

# 寻梦记账-自动记账

市面上自动记账已经非常多了，主要分为两种

- AI聊天自动记账
- 快捷指令-自动记账

因为寻梦记账的产品规划和成本问题，所以寻梦记账并没有接入AI聊天自动记账，根本原因就是我们认为AI自动记账的对于「寻梦记账」的价值还不值得我们冒险

优缺点大致如下

	优点	缺点
AI聊天记账	分类猜测较准确、可以猜测账户（不一定准确）、备注猜测较准、网络好且猜测准确的时候记录较快	成本较高
截屏自动记账	无成本	不够准确

看似AI记账完胜，但是实际上这就像我在初学进程调度的FIFO的时候，总觉得这个调度算法毫无意义，但实际上在「寻梦记账」里，我们有用到LRU，也有用到FIFO，所以最重要的还是产品的场景和定位

而寻梦记账的定位就是，记账要准确，不要快，而准确最好的方式就是自己记。寻梦记账清晰的区分了：名称、备注、分类、标签、分类组、账户、账本。这些都是不是毫无意义或者繁琐的，我们在产品设计上尽量把这些复杂的功能规划设计的好用，因为只有记录的时候记好，才能在分析的时候分析好，如果你记账都不是为了分析复盘，记账就毫无意义，还不如不记 😊↔️

所以我们的计划就是：

- 能提供一个快速打开APP记账的方法，而不是消费后还要找到APP再打开，再进入记账页面
- 能提供基础的尽量准确的识别，也就是尽量识别日期、名称、金额
- 然后直接收集信息跳转APP，不在快捷指令上让用户选择（这是另一些app做的事，这有点无意义）

有趣的是，我们这种简单直接的方式在上线后收到了很多用户的好评 ☺

对于聊天自动记账这种提供情绪价值的记账方式就更不是我们考虑的范围了，苹果输入法本来就不好用，把所有的上下文打出来都已经能记好几笔账了

## 1. 前言

在网络上，并没有找到任何关于自动记账的博客，我们只能使用GPT辅助+自己摸索，我们最后使用的方式是

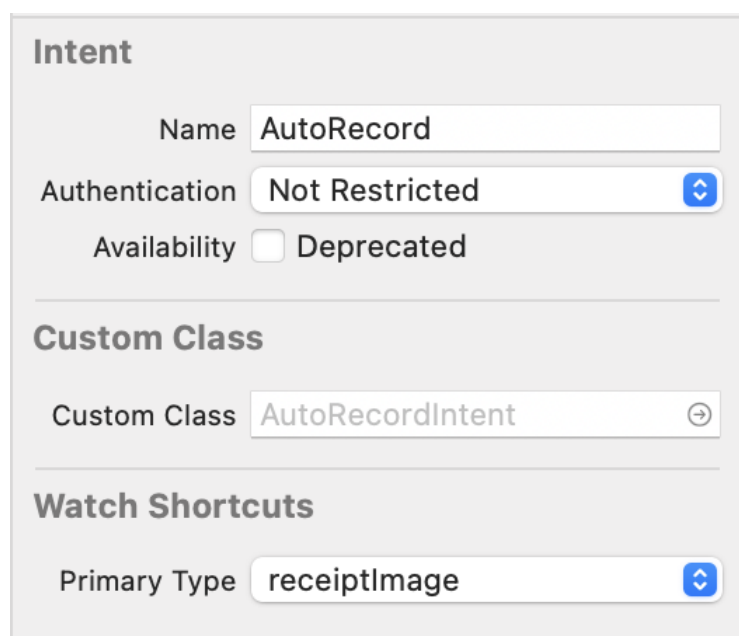
1. Siri Intents先拿到截图
2. 跳转主APP，通过App Groups的方式进行进程之间的通信，把信息传递给主App
3. ORC分析截图每一行文字
4. 代码逻辑+简单的正则表达式分析出主要的信息

## 2. Siri Intents

### 参考文章

具体的创建步骤是

- 新建一个 Intent Definition File，定义 Intent 名字、参数（如金额、日期）、语音短语（Siri 的建议短语）
  - 保存 .intentdefinition 后，**Xcode 会自动生成一个对应的 Intent 类**。这个 Extension 会包含你的 Intent 处理代码
  - 点击 .intentdefinition 在右侧 navigation bar 中的 custom class 可以看到，也可以跳转



**Intent**

Name

Authentication

Availability ☐ Deprecated

---

**Custom Class**

Custom Class

---

**Watch Shortcuts**

Primary Type

- 我们使用 .intentdefinition 主要是定义了名称，和参数类型，也就是一个 receiptImage，类型是 File，而 File 的类型是 Image，具体是这样

Parameters

Parameter

F receiptImage

Display Name

Receipt Image

Type

File

Array

☐ Supports multiple values

Configurable

☒ User can edit value in Shortcuts, widgets, and Add to Siri

Resolvable

☐ Siri can ask for value when run

Dynamic Options

☐ Options are provided dynamically

Relationship

Parent Parameter

None

File Type

Type

Image

+ -

- 打开主APP Capabilities里面的Siri支持，还有App Groups的支持，在创建Siri Intent Extension之后同样需要打开App Groups
- 创建Siri Intens Extension，这是一个Target
  - 创建好后需要填写支持的Intents类型，也就是刚才创建的AutoRecordIntent
- 这个target其实非常简单，只需要做两件事
  - 创建一个CustomIntentHandler根据拿到的参数处理业务逻辑
  - 在创建好的IntentHandler里返回这个CustomIntentHandler即可

代码块

```
1  @implementation IntentHandler
2
3  - (id)handlerForIntent:(INIntent *)intent {
4      return [[AutoRecordIntentHandler alloc] init];
5  }
6
7  @end
```

## CustomIntentHandler

代码块

```
1  #import <Foundation/Foundation.h>
2  #import "AutoRecordIntent.h"
3
4  NS_ASSUME_NONNULL_BEGIN
5
6  @interface AutoRecordIntentHandler : NSObject <AutoRecordIntentHandling>
7
8  @end
9
10 NS_ASSUME_NONNULL_END
```

```

11
12 #import "AutoRecordIntentHandler.h"
13
14 @implementation AutoRecordIntentHandler
15
16 - (void)handleAutoRecord:(nonnull AutoRecordIntent *)intent completion:(nonnull
    void (^)(AutoRecordIntentResponse * _Nonnull))completion {
17     INFile *receipt = intent.receiptImage;
18
19     NSURL *sharedURL = [self saveImageToSharedContainer:receipt.data];
20     if (!sharedURL) {
21         AutoRecordIntentResponse *response = [[AutoRecordIntentResponse alloc]
    initWithCode:AutoRecordIntentResponseCodeFailure userActivity:nil];
22         completion(response);
23         return;
24     }
25
26     NSUserActivity *userActivity = [[NSUserActivity alloc]
    initWithActivityType:@"com.someDomain.processImage"];
27     userActivity.userInfo = @{@"imageURL": sharedURL.absoluteString};
28
29     AutoRecordIntentResponse *response = [[AutoRecordIntentResponse alloc]
    initWithCode:AutoRecordIntentResponseCodeContinueInApp
    userActivity:userActivity];
30     completion(response);
31 }
32
33 - (NSURL *)saveImageToSharedContainer:(NSData *)imageData {
34     NSFileManager *fileManager = [NSFileManager defaultManager];
35     NSURL *groupURL = [fileManager
    containerURLForSecurityApplicationGroupIdentifier:@"group.com.someDomain"];
36     if (!groupURL) {
37         NSLog(@"无法获取 App Group 共享容器 URL");
38         return nil;
39     }
40
41     NSString *fileName = [NSUUID UUID].UUIDString;
42     NSURL *fileURL = [groupURL URLByAppendingPathComponent:[NSString
    stringWithFormat:@"%@.png", fileName]];
43
44     NSError *error = nil;
45     BOOL success = [imageData writeToURL:fileURL options:NSDataWritingAtomic
    error:&error];
46     if (!success) {
47         NSLog(@"保存图片失败: %@", error.localizedDescription);
48         return nil;
49     }

```

```

50
51     return fileURL;
52 }
53
54 @end

```

- AutoRecordIntent: 这个文件是在创建好intentDefination文件之后自动创建的，主要是要让 CustomIntentHandler conform这个协议，协议只有一个方法，就是handleAutoRecord
- AppGroups: 主App和新建的target在iOS系统中实际上是两个不同的进程，这里主要通过App Groups写入文件实现进程之间的通信，在这里把image存入文件系统，然后在主app里再拿出来
- 如果App Groups保存成功，就使用userActivity打开主app；如果失败，就让Intent报错

### 3. UserActivity

当 App 被 **Siri Shortcut**、**Universal Link**、**NSUserActivity** 等方式唤起时，系统会调用此方法，把对应的 NSUserActivity 作为参数传递进来。

可以在这里根据 userActivity 内容恢复页面、处理跳转、获取 Siri Intents 数据。

不过需要先在主App的info.plist里注册这个activity，不然无法处理

代码块

```

1     <key>NSUserActivityTypes</key>
2     <array>
3         <string>AutoRecordIntent</string>
4         <string>com.someDomain.processImage</string>
5     </array>

```

sceneDelegate中处理这个Activity

代码块

```

1  - (void)scene:(UIScene *)scene continueUserActivity:(NSUserActivity
    *)userActivity API_AVAILABLE(ios(13.0)) {
2      if ([userActivity.activityType
    isEqualToString:@"com.someDomain.processImage"]) {
3          NSString *imageURLString = userActivity.userInfo[@"imageURL"];
4          if (imageURLString) {
5              NSURL *imageURL = [NSURL URLWithString:imageURLString];
6              if (imageURL) {
7                  [self _recordBillWithImageURL:imageURL];
8              }
9          }
10     }

```

```
11 }
```

- 这个方法也很简单，从userInfo中取出数据，如何做进一步处理，不过这里还是会有一点点问题，既然这里能取到image，为什么还要App Groups？
- 这是因为，这个方法只能在进程运行在后台的时候调用，如果进程已经被杀死，Siri Intents只能打开App，不会调用这个方法

代码块

```
1 - (void)scene:(UIScene *)scene willConnectToSession:(UISceneSession *)session
  options:(UISceneConnectionOptions *)connectionOptions
  API_AVAILABLE(ios(13.0)) {
2     KTEexecuteOperationInBackground(^{
3         [self _openRecordControllerIfNeeded];
4     });
5 }
```

- 在每次app打开的时候，先判断App Groups里有没有存储图片，如果有就处理自动记账逻辑  
处理自动记账逻辑，

代码块

```
1 - (void)_recordBillWithURL:(NSURL *)imageUrl {
2     dispatch_async(dispatch_get_global_queue(QOS_CLASS_USER_INITIATED, 0), ^{
3         NSData *imageData = [NSData dataWithContentsOfURL:imageURL];
4         UIImage *image = [UIImage imageWithData:imageData];
5         [self clearSharedContainer];
6         if (image) {
7             WS
8             [KTORCProcessor performOCR:image completion:^(NSDictionary
  *results, NSError *error) {
9                 if (!error) {
10                     dispatch_async(dispatch_get_main_queue(), ^{
11                         UIViewController *container = nil;
12                         UIViewController *currentController = [[SourceManager
  sharedInstance] getCurrentController];
13                         if (currentController != nil) {
14                             container = currentController;
15                         }
16
17                         if (container == nil) {
18                             [weak_self.navigationController
  popToRootViewControllerAnimated:NO];
19                             container = weak_self.detailsController;

```

```

20         }
21         [RecordViewController
showWithSuperViewController:container];
22     });
23     } else {
24         NSLog(@"OCR 处理错误: %@", error.localizedDescription);
25     }
26     }];
27 } else {
28     NSLog(@"无法从 URL 读取图像");
29 }
30 });
31 }

```

- 先进行必要的转换，清理App Groups中共享的数据
- 根据SourceManager找到当前正在展示的controller（一般都会找到），兜底逻辑是返回主页面，让主页面接收消息展示记账页面

## 4. ORC识别

代码块

```

1  {
2      // 创建一个文字识别请求 (Vision 框架的 OCR 能力)，并定义识别完成后的回调逻辑
3      // Vision 框架异步处理图片或视频帧时，通过 Completion Handler 回传结果。这个请求会
      被添加到 VNImageRequestHandler 后统一调度执行
4      VNRecognizeTextRequest *textRequest = [[VNRecognizeTextRequest alloc]
initWithCompletionHandler:^(VNRequest * _Nonnull request, NSError * _Nullable
error) {
5          if (error) {
6              completion(nil, error);
7              return;
8          }
9
10         // request.results 里存放着所有被识别出来的“文字区域对象”
11         NSArray<VNRecognizedTextObservation *> *observations = request.results;
12         NSMutableArray<NSDictionary *> *recognizedTextsWithBoxes =
[NSMutableArray array];
13         // 遍历每个识别出的文本区域（可能是一行、一列或某个字符块）
14         for (VNRecognizedTextObservation *observation in observations) {
15             // Vision 内部对每个文本块会返回多个识别结果（如置信度排序），通常只取第一
            个（最高可信）
16             VNRecognizedText *topCandidate = [observation
topCandidates:1].firstObject;
17             if (topCandidate) {

```

```

18         CGRect boundingBox = observation.boundingBox;
19         // 把识别到的文字和位置封装成 NSDictionary, 加入到结果数组中
20         [recognizedTextsWithBoxes addObject:@{@"text":
topCandidate.string, @"box": [NSValue valueWithCGRect:boundingBox]}];
21     }
22 }
23 // 根据文字的位置从上到下排序
24 [recognizedTextsWithBoxes
sortUsingComparator:^(NSComparisonResult(NSDictionary *obj1, NSDictionary
*obj2) {
25     CGRect box1 = [obj1[@"box"] CGRectValue];
26     CGRect box2 = [obj2[@"box"] CGRectValue];
27     if (box1.origin.y < box2.origin.y) {
28         return NSOrderedAscending;
29     } else if (box1.origin.y > box2.origin.y) {
30         return NSOrderedDescending;
31     } else {
32         return NSOrderedSame;
33     }
34 }]];
35
36 // 按行排序
37 }
38 }

```

- ORC的使用基本上是通过GPT编写的, 注解解释了它的作用
- 其中ORC识别按行展示是属于业务逻辑, 在这里主要为了优先处理微信+支付宝的截屏记账

## 4.1 匹配时间

代码块

```

1  {
2      NSMutableArray<NSString *> *dates = [NSMutableArray array];
3      NSMutableArray *timeIntervals = [NSMutableArray array];
4
5      NSString *datePattern = @"(\\d{4}[-/年]\\d{1,2}[-/月]\\d{1,2}[日]?
\\d{0,2}:?:\\d{0,2}:?:\\d{0,2})|((\\d{1,2}[-/年]\\d{1,2}[-/月]\\d{4}))";
6
7      NSError *regexError = nil;
8
9      NSRegularExpression *dateRegex = [NSRegularExpression
regularExpressionWithPattern:datePattern options:0 error:&regexError];
10     if (regexError) {
11         completion(nil, regexError);
12         return;

```



```

13     }
14
15     NSArray<NSString *> *dateFormats = @[
16         @"yyyy年MM月dd日HH:mm:ss", // 新增无空格格式
17         @"yyyy年MM月dd日 HH:mm:ss",
18         @"yyyy-MM-dd HH:mm:ss",
19         @"yyyy/MM/dd HH:mm:ss",
20         @"yyyy年MM月dd日",
21         @"yyyy-MM-dd",
22         @"yyyy/MM/dd"
23     ];
24
25     for (NSString *text in combinedRecognizedTexts) {
26         NSArray<NSTextCheckingResult *> *dateMatches = [dateRegex
matchesInString:text options:0 range:NSMakeRange(0, text.length)];
27         for (NSTextCheckingResult *match in dateMatches) {
28             NSString *dateString = [text substringWithRange:match.range];
29             [dates addObject:dateString];
30
31             // Convert date string to TimeInterval
32             NSDate *date = [self dateFromString:dateString
withFormats:dateFormats];
33             if (date) {
34                 NSTimeInterval timeInterval = [date timeIntervalSince1970];
35                 NSString *timeIntervalStr = [NSString
stringWithFormat:@"%f", timeInterval];
36                 [timeIntervals addObject:timeIntervalStr];
37             }
38         }
39     }
40 }

```

- 正则表达式：四位整数，/和年可选；月份和日同理，:可能有具体时间，时分秒，也兼容143059等无冒号写法；或者有两位数表示的年月日，其实是支持“日/月/年”格式，如“02-08-2023”这种倒序日期
- 在扫描的文字中尝试匹配，把匹配到的可能的文字使用dateFormats中可能的形式转换成date，如果转换成功，加入数组
- 优化：这个数组可以学习YYModel做个缓存，使用static NSArray \*cache + dispatch\_once的方式只创建一次
  - 不过这里这样写其实对性能影响不大，因为自动记账不是一个大批量调用性能敏感的操作，单次调用创建数组的速度还是非常快的
  - 这条优化写出来只是表达我有认真读过源码😁

- 时间排序：这个属于产品逻辑，代码里就没有写
  - 主要是对时间做排序，最小的时间戳最有可能是下单时间，其它的有可能是送达时间、发货时间这些不需要记账的日期

## 4.2 匹配名称



名称和备注分离是我们最近半年才做的事，其实下文备注就是名称，有点习惯了备注这个称呼，但是实际上在寻梦记账里这两个东西是非常不一样的概念，下面文章里如果出现备注都是「名称」的意思

备注匹配其实很简单，并没有用到正则表达式，这里基本上就是产品逻辑，没什么可分享的技术，在这里简单分享一下调研能力和产品思考，这里也涉及到产品和成本的权衡

截屏自动记账的备注命中率非常低，主要其实就是暴力枚举，把市面上主要的支付截图列出来，然后找规律，没有其它技巧

市面上的app只要不用AI，都无法做到高命中率的备注，即使使用AI也无法做到备注的精准识别的，因为每个人对一笔账的备注定义都不同

- 对AI记账软件模式的推测：如果要精准命中名称+备注，可能需要上传用户的之前类似账单的名称+备注，让AI去推测最优解，这样上传prompt必然带来token数量的激增，同时增加成本，增加响应时间

我们的做法很简单，针对常用App做出匹配优化，主要是对微信、支付宝做了优化

- 微信支付宝有商品、商户、收款方全称、商户全称
- 或者金额的上行一行往往是备注

然后对一些不可能是备注的词作了block

代码块

```
1  NSArray<NSString *> *excludeKeywords = @[@"满减", @"减", @"余额", @"零钱", @"零钱  
   余额", @"优惠", @"券"];  
2  NSArray<NSString *> *priorityKeywords = @[@"合计", @"实付款", @"到手", @"付款",  
   @"总计", @"实付", @"订单金额", @"支付"];  
3  NSArray<NSString *> *currencySymbols = @[@"¥", @"€", @"£", @"$"];
```

像包含这一类的文字的行，一般都不是备注

所以名称的匹配是一件非常难的事情，想象一个淘宝页面，里面即使只购买了两件商品，也可能有无数种记录的可能

- 有可能要记录两件商品为备注
- 有可能两件商品是名称

- 有可能这两件商品根本不是一种东西，需要分开记成不同的账
- 有可能用户根本不在乎这是什么东西，只记录一个大概即可，也就是不记名称，也不记备注，或者用户有自己对这个产品的定义，比如（分类：学习，名称：书）或者其它什么的

所以「寻梦记账」把名称虚化，把记账主要的事情还是交给用户

## 4.3 匹配金额

匹配金额的主要做法就是，匹配带+/-/\$/¥的数字，排除一些可能是优惠的文字，本质还是暴力枚举，根据产品逻辑找到最优解

- 优先匹配带合计、实付款、到手、付款这些文字为前缀的数字
- 再匹配带金额符号或者+/-号的纯数字
- 每次匹配的时候都要过滤掉block的文字行

代码块

```
1  NSArray<NSString *> *excludeKeywords = @[@"满减", @"减", @"余额", @"零钱", @"零钱
    余额", @"优惠", @"券"];
2  NSArray<NSString *> *priorityKeywords = @[@"合计", @"实付款", @"到手", @"付款",
    @"总计", @"实付", @"订单金额", @"支付"];
3  NSArray<NSString *> *currencySymbols = @[@"¥", @"€", @"£", @"$"];
```

代码逻辑

代码块

```
1      for (NSString *text in strings) {
2          BOOL shouldExclude = NO;
3          BOOL hasPriority = NO;
4          for (NSString *exclude in excludeKeywords) {
5              if ([text containsString:exclude]) {
6                  for (NSString *priority in priorityKeywords) {
7                      if ([text containsString:priority]) {
8                          hasPriority = YES;
9                          break;
10                     }
11                 }
12                 shouldExclude = YES;
13                 break;
14             }
15         }
16         if (shouldExclude && !hasPriority) {
17             continue;
18         }
19     }
```

```

20         for (NSString *keyword in priorityKeywords) {
21             NSRange keywordRange = [text rangeOfString:keyword];
22             if (keywordRange.location != NSNotFound) {
23                 NSUInteger startIndex = keywordRange.location +
keywordRange.length;
24                 if (startIndex >= text.length) {
25                     continue;
26                 }
27                 NSString *substring = [text substringFromIndex:startIndex];
28                 NSString *foundAmount = [self
extractAmountFromString:substring withCurrencySymbols:currencySymbols];
29                 if (foundAmount) {
30                     NSNumber *number = [self
numberFromAmountString:foundAmount];
31                     if (number) {
32                         if (shouldExclude) {
33                             [priorityAmounts addObject:number];
34                         } else {
35                             [purePriorityAmounts addObject:number];
36                         }
37                     }
38                 }
39             }
40         }
41     }

```

- 先过滤带有block词语的文字行，这里要注意的是，有的文字是这样的：实付款 18元 已优惠1元，这样的情况需要判断文字里是不是同样包含优先处理的文字，如果没有，才block
- 如果有优先匹配关键字的行，就做处理，尝试把文字转换成金额

尝试解析金额

代码块

```

1  + (NSString *)extractAmountFromString:(NSString *)string withCurrencySymbols:
(NSArray<NSString *> *)currencySymbols {
2      if ([XMAmountCheckManager string:string
containsAnyCurrencySymbol:currencySymbols]) {
3          return nil
4      }
5      // 其它业务逻辑
6  }
7
8  @implementation XMAmountCheckManager
9

```

```

10 + (BOOL)string:(NSString *)string containsAnyCurrencySymbol:(NSArray<NSString
    *> *)currencySymbols {
11     if (string.length == 0 || currencySymbols.count == 0) {
12         return NO;
13     }
14
15     // 0. 初步检查字符串中是否包含任何货币符号或 +、- 符号
16     // 构建一个包含所有需要检查的符号的集合
17     NSMutableString *allSymbolsToCheck = [NSMutableString string];
18     for (NSString *symbol in currencySymbols) {
19         // 转义正则表达式中的特殊字符
20         NSCharacterSet *regexSpecialChars = [NSCharacterSet
characterSetWithCharactersInString:@"$^*+?()[]{}|\\."];
21         if ([symbol rangeOfCharacterFromSet:regexSpecialChars].location !=
NSNotFound) {
22             [allSymbolsToCheck appendFormat:@"%\\%@", symbol];
23         } else {
24             [allSymbolsToCheck appendString:symbol];
25         }
26     }
27     // 添加 + 和 - 符号
28     [allSymbolsToCheck appendString:@"+-"];
29
30     // 使用正则表达式检查是否包含至少一个指定的符号
31     NSString *checkPattern = [NSString stringWithFormat:@"% %@",
allSymbolsToCheck];
32     NSRegularExpression *checkRegex = [NSRegularExpression
regularExpressionWithPattern:checkPattern options:0 error:nil];
33     NSUInteger numberOfMatches = [checkRegex numberOfMatchesInString:string
options:0 range:NSMakeRange(0, string.length)];
34
35     if (numberOfMatches == 0) {
36         // 如果没有找到任何货币符号或 +、- 符号, 直接返回 nil
37         return NO;
38     }
39     return YES;
40 }
41
42 @end

```

- 把需要匹配的金钱字符和+-号写进正则表达式里，也就是这里是在拼接正则表达式，拼接后的正则表达式类似[\\\$¥+1]
- 尝试匹配金额，返回结果

```

1  + (NSString *)extractAmountFromString:(NSString *)string withCurrencySymbols:
   (NSArray<NSString *> *)currencySymbols {
2      // 之前的逻辑
3      NSMutableCharacterSet *allowedChars = [[NSMutableCharacterSet alloc] init];
4      [allowedChars addCharactersInString:@"+-0123456789."];
5
6      for (NSString *symbol in currencySymbols) {
7          [allowedChars addCharactersInString:symbol];
8      }
9
10     // 2. 移除所有不在允许字符集中的字符
11     NSString *cleanedString = [[string componentsSeparatedByCharactersInSet:
   [allowedChars invertedSet]] componentsJoinedByString:@""];
12
13     if (cleanedString.length == 0) {
14         return nil;
15     }
16     NSString *firstCharacter = nil;
17     if (cleanedString.length > 0) {
18         firstCharacter = [cleanedString substringToIndex:1];
19     }
20
21     // 创建一个包含所有符号的数组，包括货币符号和+、-
22     NSMutableArray<NSString *> *allSymbols = [currencySymbols mutableCopy];
23     [allSymbols addObjectsFromArray:@"+", @"-"];
24
25     if ([allSymbols containsObject:firstCharacter]) {
26         // 符号位于开头，提取从第一个字符开始的子字符串
27         NSString *substringFromSymbol = [cleanedString substringFromIndex:0];
28
29         // 验证符号后面是否有有效的数字
30         NSCharacterSet *numberSet = [NSCharacterSet
   characterSetWithCharactersInString:@"0123456789."];
31         // 从索引1开始查找非数字字符
32         NSRange firstNonNumber = [substringFromSymbol rangeOfCharacterFromSet:
   [numberSet invertedSet] options:NSLiteralSearch range:NSMakeRange(1,
   substringFromSymbol.length - 1)];
33         if (firstNonNumber.location != NSNotFound) {
34             // 有无效字符，截断
35             substringFromSymbol = [substringFromSymbol
   substringToIndex:firstNonNumber.location];
36         }
37
38         cleanedString = substringFromSymbol;
39         NSLog(@"[Amount Extraction] Cleaned String after removing prefix: %@",
   cleanedString); // 添加日志
40     } else {

```

```

41         // 符号不在开头，不认为是有效的金额
42         NSLog(@"[Amount Extraction] 符号不在字符串开头，返回 nil");
43         return nil;
44     }
45     // 其它业务逻辑
46 }
47

```

1. 提取文字中运行出现的符号和数字，删除多余的文字
2. 符号开头，认为是一个可能的金额数字，验证符号后的文字是不是一个有效的数字
  - 先把金额过滤成只有符号数字小数点，这样做主要是为了简化后续的正则表达式

正则表达式匹配金额，防止一些无关数字被匹配

代码块

```

1  {
2      // 4. 定义正则表达式模式
3      // 确保字符串以货币符号或+/-符号开头，并限制数字部分最多为8位
4      NSString *pattern = [NSString stringWithFormat:@"^(?:[+-]|[%@])?\\d{1,8}
5      (?:\\.\\d{1,2})?$", escapedSymbols];
6
7      NSError *error = nil;
8      NSRegularExpression *regex = [NSRegularExpression
9      regularExpressionWithPattern:pattern options:0 error:&error];
10     if (error) {
11         NSLog(@"正则表达式创建失败: %@", error.localizedDescription);
12         return nil;
13     }
14
15     // 5. 执行正则匹配
16     NSRange searchRange = NSMakeRange(0, cleanedString.length);
17     NSTextCheckingResult *match = [regex firstMatchInString:cleanedString
18     options:0 range:searchRange];
19     if (match) {
20         NSRange matchRange = [match range];
21         NSString *matchedAmount = [cleanedString
22         substringWithRange:matchRange];
23         NSLog(@"[Amount Extraction] Matched Amount: %@", matchedAmount); // 添加
24         日志
25         return matchedAmount;
26     }
27 }

```

- 匹配开头是可选的+/-或金钱符号，整数位1-8位数，可选的小数点后两位，\$确保整个字符串完全匹配金额的格式

以上就是优先匹配的文字匹配金额的方式，而之前提到的匹配通用文字的方式就是取掉block文字的行，然后调用这个方法尝试把文字转换成金额，就不再赘述了

## 5. 优化

优化图片大小，减少CPU的消耗，优化ORC识别的性能，防止图片过大导致ORC处理的时候OOM，在AutoRecordIntent这个target优化图片

代码块

```
1  @implementation XMImageUtils
2
3  + (NSData *)compressImageData:(NSData *)originData maxPixel:(CGFloat)maxPixel {
4      UIImage *image = [UIImage imageWithData:originData];
5      if (!image) return nil;
6      CGSize originSize = image.size;
7      CGFloat scale = MIN(1.0, maxPixel / MAX(originSize.width,
originSize.height));
8      if (scale < 1.0) {
9          CGSize targetSize = CGSizeMake(originSize.width * scale,
originSize.height * scale);
10         UIGraphicsBeginImageContextWithOptions(targetSize, NO, 1.0);
11         [image drawInRect:CGRectMake(0, 0, targetSize.width,
targetSize.height)];
12         UIImage *scaled = UIGraphicsGetImageFromCurrentImageContext();
13         UIGraphicsEndImageContext();
14         image = scaled ?: image;
15     }
16     return UIImagePNGRepresentation(image);
17 }
18
19 @end
```

- 传入一个业务需要的最大像素，比如1080
- 如果最大边大于最大像素就缩放，不然不缩放
- 缩放
  - 计算缩放后目标尺寸。开启位图上下文，准备绘制缩放。按目标尺寸绘制原图，实现等比缩放。
  - 从当前图形上下文获取缩放后的 UIImage。关闭上下文释放内存。如果缩放成功，使用缩放后的图片，否则保留原图。



- 图片返回PNG格式，防止失真导致ORC识别不了
  - 其实这里可以通过实际测试控制maxPixel，然后使用JPEG，这样ORC识别的会更快