
目录

前言	1.1
为何要有良好习惯和风格	1.2
如何才能有良好习惯和风格	1.3
见名知意	1.3.1
不好和好的风格的举例	1.3.2
不同语言的编码规范	1.3.3
代码格式化和工具	1.3.4
附录	1.4
代码中常见英文单词的缩写	1.4.1
参考资料	1.4.2

编程习惯和代码风格

- 最新版本: `v1.6`
- 更新时间: `20190530`

简介

之前不同语言编写过代码，也接触过很多别人写的代码，关于编程习惯和代码风格有些想说的话，要总结的东西，现整理出来供参考。

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：

Gitbook源码

- [crifan/program_code_style](#): 编程习惯和代码风格

如何使用此Gitbook源码去生成发布为电子书

详见: [crifan/gitbook_template: demo how to use crifan gitbook template and demo](#)

在线浏览

- [编程习惯和代码风格 book.crifan.com](#)
- [编程习惯和代码风格 crifan.github.io](#)

离线下载阅读

- [编程习惯和代码风格 PDF](#)
- [编程习惯和代码风格 ePub](#)
- [编程习惯和代码风格 Mobi](#)

版权说明

此电子书教程的全部内容，如无特别说明，均为本人原创和整理。其中部分内容参考自网络，均已备注了出处。如有发现侵犯您版权，请通过邮箱联系我 `admin` 艾特 `crifan.com`，我会尽快删除。谢谢合作。

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。

更多其他电子书

本人 `crifan` 还写了其他 100+ 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](#)

crifan.com, 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：2021-01-17 00:59:08

为何要有良好习惯和风格

在编写代码期间，常常会遇到类似这样的情况：

- 别人：之前写的代码，自己接手该项目，发现之前代码写的很烂，开始骂之前的人太挫，水平太垃圾
- 自己：之前一段时间，比如2个月、1年，写的代码，自己后来需要接着开发或维护，结果发现之前自己写的代码的逻辑都有点记不清楚了

为了避免这种情况，所以我们才要写出好的代码，有良好的代码习惯和风格。

代码写出来就是为了别人看的

很久之前，我以为代码写出来，能工作就可以了。

直到一次听到一个老师教导我们说代码写出来是为了给别人看的，随着工作经验的增长，更加深刻的意识到，其实：

代码写的好（好的风格和习惯，质量高），远远比只是让机器（电脑）看懂（能执行）更加重要

crifan.com，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：2018-01-11 21:51:28

如何才能有良好习惯和风格

接下来介绍如何才能做到良好的编程习惯和代码风格。

crifan.com, 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: 2018-01-11 20:50:35

见名知意

写代码时，除了要有良好的代码逻辑之外，其中很重要的一点是：在给变量、函数等命名时保证见名知意，这样才能：

- 看到（变量，函数等）的（代码）名字，就能理解其含义
- 看到代码，就能知道其作用

见名知意==自描述性==Expressive

之前看到过swift语言的介绍，其中就有Expressive的解释：[Swift.org - About Swift](#)

Expressive. Swift benefits from decades of advancement in computer science to offer syntax that is a joy to use, with modern features developers expect. But Swift is never done. We will monitor language advancements and embrace what works, continually evolving to make Swift even better.

即：语言有自描述性，就很好用。

所以个人也感觉 `swift` 也是属于好用的编程语言之一

注：另外一个觉得好用的是 `Python`

crifan.com, 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: 2018-01-11 21:46:44

不好和好的风格的举例

那么如何才算有好的代码风格和习惯呢？

接下来就来通过举例来具体说明。

前面提到了，好的风格和习惯主要指的是见名知意

那么如何才能实现见名知意？

即：如何给变量，函数，参数，类等等命名？

举例：不好的做法 -> 好的做法

变量名应该体现出所属类的中所要代表的含义

卓越一线中有几个大功能模板：

- 报表
- 任务
 - 售前店访
 - 店头点检
 - 小区例会
- 通知

在售前店访模块中，实现一个tableView中的一个section的item时，使用了 `CurSectionItem`

-> 从这个名字看出是，其所想要表达的含义是：当前的section的item

-> 以及和使用范围是：（以为是）当前的售前店访的专用的Section的item呢

-> 而实际上这个CurSectionItem是整个任务模块通用的

-> 所以，这个不是好的做法

-> 应该改为：

符合此处代码的原意：整个任务模块的所有页面的通用的TableView中用到的Section的item

-> 所以，此处可以改名为，类似于： `TaskTableViewSectionItem`

-> 当然，考虑到名字有点长，想要缩短，且又不会失去本身的含义，不会和其他代码有冲突

-> 可以改为： `TaskTVSectionItem` =task的tableview的sectionItem

-> TaskSectionItem=task的（所有页面通用的，只有tableview中才会有的section这个概念）的item

加上必要的换行

```
func toDictionary() -> NSDictionary
```

```
{
    let dictionary = NSMutableDictionary()
    if id != nil{
        dictionary["id"] = id
    }
    if publishDate != nil{
        dictionary["publishDate"] = publishDate
    }
    if publisherName != nil{
        dictionary["publisherName"] = publisherName
    }
    if readover != nil{
        dictionary["readover"] = readover
    }
    if subtitle != nil{
        dictionary["subtitle"] = subtitle
    }
    if title != nil{
        dictionary["title"] = title
    }
    return dictionary
}
```

```
39 func toDictionary() -> NSDictionary
40 {
41     let dictionary = NSMutableDictionary()
42     if id != nil{
43         dictionary["id"] = id
44     }
45     if publishDate != nil{
46         dictionary["publishDate"] = publishDate
47     }
48     if publisherName != nil{
49         dictionary["publisherName"] = publisherName
50     }
51     if readover != nil{
52         dictionary["readover"] = readover
53     }
54     if subtitle != nil{
55         dictionary["subtitle"] = subtitle
56     }
57     if title != nil{
58         dictionary["title"] = title
59     }
60     return dictionary
61 }
```

问题:

每个if后面，没有适当的换行

-> 容易让看代码的人误以为是 if else 的写法，是一体的呢

-> 实际上此处是独立的多个if判断。

```
func toDictionary() -> NSDictionary
```



```
{
    let dictionary = NSMutableDictionary()

    if id != nil{
        dictionary["id"] = id
    }

    if publishDate != nil{
        dictionary["publishDate"] = publishDate
    }

    if publisherName != nil{
        dictionary["publisherName"] = publisherName
    }

    if readover != nil{
        dictionary["readover"] = readover
    }

    if subtitle != nil{
        dictionary["subtitle"] = subtitle
    }

    if title != nil{
        dictionary["title"] = title
    }

    return dictionary
}
```

```
39     func toDictionary() -> NSDictionary
40     {
41         let dictionary = NSMutableDictionary()
42
43         if id != nil{
44             dictionary["id"] = id
45         }
46
47         if publishDate != nil{
48             dictionary["publishDate"] = publishDate
49         }
50
51         if publisherName != nil{
52             dictionary["publisherName"] = publisherName
53         }
54
55         if readover != nil{
56             dictionary["readover"] = readover
57         }
58
59         if subtitle != nil{
60             dictionary["subtitle"] = subtitle
61         }
62
63         if title != nil{
64             dictionary["title"] = title
65         }
66
67         return dictionary
68     }
```

去除多余的注释

代码本身如果可以说明其自身含义，就无需再加上，画蛇添足的，多余的注释了。

TODO： 加上例子说明

对此多说几句：

之前我们被教导说，写代码，一定要加上注释。并且好像注释写的越多，代码质量就越高似的。

而实际上：

后来遇到一个来自 Oracle 的前同事，得知Oracle公司内，要求写代码时，一定要去掉多余的注释，不允许有任何多余的注释。

如果非要在代码中加注释，则需要提交申请才可以的。

其背后逻辑很简单：

如果你的代码本身的逻辑已经很清楚，且各种变量函数命名很清楚，总体代码质量很高，则就不需要加上额外的多余的注释。

如果非要加上注释，也是不得已的情况，比如：

- 业务逻辑特殊，需要加注释去解释清楚
- 由于各种算法，使用场景的特殊性，代码本身没法反应出来，需要额外加注释说明

对此做法我也比较赞同，总结来说就是：

除非不得已，否则不需要的多余注释。而应该把更多精力花在构思良好的架构和见名知意的变量函数的命名上，如此可以大幅提高代码质量

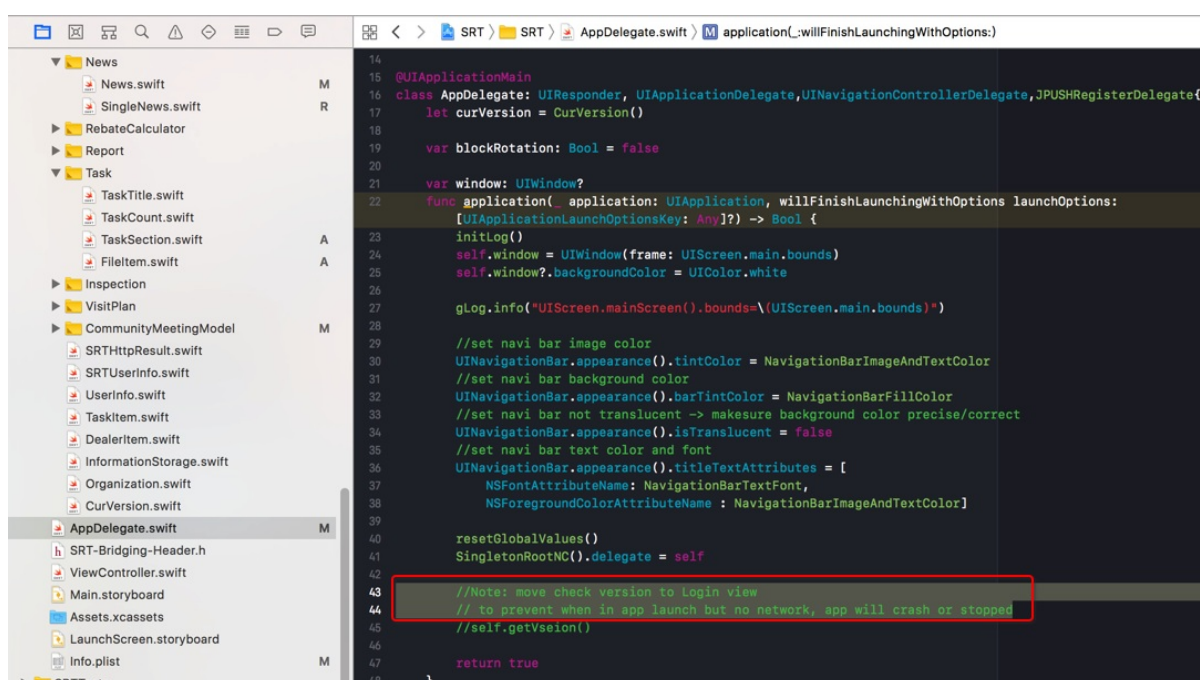
加上必要的注释

什么叫必要的注释？

如上面所述：代码本身没法体现出的逻辑、背景、特殊情况等内容，都要加上必要的注释和说明

举例如下：

如图：



背景是：

卓越一线中iOS中的AppDelegate.swift的AppDelegate中，

则部分是app启动时会执行的代码

此处把之前用于检测新版本的代码注释掉了：

```
//self.getVseion()
```

但是却没有加上合适的说明：

通过询问才了解到：

此处注释掉的原因是：

如果app启动时，用户当前时没有网络的，则此处获取版本就会死掉，导致程序卡死

所以才注释掉

-》所以，此处本身代码只有一行：

```
//self.getVseion()
```

是看不出来这个背后的逻辑的，所以才要加上必要的注释：

```
//Note: move check version to Login view  
// to prevent when in app launch but no network, app will crash or stopped  
//self.getVseion()
```

-》并且，此处也不应该直接删除掉

```
//self.getVseion()
```

这行代码

-》否则后续的维护人员（包括自己时间长回来看自己代码）就会不知道这个本该需要注意的逻辑。

去除和合并冗余的代码

写代码有个逻辑

```
less is more
```

更少的代码（如果能够实现同样的功能，且逻辑清楚的情况下）可以带来更多的（好处，比如代码简洁易懂，质量更高，更易于维护，更不容易出错等等）

如果某些代码，在多个地方（>= 2次）被用到，则一般来说，都应该提取出来写成公共的代码（函数，库，类）

比如：

对于已知的定义：

```
enum CheckListVType{  
    case writeCheckList //填写检查表  
    case onlyRead //查看检查表  
    case onlyReadWithPass //查看检查表  
}  
  
enum ShopVisitType{  
    case toBeFilled //待填写  
    case reject //驳回  
    case pass //通过  
    case approval //待审批  
}
```

多个判断条件可以被整合优化

来说，如下的代码：

```
func rightDrawComeBack(id:String, type:String){
```

```

gLog.debug("检查表\({id}")
var visitId = self.id
if type == "last"{
    visitId = getPlanId(id: id)
}

if self.curShopVisitType == .approval{
    dispatchMain_async({
        let CheckListVC = CheckListViewController(curType: .onlyReadWithPass, id: id,
visitId: visitId)
        self.show(CheckListVC, sender: self)
    })
}else if(self.curShopVisitType == .toBeFilled){
    dispatchMain_async({
        let CheckListVC = CheckListViewController(curType: .writeCheckList, id: id, vi
sitId: visitId)
        self.show(CheckListVC, sender: self)
    })
}else if (self.curShopVisitType == .pass){

    dispatchMain_async({
        let CheckListVC = CheckListViewController(curType: .onlyReadWithPass, id: id,
visitId: visitId)
        self.show(CheckListVC, sender: self)
    })
}else if(self.curShopVisitType == .reject){
    dispatchMain_async({
        let CheckListVC = CheckListViewController(curType: .writeCheckList, id: id, vi
sitId: visitId)
        self.show(CheckListVC, sender: self)
    })
}
}
}

```

对于 `curShopVisitType` 的所有的4种可能都判断了，且每部分的代码都是一样的，就显得很冗余和啰嗦

相关部分应该改为：

```

var curCheckListVCType = CheckListVCType.onlyReadWithPass
if (self.curShopVisitType == .approval) || (self.curShopVisitType == .pass) {
    curCheckListVCType = .onlyReadWithPass
} else if(self.curShopVisitType == .toBeFilled) || (self.curShopVisitType == .reject) {

    curCheckListVCType = .writeCheckList
}

let checkListVC = CheckListViewController(curType: curCheckListVCType, id: id, visitId:
visitId)
self.show(checkListVC, sender: self)

```

或者、甚至：

```
//for .approval or .pass, set default type to onlyReadWithPass
var curCheckListVCType = CheckListVCType.onlyReadWithPass
if(self.curShopVisitType == .toBeFilled) || (self.curShopVisitType == .reject) {
    curCheckListVCType = .writeCheckList
}

let checkListVC = CheckListViewController(curType: curCheckListVCType, id: id, visitId:
visitId)
self.show(checkListVC, sender: self)
```

```
334  /******
335  * 左滑回调
336  *****/
337  func rightDrawComeBack(id:String, type:String){
338      gLog.debug("检查表\(id)")
339      var visitId = self.id
340      if type == "last" {
341          visitId = getPlanId(id: id)
342      }
343
344      //for .approval or .pass, set default type to onlyReadWithPass
345      var curCheckListVCType = CheckListVCType.onlyReadWithPass
346      if(self.curShopVisitType == .toBeFilled) || (self.curShopVisitType == .reject) {
347          curCheckListVCType = .writeCheckList
348      }
349
350      let checkListVC = CheckListViewController(curType: curCheckListVCType, id: id, visitId: visitId)
351      self.show(checkListVC, sender: self)
352  }
```

都可以。

-> 如此改动，逻辑就清楚多了，代码也更不容易出错。

多分支中代码重复应该被合并

这些代码：

```
switch self.curShopVisitType {
case .approval:
    dealerTimeSection.cellList.append(dealerCell)
    dealerTimeSection.cellList.append(timeCell)
    self.sectionItemList.append(dealerTimeSection)

    descriptionSection.sectionHeader = ""
    descriptionSection.cellList.append(descriptionCell)
    self.sectionItemList.append(descriptionSection)

    lastVisitSection.sectionHeader = "上次走访项"
    self.sectionItemList.append(lastVisitSection)
    self.lastVisitSectionIndex = self.sectionItemList.count - 1

    thisVisitSection.sectionHeader = "本次走访项"
    self.sectionItemList.append(thisVisitSection)
```

```
self.thisVisitSectionIndex = self.sectionItemList.count - 1

if (gCurUserItem.userInfo.jobType == .director || gCurUserItem.userInfo.jobType ==
.VP){
    buttonSection.cellList.append(buttonTableViewCell)
    self.sectionItemList.append(buttonSection)
    self.buttonSectionIndex = self.sectionItemList.count - 1
}

case .toBeFilled:
dealerTimeSection.cellList.append(dealerCell)
dealerTimeSection.cellList.append(timeCell)
self.sectionItemList.append(dealerTimeSection)

descriptionSection.sectionHeader = ""
descriptionSection.cellList.append(descriptionCell)
self.sectionItemList.append(descriptionSection)

lastVisitSection.sectionHeader = "上次走访项"
self.sectionItemList.append(lastVisitSection)
self.lastVisitSectionIndex = self.sectionItemList.count - 1

thisVisitSection.sectionHeader = "本次走访项"
thisVisitSection.cellList.append(addProblemCell)
self.sectionItemList.append(thisVisitSection)
self.thisVisitSectionIndex = self.sectionItemList.count - 1

buttonSection.cellList.append(buttonTableViewCell)
self.sectionItemList.append(buttonSection)
self.buttonSectionIndex = self.sectionItemList.count - 1

case .pass:
typeSection.cellList.append(typeCell)
self.sectionItemList.append(typeSection)

dealerTimeSection.sectionHeader = ""
dealerTimeSection.cellList.append(dealerCell)
dealerTimeSection.cellList.append(timeCell)
self.sectionItemList.append(dealerTimeSection)

descriptionSection.sectionHeader = ""
descriptionSection.cellList.append(descriptionCell)
self.sectionItemList.append(descriptionSection)

lastVisitSection.sectionHeader = "上次走访项"
self.sectionItemList.append(lastVisitSection)
self.lastVisitSectionIndex = self.sectionItemList.count - 1

thisVisitSection.sectionHeader = "本次走访项"
self.sectionItemList.append(thisVisitSection)
self.thisVisitSectionIndex = self.sectionItemList.count - 1

case .reject:
```

```

typeSection.cellList.append(typeCell)
self.sectionItemList.append(typeSection)

dealerTimeSection.sectionHeader = ""
dealerTimeSection.cellList.append(dealerCell)
dealerTimeSection.cellList.append(timeCell)
self.sectionItemList.append(dealerTimeSection)

descriptionSection.sectionHeader = ""
descriptionSection.cellList.append(descriptionCell)
self.sectionItemList.append(descriptionSection)

lastVisitSection.sectionHeader = "上次走访项"
self.sectionItemList.append(lastVisitSection)
self.lastVisitSectionIndex = self.sectionItemList.count - 1

thisVisitSection.sectionHeader = "本次走访项"
thisVisitSection.cellList.append(addProblemCell)
self.sectionItemList.append(thisVisitSection)
self.thisVisitSectionIndex = self.sectionItemList.count - 1

buttonSection.cellList.append(buttonTableViewCell)
self.sectionItemList.append(buttonSection)
self.buttonSectionIndex = self.sectionItemList.count - 1
}

```

改为:

```

if (self.curShopVisitType == .pass) || (self.curShopVisitType == .reject) {
    typeSection.cellList.append(typeCell)
    self.sectionItemList.append(typeSection)
}

if (self.curShopVisitType == .pass) || (self.curShopVisitType == .reject) {
    dealerTimeSection.sectionHeader = ""
}
dealerTimeSection.cellList.append(dealerCell)
dealerTimeSection.cellList.append(timeCell)
self.sectionItemList.append(dealerTimeSection)

descriptionSection.sectionHeader = ""
descriptionSection.cellList.append(descriptionCell)
self.sectionItemList.append(descriptionSection)

lastVisitSection.sectionHeader = "上次走访项"
self.sectionItemList.append(lastVisitSection)
self.lastVisitSectionIndex = self.sectionItemList.count - 1

thisVisitSection.sectionHeader = "本次走访项"
if (self.curShopVisitType == .toBeFilled) || (self.curShopVisitType == .reject) {
    thisVisitSection.cellList.append(addProblemCell)
}

```



```
self.sectionItemList.append(thisVisitSection)
self.thisVisitSectionIndex = self.sectionItemList.count - 1

if (
    (
        (self.curShopVisitType == .approval) &&
        (
            gCurUserItem.userInfo.jobType == .director ||
            gCurUserItem.userInfo.jobType == .VP
        )
    ) ||
    (self.curShopVisitType == .toBeFilled) ||
    (self.curShopVisitType == .reject)
){
    buttonSection.cellList.append(buttonTableViewCell)
    self.sectionItemList.append(buttonSection)
    self.buttonSectionIndex = self.sectionItemList.count - 1
}
```

```

306     if (self.curShopVisitType == .pass) || (self.curShopVisitType == .reject) {
307         typeSection.cellList.append(typeCell)
308         self.sectionItemList.append(typeSection)
309     }
310
311     if (self.curShopVisitType == .pass) || (self.curShopVisitType == .reject) {
312         dealerTimeSection.sectionHeader = ""
313     }
314     dealerTimeSection.cellList.append(dealerCell)
315     dealerTimeSection.cellList.append(timeCell)
316     self.sectionItemList.append(dealerTimeSection)
317
318     descriptionSection.sectionHeader = ""
319     descriptionSection.cellList.append(descriptionCell)
320     self.sectionItemList.append(descriptionSection)
321
322     lastVisitSection.sectionHeader = "上次走访项"
323     self.sectionItemList.append(lastVisitSection)
324     self.lastVisitSectionIndex = self.sectionItemList.count - 1
325
326     thisVisitSection.sectionHeader = "本次走访项"
327     if (self.curShopVisitType == .toBeFilled) || (self.curShopVisitType == .reject) {
328         thisVisitSection.cellList.append(addProblemCell)
329     }
330     self.sectionItemList.append(thisVisitSection)
331     self.thisVisitSectionIndex = self.sectionItemList.count - 1
332
333     if (
334         (
335             (self.curShopVisitType == .approval) &&
336             (
337                 gCurUserItem.userInfo.jobType == .director ||
338                 gCurUserItem.userInfo.jobType == .VP
339             )
340         ) ||
341         (self.curShopVisitType == .toBeFilled) ||
342         (self.curShopVisitType == .reject)
343     ){
344         buttonSection.cellList.append(buttonTableViewCell)
345         self.sectionItemList.append(buttonSection)
346         self.buttonSectionIndex = self.sectionItemList.count - 1
347     }

```

当传递函数参数太多时，加上适当的换行，以提高代码可读性

对于调用函数参数太多时：

```

let thisTableViewCell = RadioTableViewCell(reuseIdentifier: RadioTableViewCellId, id: item.
id, count: "\(i+1)", title: item.name, content: item.curDescription, checklist: item.checklist
, finish: item.finish, isWrite: isWrite, rightBtnComeback: self.rightDrawComeback)
thisVisitSection.cellList.append(thisTableViewCell)

```

应该给每个参数加上合适的换行，提高代码可读性：

```

let thisTableViewCell = RadioTableViewCell(

```

```
reuseIdentifier: RadioTableViewCellId,  
id: item.id,  
count: "\({i} 1)",  
title: item.name,  
content: item.curDescription,  
checklist: item.checklist,  
finish: item.finish,  
isWrite: isWrite,  
rightBtnComeback: self.rightDrawComeBack  
)  
thisVisitSection.cellList.append(thisTableViewCell)
```

```
188  
189     let thisTableViewCell = RadioTableViewCell(  
190         reuseIdentifier: RadioTableViewCellId,  
191         id: item.id,  
192         count: "\({i+1})",  
193         title: item.name,  
194         content: item.curDescription,  
195         checklist: item.checklist,  
196         finish: item.finish,  
197         isWrite: isWrite,  
198         rightBtnComeback: self.rightDrawComeBack  
199     )  
200     thisVisitSection.cellList.append(thisTableViewCell)  
201 }
```

使得能看清楚具体的参数名和参数值。

crifan.com, 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: 2018-01-25 14:22:09

不同语言的编码规范

不同语言的各有各的官网或非官方的推荐的常见的代码规范/代码风格/编码规范。下面就整理出来，供参考。

Python

Python语言本身就有官网的PEP去定义好的编程规范：

[PEP 8 -- Style Guide for Python Code | Python.org](#)

且 Python 语言本身就把 缩进 作为代码逻辑关系的一部分，而不是像其他语言只是作为代码的格式化和美观方面的考虑因素。

下面是找到的一些 Python 相关的一些编程风格：

- [Python风格规范 — Google 开源项目风格指南](#)
- [Flask 开发团队内部 Python 编码风格指南 - leejun2005的个人页面 - 开源中国社区](#)

Swift

- [aywenderlich.com 的swift的代码编程规范： raywenderlich/swift-style-guide: The official Swift style guide for raywenderlich.com.](#)

Javascript

- 比较有名的airbnb的：[airbnb/javascript: JavaScript Style Guide](#)
- Google的：[Google JavaScript Style Guide](#)
- [idiomatic.js/translations/zh_CN at master · rwaldron/idiomatic.js](#)

Objective-C

- [Objective-C 风格指南 - 内容目录 — Google 开源项目风格指南](#)

C++

- [C++ 风格指南 - 内容目录 — Google 开源项目风格指南](#)

Shell

- [Shell 风格指南 - 内容目录 — Google 开源项目风格指南](#)

crifan.com, 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: 2018-01-11 21:12:39

代码格式化和工具

很多的语言，为了统一编码风格，出现了很多用于格式化代码的工具。整理于此，供参考和使用。

Javascript

- [standard/standard: JavaScript Style Guide, with linter & automatic code fixer](#)

crifan.com, 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: 2018-01-11 21:13:53

附录

此处整理出相关参考资料。

crifan.com, 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: 2018-01-03 10:39:47

代码中常见英文单词的缩写

对于在给变量、函数等起名时，为了避免名字过长，可以考虑将相关英文单词简写。

下面列出一些常见的简写，供参考：

英文全称	英文简写	举例	说明
current	cur 或 curr	curltem 当前的项	
index	idx	curlidx 当前的索引	
previous	prev	prevltem 前一个项	

crifan.com, 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: 2018-01-25 14:24:05

参考资料

crifan.com, 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: 2018-01-03 10:39:54