

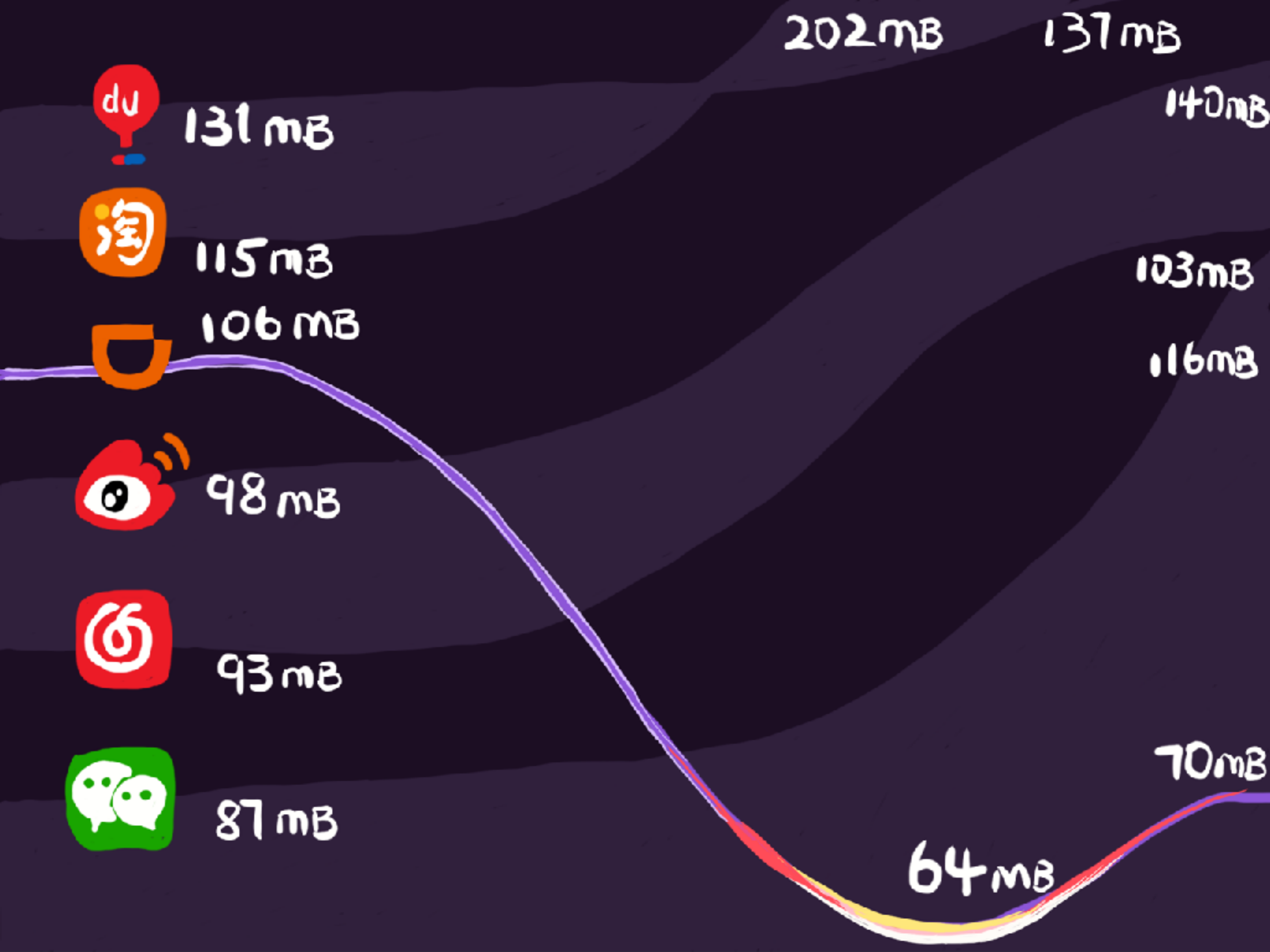
滴滴出行 iOS 端瘦身实践

戴铭 / 技术专家



福身背景





202MB

137MB



131MB



115MB



106MB



98MB



93MB



87MB

140MB

103MB

16MB

70MB

64MB

资源

Resource

瘦身

瘦身

01 获取资源文件

find 命令

```
find /Users/daiming/Project/ -name
```

03 正则匹配图片名

pattern = @"@"(.*?)\."

05 匹配编号规则

@ "image_%d"

02 设置资源类型

jpg

gif

png

webp

04 集合取差集

setS

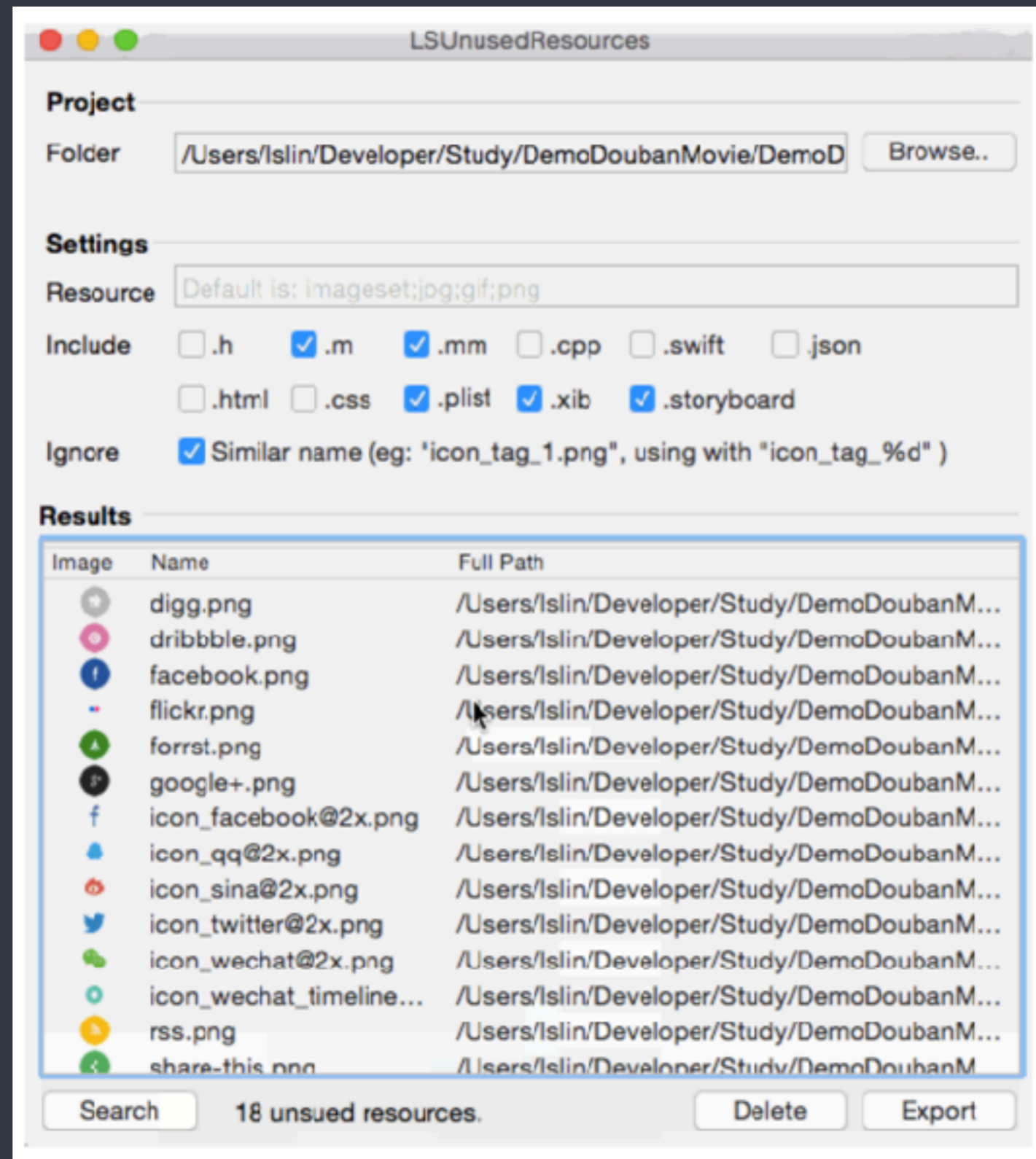
06 删除无用图片

! project.pbproj

NSFileManager



LSUnusedResources



<https://github.com/tinymind/LSUnusedResources>

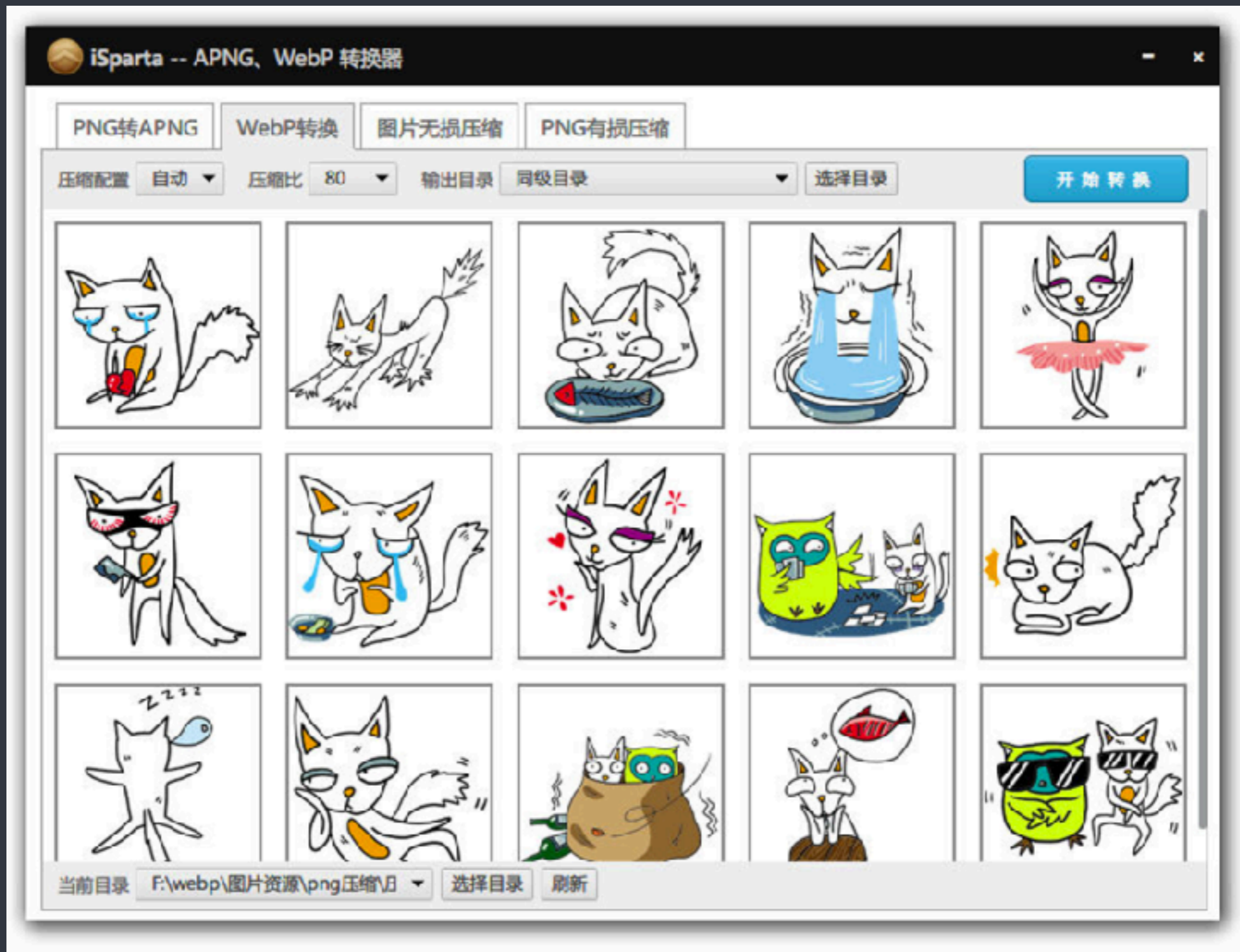


FengNiao

<https://github.com/onevcat/FengNiao>

大比例压缩

PNG 转 WebP



PNG 转换以及压缩工具iSparta - <http://isparta.github.io/>

WebP 项目主页 - <https://developers.google.com/speed/webp/>

iOS WebP 解析库 - <https://github.com/carsonmcdonald/WebP-iOS-example>

为何使用 Webp

Webp **压缩率**高，支持有损与无损压缩

WebP **体积**大幅减少，肉眼看不出差异

WebP 支持 **Alpha** 透明和 **24-bit** 颜色数，不像 **PNG8** 色彩不够出现毛边

Gif 转 Animated WebP 有损可减少 **64%**，无损 **19%**

小于 256 色适合无损压缩，压缩率高，参数使用 -lossless -q 100

大于 256 色使用 75% 有损压缩，参数使用 -q 75

远大于 256 色使用 75% 以下压缩率，参数 -q 50 -m

6

WebP 的缺点

较 PNG 消耗2倍左右 CPU 和 解码时间

全平台支持度不够。不过在 iOS 上可以通过对应的 iOS 的 WebP 解析库解决

大资源文件

比如表情包下载后使用

收益一般的方法

重复资源检测: <https://github.com/adrianlopezroche/fdupes>

音频压缩瘦身: <http://trac.ffmpeg.org/wiki/CompilationGuide/MacOSX>

简单图片使用**代码替换**

将代码里的静态**字符串**抽取出来放到静态文件里

基于编译后的瘦身

Link Map

Object File

LibNetwork.a
SMRequest.o
[3064]
...

Section

_text
_cstring
_objc_ivar
_objc_protorets

Symbols

0x1012D5CBC 0x000000B0
-[SMRequest cancel]

0x1040FDE0C 0x00000004
_OBJC_IVAR_
\$_SMNetwork._share

Mach-O

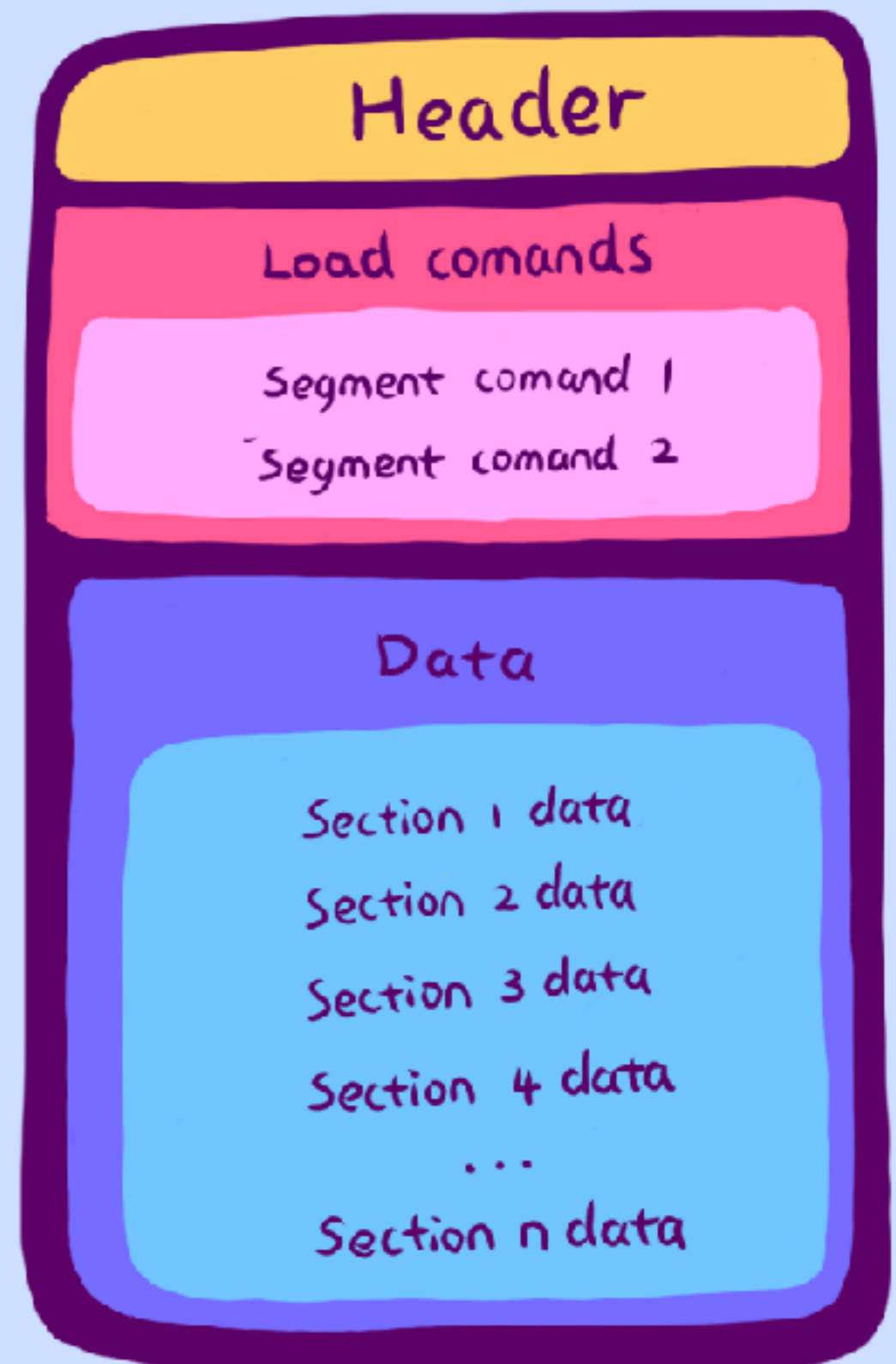
_DATA _objc _selrefs

otool

otool -v -s _DATA

_objc _selrefs Mach-O

0000001040da4f0 _TEXT: _objc_methname:cancel
0000001040da4f8 _TEXT: _objc_methname:request:





—



=



基于编译过程的 Clang Plugin 瘦身

能做的事情

分析调用关系，找出没被调用的代码

原理

编写分析全部源码的插件

编译过程中将插件作为 Clang 参数载入生成中间文件

编写工具分析所有的方法有哪些是会被调用

通过 Clang 遍历语法树 获取嵌套访问关系

```
@interface ViewController : UIViewController

@end

@implementation ViewController

- (void)viewDidLoad {

    [super viewDidLoad];

    [self.view setBackgroundColor:[UIColor redColor]];

}

@end
```

则称:-[ViewController viewDidLoad]调用了:

```
-[UIViewController viewDidLoad]
-[ViewController view](语法糖)
+[UIColor redColor]
-[UIView setBackgroundColor:]
```


设计插件数据结构

类接口与继承体系

ObjCInterfaceDecl (接口声明)

ObjCCategoryDecl (分类声明)

ObjCPropertyDecl (属性声明)

ObjCMethodDecl (方法声明)

协议的接口与继承体系

ObjCProtocolDecl (协议声明)

ObjCPropertyDecl (属性声明)

ObjCMethodDecl (方法声明)

接口方法调用

正常的 - -/+**[Class method:*]**

NSObject 协议的 **performSelector** 方法簇

手势/按钮的事件处理 **selector**

NSNotificationCenter 添加通知处理 **Selector**

UIBarButtonItem 添加事件处理 **Selector**

Timer

NSThread

CADisplayLink

KVO 机制

IBAction 机制

[XXX new] - 包含 **+ [XXX alloc]** 和 **- [XXX init]**

编写插件

编写插件步骤

自定义继承

clang::PluginASTAction (基于 consumer 的抽象语法树(Abstract Syntax Tree/AST) 前端 Action 抽象基类)

clang::ASTConsumer (用于客户读取抽象语法树的抽象基类),

clang::RecursiveASTVisitor (前序或后续地深度优先搜索整个抽象语法树, 并访问每一个节点的基类)等基类。

根据自身需要重载一下方法 实现自定义的分析逻辑

PluginASTAction::CreateASTConsumer

PluginASTAction::ParseArgs

ASTConsumer::HandleTranslationUnit

RecursiveASTVisitor::VisitDecl

RecursiveASTVisitor::VisitStmt

插件生成与集成

注册插件: static

```
FrontendPluginRegistry::Add<MyPlugin>  
X("my-plugin-name", "my-plugin-description");
```

编译生成插件(dylib)

使用 **XcodeHacking** 是插件与Xcode集成

示例: [https://github.com/kangwang1988/
XcodeZombieCode](https://github.com/kangwang1988/XcodeZombieCode)

代码级瘦身

AppCode 清理无用的类

The screenshot shows the IDE interface for a project named 'HomePageTest'. The left sidebar displays the project structure, including folders for Frameworks, HomePageTest, HomePageTestTests, HomePageTestUITests, Pods, and Products. The main area shows the 'Inspection Results of 'Project Default' Profile on Project: HomePageTest'. The results are categorized under 'Unused import statement' with 173 warnings. A list of files and their respective warning counts is shown, with 'AppDelegate.m' highlighted, indicating 8 warnings. The right pane displays the code for 'AppDelegate.m', showing several unused import statements that are highlighted in blue:

```
#import "AppDelegate.h"
#import "SMLagMonitor.h"
#import "testTableViewCell.h"
#import "RCTestViewController.h"
#import "PraiseViewController.h"
#import "GroupListViewController.h"
#import "HomeController.h"
#import "WebCacheViewController.h"
#import "SecondViewController.h"

#import <FBMemoryProfiler/FBMemoryProfiler.h>
#import "CustomURLCache.h"
```

The bottom status bar indicates 'Platform and Plugin Updates: AppCode is ready to update. (today 15:36)'.

AppCode 还能清理什么

▼ Unused code 1142 warnings

- ▶ Unused class 31 warnings
- ▶ Unused global declaration 34 warnings
- ▶ Unused import statement 173 warnings
- ▶ Unused instance variable 28 warnings
- ▶ Unused local variable 8 warnings
- ▶ Unused macro 227 warnings
- ▶ Unused method 470 warnings
- ▶ Unused parameter 31 warnings
- ▶ Unused property 129 warnings
- ▶ Unused value 11 warnings

更多功能

Inspection Results of 'Project Default' Profile on Project 'HomePageTest'

- ▶ **Clang analyzer issue** 8 errors
- ▶ **Clang compiler issues** 21 errors
- ▶ **Classes** 8 warnings
- ▶ **Code style issues** 4 warnings
- ▶ **Data flow analysis** 79 warnings
- ▶ **Declaration order** 16 errors 75 warnings
- ▶ **Functions** 1 warning
- ▶ **General** 110 errors 2644 warnings
- ▶ **JavaScript validity issues** 5 warnings
- ▶ **Message resolution** 9 warnings
- ▶ **Methods** 8 warnings
- ▶ **Potentially confusing code constructs** 3 warnings
- ▶ **Properties** 4 warnings
- ▶ **Spelling** 5238 typos
- ▶ **Type checks** 27 errors 141 warnings
- ▶ **Unused code** 1142 warnings

AppCode 的问题

JSONModel 里定义了未使用的协议会被判定无用协议

如果子类使用了父类的方法不会被认为使用

通过点使用属性会被认为没有使用

UITableView **registerClass** 的问题。自定义Cell 在 tableView registerClass 但会被认为没用

使用 **NSClassFromString** 的情况查不出来，比如 `layerClass = NSClassFromString(@"SMFloatLayer");`

使用 `[[self class] accessToken]` 这样的使用类方法的会被认为没有用

运行时比如 `self performSelector:@selector(arrivalRefreshTime)` 检测不出

结构瘦身

基于字符查找相似代码

SameCoderFinder

SameCodeFinder 可以在源代码文件中检测到相同的 **function**。可以显示两个 **function** 之间的 **Hamming** 距离。

找到需要提取重复使用的相同代码

显示每个源文件之间的 **Hamming** 距离（支持各种 **soucecode** 类型）

显示每个源文件 **function** 之间的 **Hamming** 距离（现在支持 **Java** 和 **Objective-C**）

GitHub地址：<https://github.com/startry/SameCodeFinder>

Simhash

确定simhash的**位数**，比如说32位

将simhash的各位**初始化为0**

提取原始文本中的特征，一般采用各种分词的方式。比如对于"the cat sat on the mat"，采用两两分词的方式得到如下结果：`{"th", "he", "e ", " c", "ca", "at", "t ", " s", "sa", " o", "on", "n ", " t", " m", "ma"}`

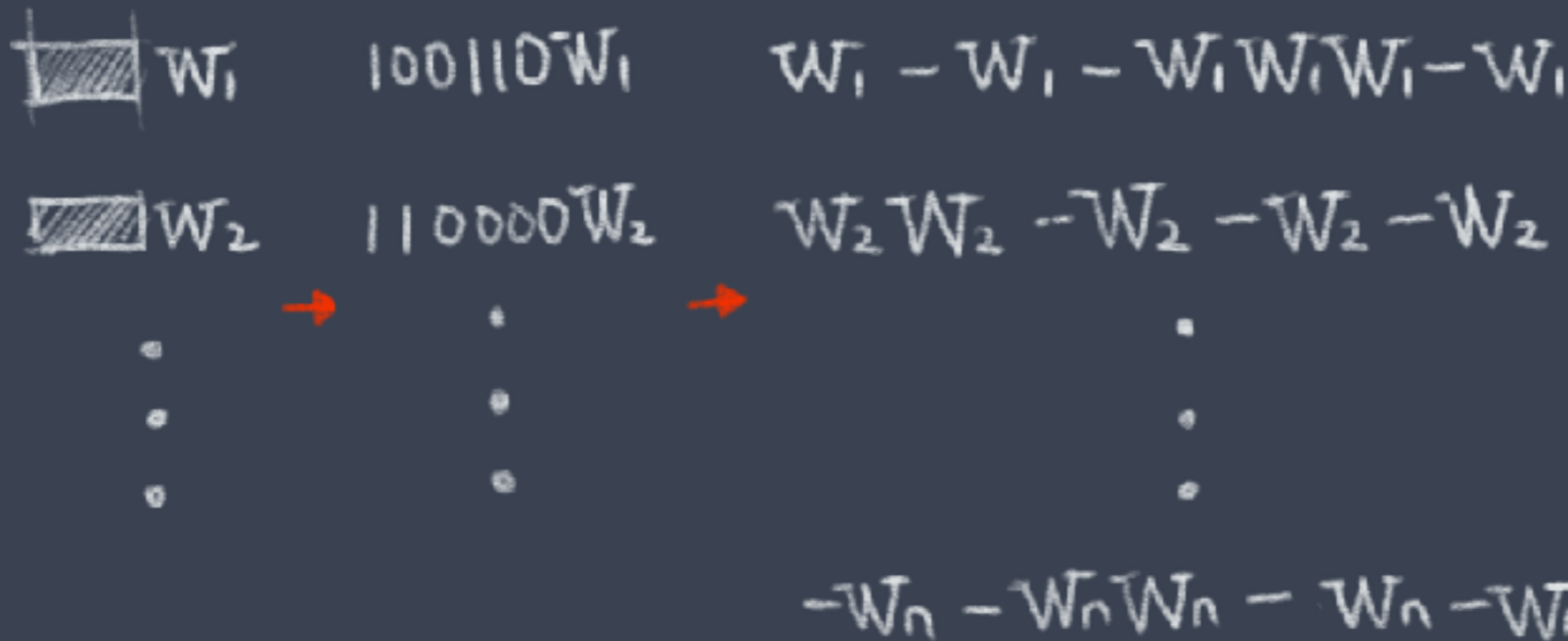
使用传统的32位hash函数计算各个word的hashcode，比如：`"th".hash = -502157718`，`"he".hash = -369049682`

然后对 hash_weight_pairs (5 -5 5 -5 5 5) 进行位的纵向累加，如果**该位是1**，则**+weight**，如果是**0**，则**-weight**，最后生成 bits_count 个数字，如图所示是[13, 108, -22, -5, -32, 55]，这里产生的值和hash函数所用的算法相关

[13,108,-22,-5,-32,55] -> 110001这个就很简单啦，**正1负0**

Doc

Simhash



ADD

13, 108, -22, -5, -32, 55 **SIGN** 110001

simhash值的海明距离计算

二进制串A 和 二进制串B 的海明距离 就是 $A \text{ xor } B$ 后二进制中1的个数

$A = 100111;$

$B = 101010;$

$\text{hamming_distance}(A, B) = \text{count_1}(A \text{ xor } B) = \text{count_1}(001101) = 3;$

比如

p1: the cat sat on the mat

p2: the cat sat on a mat

p3: we all scream for ice cream

p1.simhash => 851459198

00110010110000000011110001111110

p2.simhash => 847263864

00110010100000000011100001111000

p3.simhash => 984968088

00111010101101010110101110011000

(p1,p2)=4

(p1,p3)=16

(p2,p3)=12

两个 Paper

《**Detecting Near-Duplicates For Web Crawling**》 [http://wwwconference.org/
www2007/papers/paper215.pdf](http://wwwconference.org/www2007/papers/paper215.pdf)

《**Similarity estimation techniques from rounding algorithms**》 [http://
www.cs.princeton.edu/courses/archive/spr04/
cos598B/bib/CharikarEstim.pdf](http://www.cs.princeton.edu/courses/archive/spr04/cos598B/bib/CharikarEstim.pdf)

对于大量代码比较
如何提高效率

高效计算二进制序列中1的个数

```
bool isEqual(uint64_t lhs, uint64_t rhs, unsigned short n = 3)
{
    unsigned short cnt = 0;
    lhs ^= rhs;
    while(lhs)
    {
        lhs &= lhs - 1;
        cnt++;
    }
    if(cnt <= n)
    {
        return true;
    }
    return false;
}
```

$O(n)$ 算法, $n = 1$ 的个数

110

$110 \& 101 = 100$ 计数器加1。

$100 \& 011 = 000$ 计数加1, 为0结

束, 1个数为2

$O(1)$

根据位数比如8位的话就列出1~256每个数中1的个数，进行打表，典型的空间换时间

```
Int count (Int num)
```

```
{
```

```
    Int sum[256] = {0, 1, 1, 2, 1, 2.....6, 7,  
7, 8};
```

```
    Return(num);
```

```
}
```

其它算法介绍

百度的去重算法，找出此文章的最长的n句话，做一遍hash签名。n一般取3。准确率达80%以上

shingle算法，原理复杂，偏学院派，大量数据时速度慢

基于代码结构 查找相似代码

基于代码行的

基于标识符 (**Token**) 的

基于度量 (**Metrics**) 的。度量指利用代码的一些特征而构成的指标

基于抽象语法树 (**AST**) 的。论文《Clone Detection Using Abstract Syntax Trees》

基于程序依赖图 (**Program Dependence Graph, PDG**) 的。论文《Revisiting Capability of PDG-based Clone Detection》

将上面的方案
统一成一个系统

系统需要解决的问题

上面查找无用资源时，需要变量值路径检索来保证更高的命中率

Mach-O 方式和 AppCode 方式同样需要变量值路径来解决运行时字符串值正确率

方便集中操作，减少流程

int num = func(local1, &local2, localA1, localB)

local1 : 1 ←

local2 : 2

local3 : 3

localN1-next : &localN2

localN1-value : 10

localN2-next : 0

localN2-value : 20

localA1-n-next : localN2

localA1-n-value : 10

localA1-count : 0

localB-memptr : &local3 ←

global : 10

路径引擎

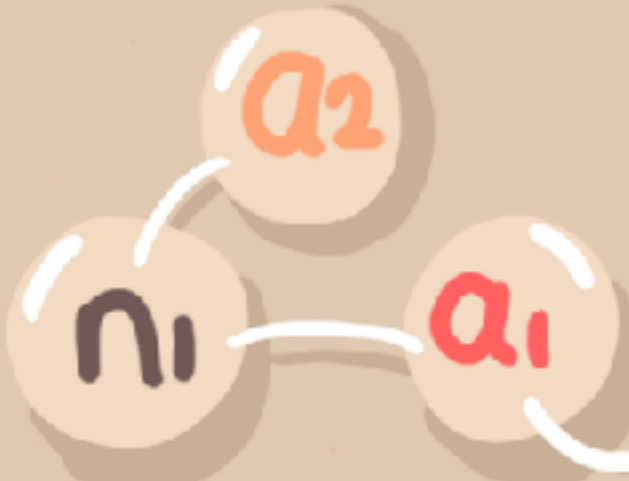
内存模型

Region
The Living

$a_1, 0, \text{direct} = \text{conj} - \2
 $a_1, 0, \text{direct} = \&n_1.\text{value}$

$\langle a_2 = \&a_2 \rangle$ Live
 $\langle a_1 = \&a_1 \rangle$ Dead
 $\langle a_1 = \text{lazyCompound} \rangle$ Live

```
struct A a2;  
a2 = a1;  
return a2;
```



scan

get

while

工程
Code

OC

Swift

More

Struct

Command
Line

统一结构

词法
分析

值路径

内存模型

macOS
App

无用
资源

无用
代码

相似
代码

THANKS!

