



iOS工程模块化实践与优化

基于Xcodeproj & Carthage 的持续集成



Amateurish VS Professional

Xcodeproj => Makefile => Gradle

Imperative => Declarative

Everyone Can Code

<https://www.apple.com/everyone-can-code/>

```
wc -l tools/template/universal-project.pbxproj
1793 tools/template/universal-project.pbxproj
```

```
151 install: install-progs install-dato
152 install:
153
154 install: install-libs install-headers
155
156 install-libs: install-libs-yes
157
158 install-progs-yes:
159 install-progs-$(CONFIG_SHARED): install-libs
160
161 install-progs: install-progs-yes $(AVRPROG)
162     $(CXX) $(CXXFLAGS) $(CXXLIBS)
163     $(INSTALL) -c -m 755 $(OBSOLETE) "$(CXXLIBS)"
164
165 install-dato: $(DATA_FILES) $(EXAMPLE_FILES)
166     $(CXX) $(CXXFLAGS) $(CXXLIBS) $(CXXLIBS)
167     $(INSTALL) -m 644 $(DATA_FILES) "$(DATA_DIR)"
168     $(INSTALL) -m 644 $(EXAMPLE_FILES) "$(DATA_DIR)/examples"
169
170 uninstall: uninstall-libs uninstall-headers uninstall-progs uninstall-dato
171
172 uninstall-progs:
173     $(RM) $(addprefix "$(OBJDIR)/", $(ALL_AVRPROGS))
174
175 uninstall-dato:
176     $(RM) -r "$(DATA_DIR)"
177
178 clean:
179     $(RM) $(ALL_AVRPROGS) $(ALL_AVRPROG_O)
180     $(RM) $(CCELEANUPFILES)
181     $(RM) $(CCELEANUPFILES.$(TOOLSOBJ))
182     $(RM) -r $(coverage-html) $(coverage-info) $(lev)
183     $(RM) -r $(coverage.html) $(coverage.info) $(lev)
```

```
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

Pbxproj & Makefile & Gradle

Pain

Merge hell



All in one

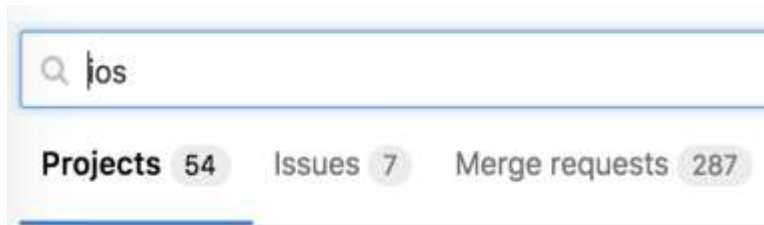
Showing **705 changed files** ▼ with **2061 additions** and **441 deletions**



Pain

version	developers	source code (oc)	date
v2.1.3	1	119	2013.09.15
v3.6.3	3	441	2015.03.04
v4.23	9	1438	2016.06.24
v4.38	40+	3000+	2017.02.22
v5.18	120+	6000+	2017.10.16

54个项目仓库(oc/c++/c)



3个应用



暗中观察

Modular Design

1. Components
2. Data bus
3. Router
4.





Carthage / CocoaPods

Differences between Carthage and CocoaPods

[CocoaPods](#) is a long-standing dependency manager for Cocoa. So why was Carthage created?

Firstly, CocoaPods (by default) automatically creates and updates an Xcode workspace for your application and all dependencies. Carthage builds framework binaries using xcodebuild, but leaves the responsibility of integrating them up to the user. CocoaPods' approach is easier to use, while Carthage's is flexible and unintrusive.

The goal of CocoaPods is listed in its [README](#) as follows:

... to improve discoverability of, and engagement in, third party open-source libraries, by creating a more centralized ecosystem.

By contrast, Carthage has been created as a *decentralized* dependency manager. There is no central list of projects, which reduces maintenance work and avoids any central point of failure. However, project discovery is more difficult—users must resort to GitHub's [Trending](#) pages or similar.

CocoaPods projects must also have what's known as a [podspec](#) file, which includes metadata about the project and specifies how it should be built. Carthage uses xcodebuild to build dependencies, instead of integrating them into a single workspace, it doesn't have a similar specification file but your dependencies must include their own Xcode project that describes how to build their products.

Ultimately, we created Carthage because we wanted the simplest tool possible—a dependency manager that gets the job done without taking over the responsibility of Xcode, and without creating extra work for framework authors. CocoaPods offers many amazing features that Carthage will never have, at the expense of additional complexity.



CocoaPods

优点

1. 大部分第三方库都写好了 podspec (模块的导出)
2. 管理工具方便 pod

缺点

1. 基于源码编译(无法享受incremental build)
2. 污染Xcode编译中间文件, 导致各种奇怪的编译错误需要频繁清理cache
3. 难以(无法)管理resource
4. 维护pod困难 (local repo , podspec书写...)

Carthage

优点

恰到好处

缺点

恰到好处Orz

```
Available commands:

archive      Archives built frameworks into a zip that Carthage can use
bootstrap   Check out and build the project's dependencies
build       Build the project's dependencies
checkout    Check out the project's dependencies
copy-frameworks  In a Run Script build phase, copies each framework specified by a
fetch       Clones or fetches a Git repository ahead of time
help        Display general or command-specific help
outdated    Check for compatible updates to the project's dependencies
update      Update and rebuild the project's dependencies
version     Display the current version of Carthage
```

What bilibili do for Carthage

Central module server (macross)

1. Download

- a. 一个文件服务来描述模块列表<https://github.com/Carthage/Carthage/blob/master/Documentation/Artifacts.md#cartfile>

2. Upload

- a. 一个jenkins用来模块构建及静态检查及模块上传服务

3. Dashboard

- a. 一个用来监控模块体积，异常依赖的查询服务

Name	Code	Create Time	Details	依赖项	依赖项
lib	lib	2018-04-11 10:00:00	lib 1.0.0 lib 1.0.0	lib 1.0.0 lib 1.0.0	lib 1.0.0 lib 1.0.0
lib	lib	2018-04-11 10:00:00	lib 1.0.0 lib 1.0.0	lib 1.0.0 lib 1.0.0	lib 1.0.0 lib 1.0.0
lib	lib	2018-04-11 10:00:00	lib 1.0.0 lib 1.0.0	lib 1.0.0 lib 1.0.0	lib 1.0.0 lib 1.0.0
lib	lib	2018-04-11 10:00:00	lib 1.0.0 lib 1.0.0	lib 1.0.0 lib 1.0.0	lib 1.0.0 lib 1.0.0

What is module

1. Semver (<https://semver.org/>)
2. 依赖关系 (当次构建的依赖 & 对依赖的兼容关系)
3. 更新日志

1. Library
 - a. 自定义 dylib 或者 static lib的方式自由集成
2. Resource
 - a. 独立的资源空间，解决命名冲突
3. Symbol
 - a. 统一构建拥有完整的调试符号，可以在ide中debug step into代码

```

1 module directory structure
2
3 BBAAd
4   - news.md
5   - Cartfile
6   - Cartfile.resolved
7   - BBAAd.framework
8     - BBAAd (binary)
9     - Assets.car (resource)
10    - Heads (head file)
11    - Modules (module define)
12    - Info.plist (plist)
13    - x.png (resource)
14    - x.nib (resource)
15    - BBAAd.framework.dsym (symbol)
16
  
```

XcodeProj

<https://github.com/CocoaPods/Xcodeproj>

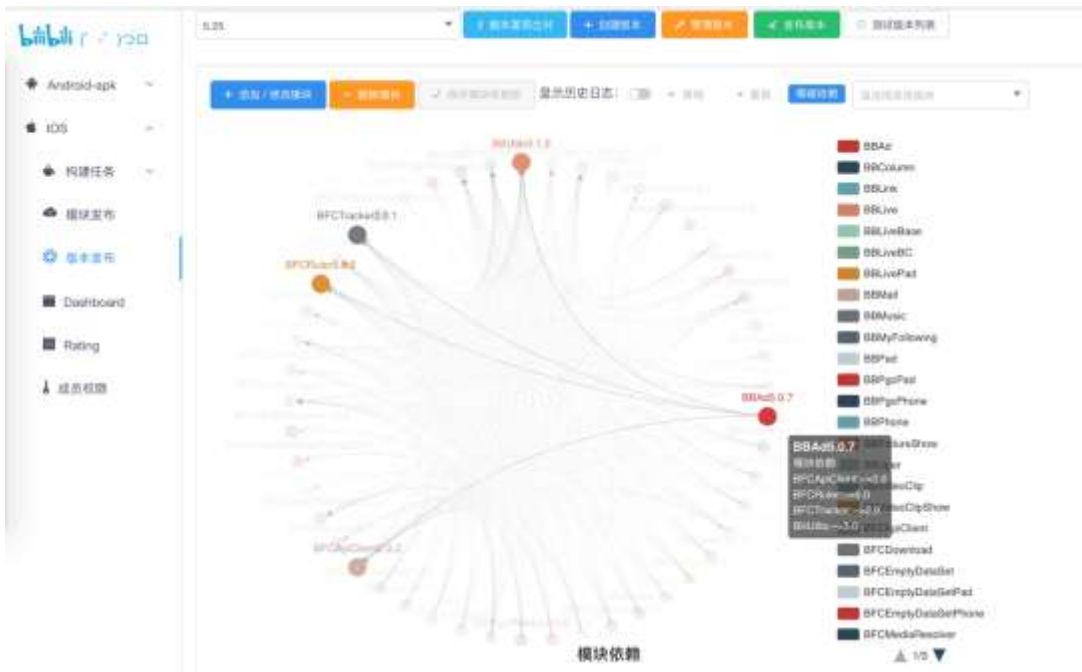
Gradle VS project.pbxproj (fuck it!)

1. Build
 - a. Feature (lldb, editor, profile...)
 - b. Source code OR Pre compile
2. Merge
 - a. aiuhfsjklfhjagbgaf ! ! (remote ref, local ref, build phase, group,)



```
zhangxinzheng in ~: ❯ cat .gitignore | grep pbxproj
bili-universal/bili-universal.xcodeproj/project.pbxproj
bili-blue/bili-blue.xcodeproj/project.pbxproj
bili-ipad/bili-ipad.xcodeproj/project.pbxproj
```

Dependencies Management



Bilibili Project Structure



Main:

1. Initialize const variable
2. Initialize base lib (BFC)
3. Initialize module bus



Build Step

1. Generate project.pbxproj (sh build.sh)
2. Generate ipa (xcodebuild release)
 - a. Compile modules
 - b. Compile target / extensions
 - c. Link target / extensions
 - d. Copy resource
 - e. Delete binary
 - f. Resign

What bilibili do for CI

1. ignore project配置文件，基于Cartfile内容，分别相应的生成，xcode的目录结构依赖以及buildphase
2. 基于不同的target (ipa, extension) 通过生成脚本自由生成不同业务角色最合适的项目项目
 - a. Cartfile.filter('git').addToReference.addToLinkPhases.addToCopyPhases
 - b. Cartfile.filter('binary').addToLinkPhases.addToCopyPhases
3. 一系列辅助工具使得开发与发布全自动化(strip, sign...)
 - a. 根据module内的binary属性，使用不同的链接方式(dy,static)
 - b. 根据构建的不同类型，选择是否裁切相应架构 (x86, arm)
 - c. 根据构建的不同类型，选择相应的辅助工作的执行，符号上传，签名等
4. 模块/版本发布工具 Macross



More declarative (TODO)

1. 把最小集成单元继续从模块往下降成单个代码文件，单个函数 (模块级别的CI -> 文件级别的CI)
2. 进一步降解xcodeproj，使xcode最终退化成编辑器及调试器
3. 通过xcrun toolchain定制一套构建对构建速度进行极限优化（去除module server, 去除 version)

吐槽

1. Carthage的版本冲突计算由于会递归自己历史版本算真的很慢很慢 ==
2. Carthage的寻找shared scheme的条件实在有点呆萌 ==
3. Xcode 的 static framework的link方式真的很迷很迷 ==
4. Xcode的增量编译真的很蠢很蠢 ==





Q/A