程序员 Java 开源 C++ 修改

如何去阅读并学习一些优秀的开源框架的源码？ 修改

对框架的使用已经比较熟练，实现的原理也有一定的了解，但不知道如何理清项目的整体架构和流程，能有个类似流程图的东西就好了 修改

40 个回答 默认排序

phodal， 待我代码编成，娶你为妻可好 @花仲马
野原圆长、 魏晋 等 965 人赞同

所有让你直接看源码的回答都是在扯淡，你应该从“某个版本”开始阅读代码。

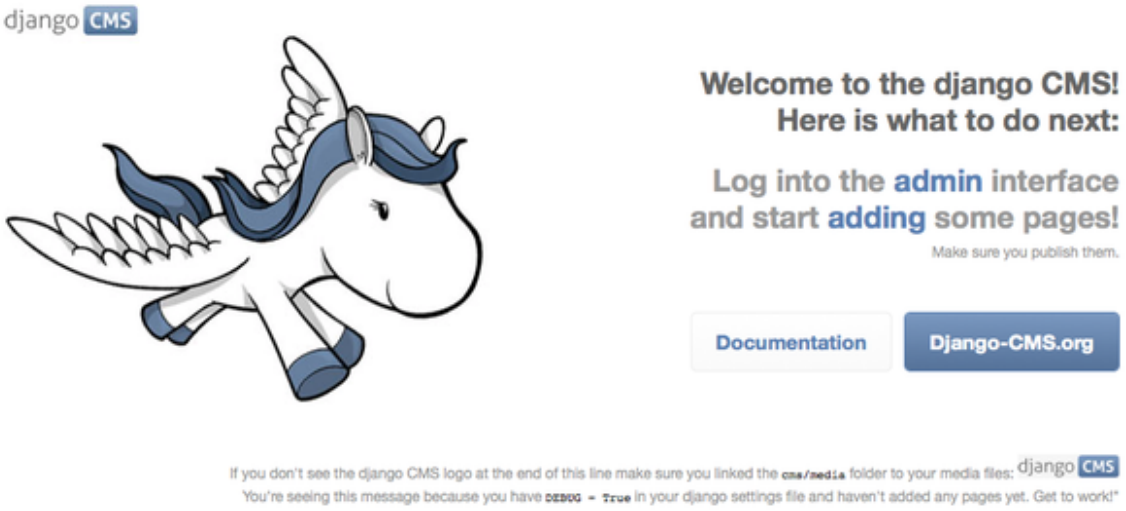
我们并不建议所有的读者都直接看最新的代码，正确的姿势应该是：

- **clone**某个项目的代码到本地
- 查看这个项目的**release**列表
- 找到一个看得懂的**release**版本，如**1.0**或者更早的版本
- 读懂上一个版本的代码
- 向后阅读大版本的源码
- 读最新的源码

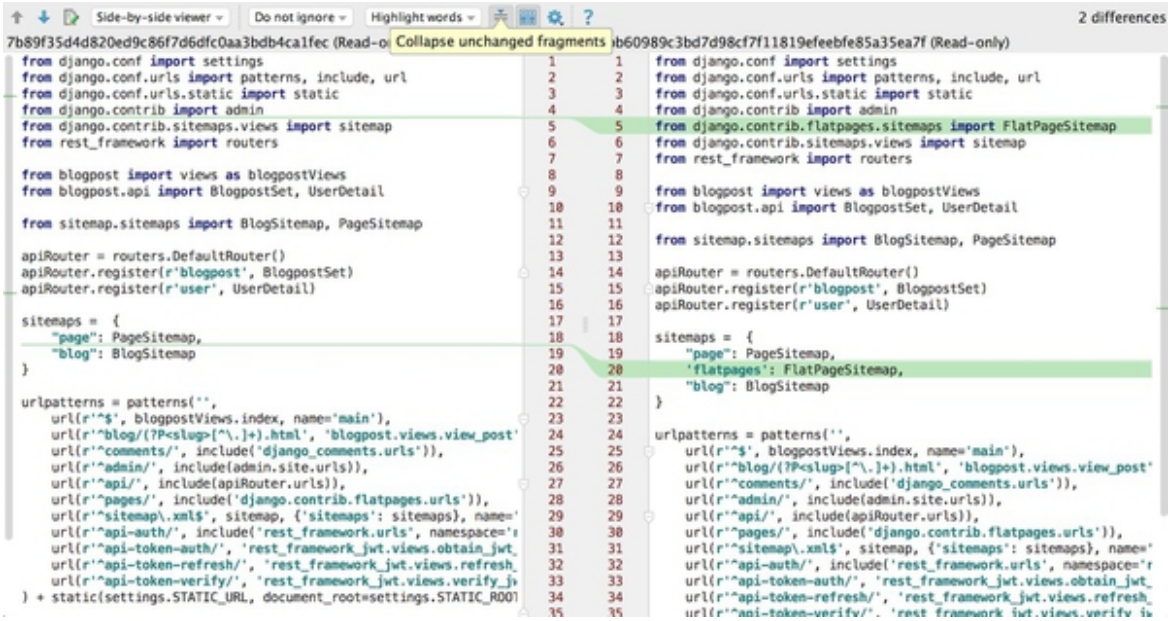
最好的在这个过程中，可以自己造轮子来实现一遍。

阅读过程

在我阅读的前端库、Python后台库的过程中，我们都是以造轮子为目的展开的。所以在最开始的时候，我需要一个可以工作，并且拥有我想要的功能的版本。



紧接着，我就可以开始去实践这个版本中的一些功能，并理解他们是怎么工作的。再用git大法展开之前修改的内容，可以使用IDE自带的Diff工具：



或者类似于SourceTree这样的工具，来查看修改的内容。
在我们理解了基本的核心功能后，我们就可以向后查看大、中版本的更新内容了。
开始之前，我们希望大家对版本号管理有一些基本的认识。

版本号管理

我最早阅读的开始软件是Linux，而下面则是Linux的Release过程：

表 1 - 1 内核的主要版本

版本号	发布/编制日期	说明
0.00	1991.2-4	两个进程，分别在屏幕上显示'AAA...'和'BBB...'。（注：没有发布）
0.01	1991.9.17	第一个正式向外公布的 Linux 内核版本。多线程文件系统、分段和分页内存管理。还不包含软盘驱动程序。
0.02	1991.10.5	该版本以及 0.03 版是内部版本，目前已经无法找到。特点同上。
0.10	1991.10	由 Ted Ts'o 发布的 Linux 内核版本。增加了内存分配库函数。在 boot 目录中含有一个把 as86 汇编语法转换成 gas 汇编语法的脚本程序。
0.11	1991.12.8	基本可以正常运行的内核版本。支持硬盘和软驱设备以及串行通信。
0.12	1992.1.15	主要增加了数学协处理器的软件模拟程序。增加了作业控制、虚拟控制台、文件符号链接和虚拟内存对换（swapping）功能。
0.95.x (即 0.13)	1992.3.8	加入虚拟文件系统支持，但还是只包含一个 MINIX 文件系统。增加了登录功能。改善了软盘驱动程序和文件系统的性能。改变了硬盘命名和编号方式。原

表格源自一本书叫《Linux内核0.11(0.95)完全注释》，简单地再介绍一下：

- 版本0.00是一个hello,world程序
- 版本0.01包含了可以工作的代码
- 版本0.11是基本可以正常的版本

这里就要扯到《GNU 风格的版本号管理策略》：

1．项目初版本时，版本号可以为 0.1 或 0.1.0, 也可以为 1.0 或 1.0.0，如果你为人很低调，我想你会选择那个主版本号为 0 的方式；
2．当项目在进行了局部修改或 bug 修正时，主版本号和子版本号都不变，修正版本号加 1；
3. 当项目在原有的基础上增加了部分功能时，主版本号不变，子版本号加 1，修正版本号复位为 0，因而可以被忽略掉；
4．当项目在进行了重大修改或局部修正累积较多，而导致项目整体发生全局变化时，主版本号加 1；
5．另外，编译版本号一般是编译器在编译过程中自动生成的，我们只定义其格式，并不进行人为控制。

因此，我们可以得到几个简单的结论：

- 我们需要阅读最早的有核心代码的版本
- 我们需要阅读1.0版本的Release
- 往后每一次大的Release我们都需要了解一下

示例

以Flask为例：
一、先Clone它。

二、从Release页面找到它的早期版本：

三、 从上面拿到它的提交号8605cc3，然后checkout到这次提交，查看功能。在这个版本里，一共有六百多行代码

还是有点长

四、我们可以找到它的最早版本:

然后查看它的flask.py文件，只有简单的三百多行，并且还包含一系列注释。

五、接着，再回过头去阅读

- 0.1版本
- 。。。
- 最新的0.10.1版本

首发自我的微信公众号：如何以“正确的姿势”阅读开源软件代码

```
from flask import Flask

app = Flask(__name__)


@app.route('/')
def index():
    return 'Yet another hello!'

if __name__ == '__main__':
    app.run()
```

如上述代码，我想知道一个"/"路径的HTTP请求进来后，是如何进入到index视图处理函数中的，方法很简单，Python中通过sys._getframe方法能得到当前上下文的调用栈，拿到调用栈后，用pygraphviz把它画出来：

```
from flask import Flask

app = Flask(__name__)


@app.route('/')
def index():
    from utils import cheese # 下面附上utils模块的实现
    cheese()
    return 'Yet another hello!'

if __name__ == '__main__':
    app.run()
```

运行上述代码，访问127.0.0.1:5000/ ，会得到一张图：

可以简单介绍下，绿线表示程序启动后的第一次调用，红线表示进入当前上下文的最后一次调用。每一条线表示一次invoke，#符号后面的数字是序号，at XXX表示该次调用发生在这个文件（文件路径在框上方）的第几行。在圆圈里，XXX:YYY，YYY是调用的方法名，XXX表示这个方法是在该文件的第几行被定义的。

实现的思路就两点：1、拿到调用栈（我只知道Python怎么拿）。2、将它画出来，这边使用的pygraphviz是对

graphviz这个画图组件的binding，graphviz在很多语言里都有binding实现，实在不济，先生成输出dot的源码，再绘制出来。

最后贴上那个utils.py的源码，记得安装必要的依赖：

```
from __future__ import unicode_literals


def cheese(frame=None, slient=False):
    import sys
    import tempfile
    import webbrowser
    import pygraphviz as pgv

    if not frame:
        frame = sys._getframe().f_back

    G = pgv.AGraph(strict=False, directed=True)

    stack = []

    node_set = set()
    subgraph_set = {}

    while frame:
        filename = frame.f_code.co_filename
        firstlineno = frame.f_code.co_firstlineno
        function = frame.f_code.co_name

        node = '{0}:{1}:{2}'.format(filename, firstlineno, function)
        if node not in node_set:
            node_set.add(node)
            if filename not in subgraph_set:
                subgraph_set[filename] = G.add_subgraph(
                    name='cluster' + filename,
                    label=filename
                )
            subgraph = subgraph_set[filename]
            subgraph.add_node(
                node,
                label='{0}:{1}'.format(firstlineno, function)
            )

        stack.append(frame)
        frame = frame.f_back

    stack.reverse()

    len_stack = len(stack)

    for index, start in enumerate(stack):

        if index + 1 < len_stack:
            start_filename = start.f_code.co_filename
            start_firstlineno = start.f_code.co_firstlineno
            start_function = start.f_code.co_name
            start_lineno = start.f_lineno
```

```
            start_subgraph = subgraph_set[start_filename]

            end = stack[index + 1]
            end_filename = end.f_code.co_filename
            end_firstlineno = end.f_code.co_firstlineno
            end_function = end.f_code.co_name
            end_subgraph = subgraph_set[end_filename]

            if index == 0:
                color = 'green'
            elif index == len_stack - 2:
                color = 'red'
            else:
                color = 'black'

            G.add_edge(
                '{0}:{1}:{2}'.format(start_filename,
                                    start_firstlineno,
                                    start_function),
                '{0}:{1}:{2}'.format(end_filename,
                                    end_firstlineno,
                                    end_function),
                color=color,
                ltail=start_subgraph.name,
                lhead=end_subgraph.name,
                label='#{0} at {1}'.format(index + 1, start_lineno)
            )

    fd, name = tempfile.mkstemp('.png')

    G.draw(name, prog='dot')
    G.close()

    if not slient:
        webbrowser.open('file://' + name)

    return name
```

编辑于 2014-12-01　27 条评论　感谢　分享　收藏 • 没有帮助 • 举报 • 作者保留权利

357

ze ran，编程话题优秀回答者 • less is more

357 人赞同

读代码通常要能回答两个问题，

1，要解决什么问题？
2，如何实现的？

大到整个项目，小到一个模块，函数，看的时候，都要抱着这两个问题去看。看完了，这两个问题能答上来，才是有效。

代码不仅是读，还要理和试。具体可以这样做，

背景调查。看官网介绍，维基百科，了解主要功能，被应用于哪些项目，同类框架还有哪些。

使用框架。至少 follow "Get Started" 做个小 demo。别奇怪，真有人连 API 都没调用过，上来就看代码。以为省了时间，实际是迈向自我摧残。

看早期版本。早期版本往往用最直接的方法实现最核心的功能，理解起来效率最高。后期的异常处理，重构等，反而增加阅读难度，错失重点。读了半天，可能看的是个 corner case.

在IDE里读。IDE里可以方便跳转，查看定义，比起网页上看效率高得多。

尽可能编译调试。能调试的代码，几乎没有看不懂的。

了解一些设计模式。这样看到名字里有 proxy, builder, factory 之类的，就心领神会了。

横向分层，纵向分块。代码都是分模块的，有的是 core, 有的是 util，parser 之类的，要知道看的是哪一层，那一块。有的小项目分层不明显，也不必强求。

要看的不只是语法上的技巧，更重要的是设计上的思路和原理。读没读懂，最简单的标准是，假如给充足的时间，有没有信心写出一个差不多的东西来。

叶羽客

既然 @Karma 写了个 python 的 Call Graph，
那我就参考他的方法写了个 C/C++ 的 Call Graph 生成器：
CodeSnippet/python/SRCGraphviz/c++ at master · Cheukyin/CodeSnippet · GitHub

主要思路是
利用 gcc/g++ 的 -finstrument-functions 的注入选项，
得到每个函数的调用地址信息，生成一个trace文件，
然后利用 addr2line 和 c++filt 将 函数名及其所在源码位置 从地址中解析出来，
从而得到程序的 Call Stack，
然后用 pygraphviz 画出来

先举个例子看看用法：
比如我现在有A.hpp、B.hpp、C.hpp、ABCTest.cpp这几个文件，
我想看他们的Call Graph
源码如下：

然后按下面编译(instrument.c 在上面github地址中可以下载，用于注入地址信息)：

```
g++ -g -finstrument-functions -O0 instrument.c ABCTest.cpp -o test
```

然后运行程序，得到 trace.txt

```
./test
```

最后

```
python CallGraph.py trace.txt test
```

弹出一张Call Graph

图上标注含义在Karma的答案有说明，我就不再重复了

下面说说获取**C/C++**调用关系的原理：

利用 -finstrument-functions 编译选项，
可以让编译器在每个函数的开头和结尾注入 __cyg_profile_func_enter 和 __cyg_profile_func_exit
这两个函数的实现由用户定义

在本例中，只用到 __cyg_profile_func_enter，定义在 instrument.c 中，
其函数原型如下:

```
void __cyg_profile_func_enter (void *this_fn, void *call_site);
```

其中 this_fn 为 被调用的地址，call_site 为 调用方的地址

显然，假如我们把所有的 调用方和被调用方的地址 都打印出来，
就可以得到一张完整的运行时**Call Graph**

因此，我们 instrument.c 实现如下：

其中 main_constructor 在 调用main 前执行，main_deconstructor 在调用main 后执行，
以上几个函数的作用就是 将所有的 调用方和被调用方的地址 写入 **trace.txt** 中

然而，现在有一个问题，就是 trace.txt 中保存的是地址，我们如何将地址翻译成源码中的符号？
答案就是用 **addr2line**，
以上面ABCTest.cpp工程为例，比如我们现在有地址 0x400974，输入一下命令

```
addr2line 0x400aa4 -e a.out -f
```

结果为

```
_ZN1A4AOneEv
/home/cheukyin/PersonalProjects/CodeSnippet/python/SRCGraphviz/c++/A.hpp:11
```

第一行该地址所在的函数名，第二行为函数所在的源码位置

然而，你一定会问，_ZN1A4AOneEv是什么鬼？

为实现重载、命名空间等功能，因此C++有 **name mangling**，因此函数名是不可读的，
我们需要利用 **c++filt** 作进一步解析：

```
addr2line 0x400aa4 -e a.out -f | c++filt
```

结果是不是就清晰很多：

```
A::AOne()
/home/cheukyin/PersonalProjects/CodeSnippet/python/SRCGraphviz/c++/A.hpp:11
```

最后用 pygraphviz 将 每一条调用关系 画出来即可

编辑于 2016-05-14　21 条评论　感谢　分享　收藏 • 没有帮助 • 举报 • 作者保留权利

36

小爝， 前端开发 / Node.js / …

36 人赞同

从第一个commit读起来，而不是最后一个。

发布于 2016-05-15　3 条评论　感谢　分享　收藏 • 没有帮助 • 举报 • 作者保留权利

42

萧井陌， 微信公众号：炼瓜研究所 技术社区 …

42 人赞同

强行读下去

好吧，其实是有工具可以分析代码生成流程图和类调用关系图什么的，我已经说得这么明白了你就搭梯子问谷歌吧。。

编辑于 2014-11-24　2 条评论　感谢　分享　收藏 • 没有帮助 • 举报 • 作者保留权利

26

Milo Yip ，编程、计算机图形学、C++等 6 个话题优秀回答者 • 游…

26 人赞同

Code Reading ：The Open Source Perspective

发布于 2016-10-04　添加评论　感谢　分享　收藏 • 没有帮助 • 举报 • 申请转载

21

hi大头鬼hi， 阿里内推，长期招人

21 人赞同

最早是看linux源码分析那本书，从0.1版本开始分析源码，然后我就学会了看开源项目从第一个release的源码开始分析，代码会简化很多，比较容易弄懂！

比如vue我就是这么看的

发布于 2016-10-05　2 条评论　感谢　分享　收藏 • 没有帮助 • 举报 • 作者保留权利

16

钟宇腾， Bug Creator

16 人赞同

告诉你一个好办法，拿张白纸，跟着函数调用，画图记下关键调用和算法。

发布于 2014-11-24　2 条评论　感谢　分享　收藏 • 没有帮助 • 举报 • 作者保留权利

51

Xiaoyu Ma， 程序员

王名扬 等 51 人赞同

对几万行以下的项目其实怎么读都可以，上十万几十万的东西大概要这样：
. 你需要了解这个系统的核心原理，比如你要看Zookeeper代码，你最好知道Paxos算法；看Hive代码，至少心里清楚Hive的各个组件是干嘛的，一个SQL大致上会怎么被翻译成Job。这些不知道，你看源码需要记忆和理解的东西就太多了。
. 自己部署一份，开启调试，按照你想要了解的场景去用，调试器attach上去看调用栈，尝试猜测这个东西会hit到哪些代码然后加断点。
. 也有时候试着去找相关的unit test和小型的回归测试代码，会更容易理解，因为这些测试设计上会屏蔽一些无关的逻辑，而完整的系统会包含很多繁杂的细节
. 找到社区，找到相关的issue tracking系统，比如jira，可以找找你感兴趣的ticket，看相关的实现。比如你想实现一个不同安全机制，你可以找实现一个安全层的jira ticket，看patch，看别人如何实现一个安全层之类的。更重要的是，也许会有设计相关的讨论，比如为什么用了某个库，为什么用了某个算法，取舍和考量是什么，光看代码你是看不到这些的。

编辑于 2016-05-16　2 条评论　感谢　分享　收藏 • 没有帮助 • 举报 • 作者保留权利

刘侃

1 人赞同

1. 从一个功能的入口看起，熟悉主线逻辑，忽略细节分支。然后重复。。。

2. 从 main 开始看起，了解完整的结构。

3. 思考其他功能的实现方式，和现有代码比对。

从哪个分支看真的不重要，直接拉最新的看。。。等现在的代码都读懂了再想想为什么这么做再考古也不迟。

至于方式，有符号 跳转最好，真想看的话 grep 也够了，几十万行级别代码 grep 也不是很慢。有设计文档能看最好，看ut就耍流氓了。

编辑于 2016-10-05　添加评论　感谢　分享　收藏 • 没有帮助 • 举报 • 作者保留权利

Alvin Qi， INTJ，喜欢Big Clean Problem，EGOIST脑…

9 人赞同

架构也好，流程图也好，都是需要自己梳理的啊。
我想你需要的大概是这个：高效阅读源代码指南

发布于 2016-05-15　添加评论　感谢　分享　收藏 • 没有帮助 • 举报 • 作者保留权利

---

you conquer， 很多人认为他们在思考，其实他们只是重新…

23 人赞同

1. 质量过关的开源项目，源码存放的路径、源码文件命名、源码中变量的命名都会符合特定的规约，根据规约去梳理代码结构

比如 lib 一般用来存放所用到的各种库文件，比如对某种协议栈的实现等等

比如 utils 一般用来存放一些 utility 之类的东西，比如什么把某个 id 映射到 MAC 地址啊，自己实现的某种 Hash 算法的小函数啊之类的

2. 借助 IDE 的代码结构分析工具，比如 Visual Studio 中就可以自动生成代码结构图及调用关系图，具体参考这

个：IDE 而言，是 Xcode 的技术比较先进还是 Visual Studio？ - 小郑的回答

3. 如果是十分知名的开源项目，网上一般可以搜到“XXX源码结构分析”之类的书或者文章，跟着那个看就可以了，即使文章并不是很完整，也可以起到比较好的梳理作用

4. 对我自己而言，我一般是先对源代码进行搜索，找到函数入口，先把入口函数读懂，然后一层一层向下递归展开

比如入口函数是个大的 if-else 结构，那么每一个分支就对应一套处理逻辑，分支里一般都是封装好的函数调用，然后你去跟那些函数调用就好了，函数调用里再调用的结构体啊数组啊都可以一步一步跟下去

比如入口函数只是一个 Event Loop，对于事件队列进行分发，交给后续的 Event Handler 处理，然后就可以从事件队列的初始化、事件进入队列、Handler 如何注册等等跟下去

我读 C 的代码相对多一些，基本在 Windows 底下就是用 Source Insight，在 Linux 底下就是用 Vim + Ctags + taglist，似乎 Source Insight 更加顺手一些

另外读代码的时候也要多读才能读出感觉来，比如你看某个函数抛了一个 up_call_info，那凭直觉就应该猜到这玩意应该是个自定义结构体，用来当作 message 传递给上层的，没准与之对应的还有一个 down_call_info

大概这样

编辑于 2014-11-25　1 条评论　感谢　分享　收藏 • 没有帮助 • 举报 • 作者保留权利

9

王帆， 游戏开发/汽车修理

9 人赞同

说说C与c++的经验吧，一个开源库有三个难点
A，初始化流程。
B，函数之间的调用关系。
C，线程模型。
1，搞清楚开源项目的功能
2，编译运行， 能够设置断点。
3，然后通过一些系统函数了解函数之间的调用关系。
4，用main开始一步步了解初始化流程。
5，了解线程模型，线程之间的关系。
6，步骤3到5一直循环

发布于 2016-05-16　添加评论　感谢　分享　收藏 • 没有帮助 • 举报 • 作者保留权利

15

王豪杰， | hawky.cn/blog

15 人赞同

仔细阅读题主的问题,题主关心的是 java,C++相关的开源框架,而且对框架的 API 和整体理论知识已经很清楚了,猜测框架的代码量应该也比较大.

一楼答案虽然很 cool, 题主未必能用的上.

我试着分享一下自己的方法,要点如下:

- 一开始不要贪心,不要试图画个类图,把整个代码全部概括起来的类图看起来比直接看代码的效果好不了多少.选择一个自己经常用到,而且比较感兴趣内部如何实现的特性作为切入点.

- 写一个测试用例,或者直接从项目中找测试代码 单步调试. java 的话,建议使用 maven 管理依赖的框架代码,非常方便调试.在调试的同时,从作者的角度去考虑如何解决遇到的问题,并尝试从代码中验证你的想法,慢慢在头脑中建立起这个特性的整体架构.所谓"窥一斑而知全豹",举一反三,自然能推出其他特性的流程是怎么样的.
- 无目的地查看框架中其他感兴趣的代码点,做出一些推断,在那里打断点,写测试用例验证你的推断,日积月累,整个代码的框架就清楚了.

最重要的是,在这个过程中,能够学到框架作者解决问题的策略,在以后的工作中可以模仿学习,加深理解.

==========
除此之外,在这里我想简单分享一下,在学习新语言的一些方法.
学习新语言的步骤:

1. 在网上搜教程,一般是视频教程或学习笔记,以最快的速度掌握语言的基础特性.这个阶段,对语言整体有个大致了解.
2. 自己写一些小工具.实现一些工作中经常用到,或有意思的工具,比如说爬虫.这个阶段的主要目标是,把 循环 判断 常用数据结构 基础语法 给用熟练了.
3. 开始在网上学习高级教程,或买一本入门的好书,比如 head first 系列.加深基础特性理解的同时,了解一些高级特性.高级语言一般包括这几个方面的知识:字符串处理,网络编程,进程/线程,面向对象, I/O,图像处理.
4. 在 github 上找一些代码量少,功能有趣,能跑通,重要的是作者比较牛叉的项目,不要去找框架因为跑不起来,也比较乏味. clone 下来,抄之.不要漫无目的的抄,要带着问题,遇到不理解的地方多做实验,尽量做到全部代码360度无死角.也可以根据自己的需要对代码做一些增删改,然后把自己的代码跑起来,是不是很有成就感呢?(节操在哪啊...).这个过程中就可以实践到语言的高级特性,还有一些惯例用法,好的代码规范等等.

编辑于 2014-11-26　添加评论　感谢　分享　收藏 • 没有帮助 • 举报 • 作者保留权利

3

Wang Zhong

3 人赞同

读早期版本，掌握大概架构，不要扣细节。

看ticket list. 研读感兴趣的feature或者优化

编辑于 2016-05-15　1 条评论　感谢　分享　收藏 • 没有帮助 • 举报 • 作者保留权利

5

韩葆-逸松， Coverity 中国-七年软件质量与安全经验，...

5 人赞同

推荐sourceinsight

发布于 2016-05-15　添加评论　感谢　分享　收藏 • 没有帮助 • 举报 • 作者保留权利

2

杨其斌， IT攻城师/meme&gene爱好者

2 人赞同

最笨的但也最有效果的法子，抄一个。

发布于 2014-11-24　添加评论　感谢　分享　收藏 • 没有帮助 • 举报 • 作者保留权利

0

Cosmia Fu， 会写Python的Haskell原教旨主义者

能直接看懂就直接看，相对万能的是 single-step ，不过比较费时间

。。。刚发现这是坟。。。

编辑于 2016-05-16　添加评论　感谢　分享　收藏 • 没有帮助 • 举报 • 作者保留权利

0

法兰克斯雪诺， 疯狂码农，独立游戏制作，退役半职业魔兽…

1通读文档
2通读一遍官方demo
3开始看源码
4不懂的地方文档里找答案
5找不到百度
6再找不到谷歌
（4,5,6顺序不固定）
7实在找不到跳过去
8继续2

编辑于 2016-10-03　添加评论　感谢　分享　收藏 • 没有帮助 • 举报 • 作者保留权利

更多

ldy　填写话题经验，提升回答可信度

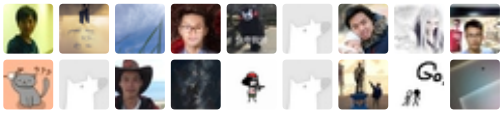
写回答…

匿名 | 允许规范转载 ▾

发布回答

关注问题

5016 人关注该问题



相关问题

中国开源现状如何？ 65 个回答

关于阅读开源项目的源码，有哪些经验值得分享？ 33 个回答

如何更有效地学习开源项目的代码？ 24 个回答

哪些项目的源代码最值得阅读？ 48 个回答

GitCafe 这样的代码托管网站在国内的前景如何？ 56 个回答

问题状态

最近活动于 2016-10-07 • 查看问题日志

被浏览 **44912** 次，相关话题关注者 **520613** 人