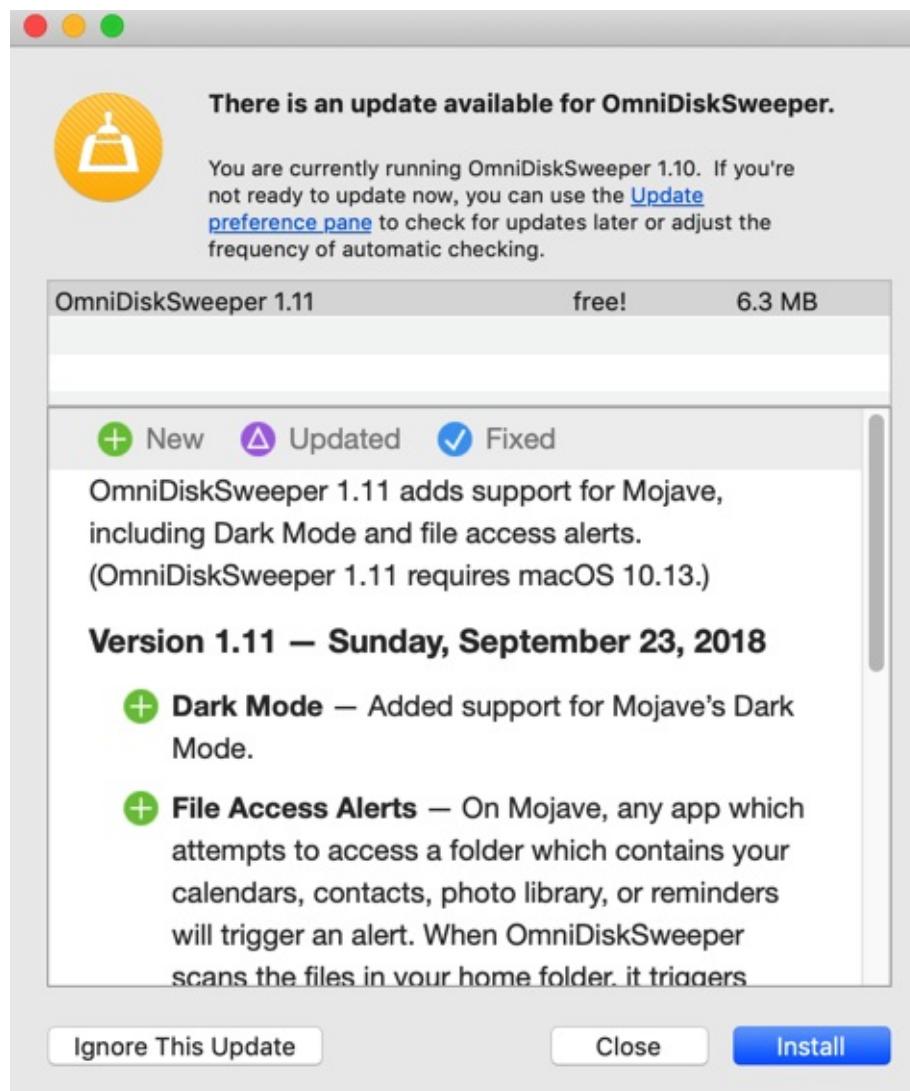
- 新增 = Add = New : 新增新的功能，需求
- 优化 = Optimize = Update : 或叫做 更新，对于现存问题，做了哪些改动，目的是：
  - 改动内部代码逻辑，变得更好
  - 优化现有做法，变得更好
  - 等等
- 修复 = Fix = Fixbug = BugFix : 之前存在的问题，被修复了
  - 典型的之前存在某bug被fix了

## 举例

### git的代码更新

- 概述
  - 文件未追踪 (Untracked)
  - 新文件 (Added, Staged)
  - 文件有修改 (Modified)
  - 文件有修改 (Modified, Staged)
  - 文件有冲突 (Conflict)
  - 文件被删除 (Deleted)
- 详见
  - [相关支持 · 最流行的版本控制系统：Git](#) 中的，各种不同标记，用于表示代码改动的类型

### 某软件：OmniDiskSweeper



crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新: 2025-06-05 22:31:41

## 版本发布频率的策略

### Nightly build 每日编译

- Nightly build = 每日编译 = 每日发布
  - 含义
    - 对于很多软件，库的发布策略，除了发布最近的稳定版之外，往往还会提供一个，给喜欢尝鲜用户的最新的，每天都（或者不定期就）会更新的版本
      - 这种每天都会（往往是用构建系统去自动）编译并发布的版本，叫做：Nightly build
  - 与之相对应的是：
    - Stable version = 稳定版本 = 最新的稳定版本
      - = release version = 已发布版本
        - 因为只有充分测试了，没有大的问题了，才会，才敢，去发布
  - 特点
    - 往往每天都会更新
      - 经常会包含新加入的最新的功能
    - 不像稳定版那样经过充分的测试
      - 有时候会有小bug或问题
      - 官网往往希望你遇到问题就及时反馈
        - 便于官网及时修复bug，促进后续发布更新的稳定版本

### 举例

#### FFmpeg

比如：

[强大的音视频处理工具：FFmpeg](#)

中的：

[【已解决】mac中下载和安装最新版的ffmpeg](#)

时遇到的：

下载ffmpeg库时：

[Builds - Zeranoe FFmpeg](#)

提供了：

- 某个最新的稳定的版本：4.1.1
  - 图

Version	Architecture	Linking
20190318-15d016b 4.1.1	Release builds are recommended for distributors, but cannot be used when submitting bugs.	Static Shared Dev

Download Build

- 说明
  - stable version: Release builds are recommended for distributors, but cannot be used when submitting bugs
    - 已发布版本 = 稳定版
    - 一般意味着很稳定
    - 没有大的问题
    - 不可用于提交bug问题
- Nightly build的版本: 20190318-15d016b
- 图

Version	Architecture	Linking
20190318-15d016b 4.1.1	Nightly git builds contain more features, are usually stable, and are the required version when submitting bugs.  macOS 64-bit	Static Shared Dev

Download Build

- 说明
  - Nightly build: Nightly git builds contain more features, are usually stable, and are the required version when submitting bugs
    - 每日编译的版本
    - 往往包含更多的功能，经常也是很稳定的；但是不是绝对的，偶尔也会包含小bug问题
    - 可用于提交bug问题

# 静态库vs动态库

- 静态库 = static library = static lib
- 动态库 = dynamic library = dynamic lib = dylib
  - 静态还是动态，是属于程序运行时，去加载（其所依赖的，第三方的，其他的）库时的方式：
    - -》属于程序运行的linking链接阶段
    - -》所以： 动态库 = dll = dynamic link library = 动态链接库
  - 加载别的库时，如果是动态方式，那么被加载的第三方库，可被多个其他程序所加载，多个程序共享此库
    - -》所以： 动态库 = 共享库 = shared library = shared lib

一般来说，能谈到库的加载是动态还是静态，往往是属于程序（app，软件等）的内部运行机制中的编译和链接的方面的领域，属于典型的编译原理方面的知识

-》普通软件使用者，往往不需要关心

-》作为程序开发人员，研究程序运行机制和原理的人，往往才需要关心

## 举例

### ffmpeg

比如：

[强大的音视频处理工具：FFmpeg](#)

中：

【已解决】mac中下载和安装最新版的ffmpeg

时遇到的：

下载ffmpeg库时，Linking提供static和shared的方式：

[Builds - Zeranoe FFmpeg](#)

- static: The recommended default build. No dll or dylib files
  - 图

Version	Architecture	Linking
20190318-15d016b 4.1.1	The recommended default build. No dll or dylib files. Windows 32-bit macOS 64-bit	Static Shared Dev

[Download Build](#)

- - 说明
    - 不依赖于dll或动态库文件
- shared: Executables that depend on the included dll or dylib files
  - 图

Version	Architecture	Linking
20190318-15d016b 4.1.1	Windows 64-bit Executables that depend on the included dll or dylib files. macOS 64-bit	Static Shared Dev
		<a href="#">Download Build</a>

- 说明
  - 依赖于所包含的dll或动态库文件

## 下载

虽然和软件发布关系很密切，但是有些内容更主要针对的是，软件发布后，如何去供用户和别人去下载方面的事情：

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新： 2025-06-05 22:55:27

# 不同平台和位宽

针对不同平台和位宽不同，往往有不同版本的

- 平台
  - Windows = Win
  - Mac
  - Linux
    - 各种发行版本 Distribution
- 位宽
  - 32位 = 32 bit = x86
  - 64位 = 64 bit = x64

## 举例

### ffmpeg

比如：

强大的音视频处理工具：[FFmpeg](#)

【已解决】mac中下载和安装最新版的ffmpeg

时遇到的：

下载ffmpeg库时：

[Builds - Zeranoe FFmpeg](#)

Version	Architecture	Linking
20190318-15d016b	Windows 64-bit	Static
4.1.1	Windows 32-bit	Shared
	macOS 64-bit	Dev

[Download Build](#)

- Win
  - X64 = 64bit : Windows 64 bit
  - x86 = 32bit : Windows 32 bit
- Mac
  - macOS 64 bit
    - 注：最新版本的macOS，只有 x64，没有 x86 版本
    - 所以此处没有发布 macOS 32 bit



## 历史版本=旧版本

除了发布最新版本，往往还有历史版本，旧版本供下载

### 举例

#### ffmpeg

比如：

[强大的音视频处理工具：FFmpeg](#)

中：

【已解决】mac中下载和安装最新版的ffmpeg

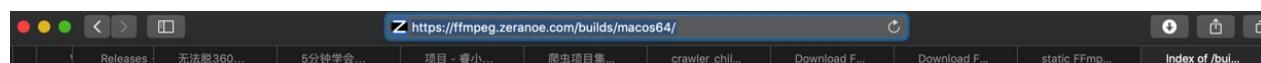
时遇到的：

[Builds - Zeranoe FFmpeg](#)

-» 旧版本的ffmpeg：

The screenshot shows a web page titled "Other Downloads". Below it is a section titled "Previous Builds" with three buttons: "Windows 64-bit Builds", "Windows 32-bit Builds", and "macOS 64-bit Builds". The "macOS 64-bit Builds" button is highlighted with a red border.

[Index of /builds/macos64/](#)



**Index of /builds/macos64/**

<a href="#">File Name ↓</a>	<a href="#">File Size ↓</a>	<a href="#">Date ↓</a>
<a href="#">Parent directory/</a>	-	-
<a href="#">dev/</a>	-	2019-Mar-18 08:53
<a href="#">shared/</a>	-	2019-Mar-18 08:53
<a href="#">static/</a>	-	2019-Mar-18 08:42

-»

[Index of /builds/macos64/static/](#)

<a href="#">Releases</a>	<a href="#">无法脱360...</a>	<a href="#">5分钟学会...</a>	<a href="#">项目 - 首页</a>	<a href="#">爬虫项目集...</a>	<a href="#">crawler_chil...</a>	<a href="#">Download F...</a>	<a href="#">Download F...</a>	<a href="#">static FFmp...</a>	<a href="#">Index of /bu...</a>	<a href="#">+</a>
<a href="#">ffmpeg-20190225-2e67f75-macos64-static.zip</a>						65.1 MiB		2019-Feb-25 09:45		
<a href="#">ffmpeg-20190225-f948082-macos64-static.zip</a>						65.1 MiB		2019-Feb-26 09:44		
<a href="#">ffmpeg-20190227-85051fe-macos64-static.zip</a>						65.1 MiB		2019-Mar-01 09:44		
<a href="#">ffmpeg-20190301-3b23eb2-macos64-static.zip</a>						65.1 MiB		2019-Mar-02 09:44		
<a href="#">ffmpeg-20190303-4635f64-macos64-static.zip</a>						65.1 MiB		2019-Mar-04 09:45		
<a href="#">ffmpeg-20190303-73b62f3-macos64-static.zip</a>						65.1 MiB		2019-Mar-03 09:44		
<a href="#">ffmpeg-20190304-db33283-macos64-static.zip</a>						65.1 MiB		2019-Mar-05 09:44		
<a href="#">ffmpeg-20190306-056a2ac-macos64-static.zip</a>						65.1 MiB		2019-Mar-06 09:45		
<a href="#">ffmpeg-20190307-0ce759d-macos64-static.zip</a>						65.1 MiB		2019-Mar-07 09:45		
<a href="#">ffmpeg-20190308-147ef1d-macos64-static.zip</a>						65.1 MiB		2019-Mar-09 09:45		
<a href="#">ffmpeg-20190308-9645147-macos64-static.zip</a>						65.1 MiB		2019-Mar-08 09:46		
<a href="#">ffmpeg-20190310-1144d5c-macos64-static.zip</a>						65.1 MiB		2019-Mar-10 08:49		
<a href="#">ffmpeg-20190310-5ab44ff-macos64-static.zip</a>						65.1 MiB		2019-Mar-11 08:46		
<a href="#">ffmpeg-20190312-d227ed5-macos64-static.zip</a>						65.1 MiB		2019-Mar-12 08:47		
<a href="#">ffmpeg-20190315-def18ac-macos64-static.zip</a>						65.1 MiB		2019-Mar-15 16:52		
<a href="#">ffmpeg-20190316-6cfa173-macos64-static.zip</a>						65.1 MiB		2019-Mar-16 08:44		
<a href="#">ffmpeg-20190316-d15117c-macos64-static.zip</a>						65.1 MiB		2019-Mar-17 08:43		
<a href="#">ffmpeg-20190318-15d016b-macos64-static.zip</a>						65.1 MiB		2019-Mar-18 08:43		
<a href="#">ffmpeg-3.2-macos64-static.zip</a>						51.6 MiB		2017-Oct-30 19:41		
<a href="#">ffmpeg-3.3-macos64-static.zip</a>						45.6 MiB		2017-Aug-30 02:24		
<a href="#">ffmpeg-3.3.4-macos64-static.zip</a>						51.6 MiB		2017-Oct-15 16:32		
<a href="#">ffmpeg-3.4-macos64-static.zip</a>						53.8 MiB		2017-Oct-16 16:21		
<a href="#">ffmpeg-3.4.1-macos64-static.zip</a>						53.9 MiB		2017-Dec-13 07:20		
<a href="#">ffmpeg-3.4.2-macos64-static.zip</a>						61.0 MiB		2018-Feb-25 02:16		
<a href="#">ffmpeg-4.0-macos64-static.zip</a>						59.9 MiB		2018-Apr-30 00:46		
<a href="#">ffmpeg-4.0.1-macos64-static.zip</a>						60.5 MiB		2018-Jul-11 16:04		
<a href="#">ffmpeg-4.0.2-macos64-static.zip</a>						60.6 MiB		2018-Jul-25 20:52		
<a href="#">ffmpeg-4.1-macos64-static.zip</a>						62.6 MiB		2018-Nov-12 03:01		
<a href="#">ffmpeg-4.1.1-macos64-static.zip</a>						63.8 MiB		2019-Feb-21 13:40		
<a href="#">ffmpeg-4.2.0-macos64-static.zip</a>						65.1 MiB		2019-Mar-18 08:42		

-》 其他软件，库等，也往往都可以找到历史版本，旧版本。

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新: 2025-06-05 23:01:02

## 附录

下面列出相关参考资料。

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新: 2025-06-04 22:13:11

## 参考资料

- [【已解决】VSCode中调试Python代码](#)
- [【整理】关于版本号version number和build number](#)
- [最智能的Python的IDE: PyCharm](#)
- [版本选择 · 最智能的Python的IDE: PyCharm](#)
- [强大的音视频处理工具: FFmpeg](#)
- 
- [如何运用 DDD 解决团队协作与沟通问题?](#)
- [Bug是如何产生的? - 知乎 \(zhihu.com\)](#)
- [IT: Software doesn't change, requirements do | LinkedIn](#)
- [YesDev-创业团队的研发全流程闭环管理-阿里云开发者社区 \(aliyun.com\)](#)
- [大厂前端日常窥探「壹」：企业级软件开发流程长啥样？-腾讯云开发者社区-腾讯云 \(tencent.com\)](#)
- [信息管理软件需求分析阶段的实践经验及论述（2010年）项目管理朱又生\\_InfoQ写作社区](#)
- [Senior Workplace Strategist \(gsa.gov\)](#)
- [Vue 2.0 浅谈–生命周期和钩子函数 - 小慢车 - SegmentFault 思否](#)
- [Vue2.0 探索之路——生命周期和钩子函数的一些理解 - JS那些事儿 - SegmentFault 思否](#)
- [生命周期钩子 | Vue.js](#)
- [Page Lifecycle API 教程 - 阮一峰的网络日志](#)
- [Page Lifecycle 1](#)
- [\[整理\] 数据库设计主键时UUID和GUID的区别](#)
- [Does Column Width of 80 Make Sense in 2018? – Hacker Noon](#)
- [childes-db: API](#)
- [r-lib/pkgdown: Generate static html documentation for an R package](#)
- [Make Static HTML Documentation for a Package • pkgdown](#)
- [GitHub - onevcat/VVDocumenter-Xcode: Xcode plug-in which helps you write documentation comment easier, for both Objective-C and Swift.](#)
- [Semantic Versioning 2.0.0 | Semantic Versioning](#)
- [Download PyCharm: Python IDE for Professional Developers by JetBrains](#)
- [Mongo Management Studio - the professional MongoDB GUI](#)
- [Choose an open source license | Choose a License](#)
- [Licenses | Choose a License](#)
- [关于开源的一些注意事项 | Mrljdx's Blog](#)
- [开源协议 | 程序员的自我修养](#)
- [CocoaPods使用 - iOS - 掘金](#)
- [Builds - Zeranoe FFmpeg](#)
- 

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新: 2025-06-05 22:56:53