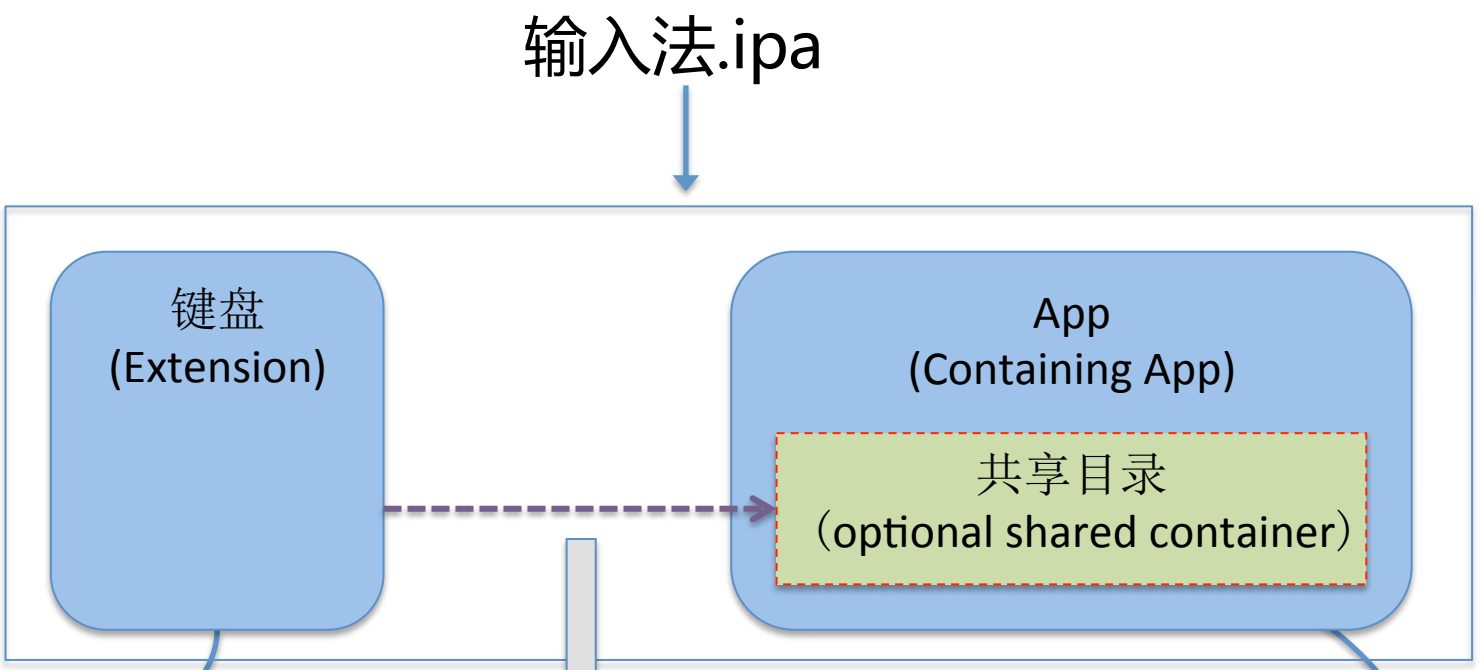
MDCC 2016
中国移动开发者大会
Mobile Developer Conference China 2016

# 搜狗输入法性能优化实践

李腾杰@Sogou.com

mdcc.csdn.net

- 输入法架构概述
- 面临的问题
- 关键性能指标
- 优化实践
- 未来展望





| 允许完全访问 | 对共享目录的读写权限 | |
|---|---|---|
| | 键盘 | App |
| 打开 | 可读+可写 | 可读+可写 |
| 关闭 | iOS 8.3以下：不可读+不可写<br>iOS 8.3及以上：可读+不可写 | 可读+可写 |

- 自定义键盘框架的稳定性
  - 调不起键盘
  - 键盘被切成系统输入法
  - 键盘不见了
  - 键盘调起不流畅等
- 允许完全访问控制
  - 产品交互
  - 技术实现
- 内存限制
  - 低内存

输入法不好用！

调不起键盘！
键盘被切成系统输入法！
键盘不见了！
(null)-搜狗输入法!
调起不流畅!

不能换肤！
没有按键音！
不能导入通讯录词库！
不能使用表情！

莫名其妙崩溃！

iOS系统自定义键盘框架的稳定性
产品自身的性能及稳定性

允许完全访问的影响

低内存
产品自身稳定性

# 好用的输入法？

打字准　　打字快　　功能稳定　　功能丰富

输入法内核！
首选率
退格率
首屏率
…

流畅+快！
按键响应时间
键盘调起速度

不崩溃！
崩溃率
内存
CPU/耗电量

皮肤
表情输入
手写输入
语音输入
…

- 在线性能监测
  - 监测策略
  - 存储策略
  - 上传策略
  - 分析策略
- 自动化评测
  - 越狱设备
  - 借助Hook某些系统API+第三方越狱插件、开源库，实现评测需要的功能

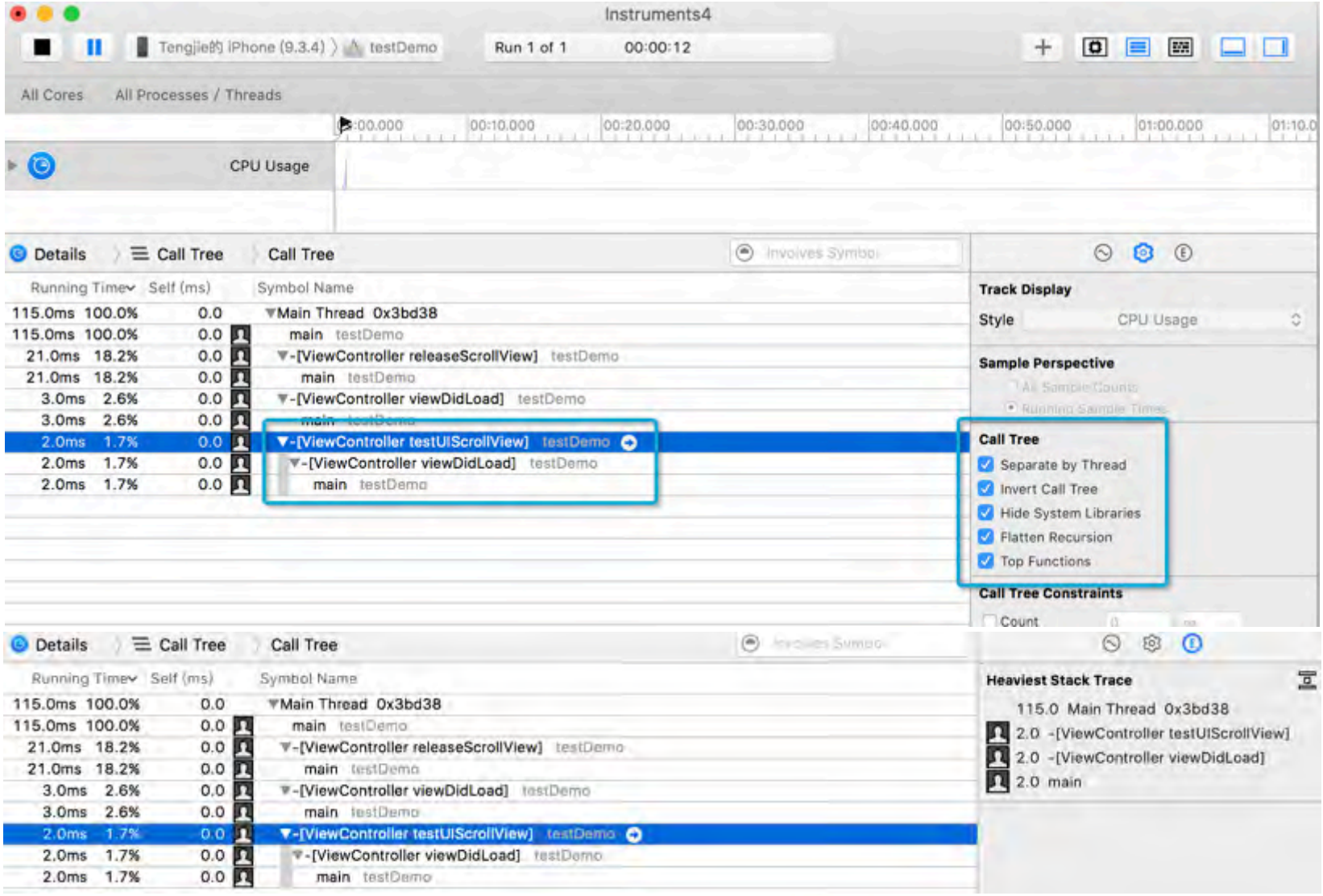
- openssh
  - ssh是一种可以保证用户远程登录到系统的协议
  - openssh是ssh的开源实现
- IPA Installer Console
  - 命令行插件，通过命令行的方式安装、卸载应用
- open
  - 命令行插件，用于启动应用
- SimulateTouch
  - 开源项目，模拟点击、滑动事件
- SimulateKey Events
  - 模拟发送物理按键事件的插件

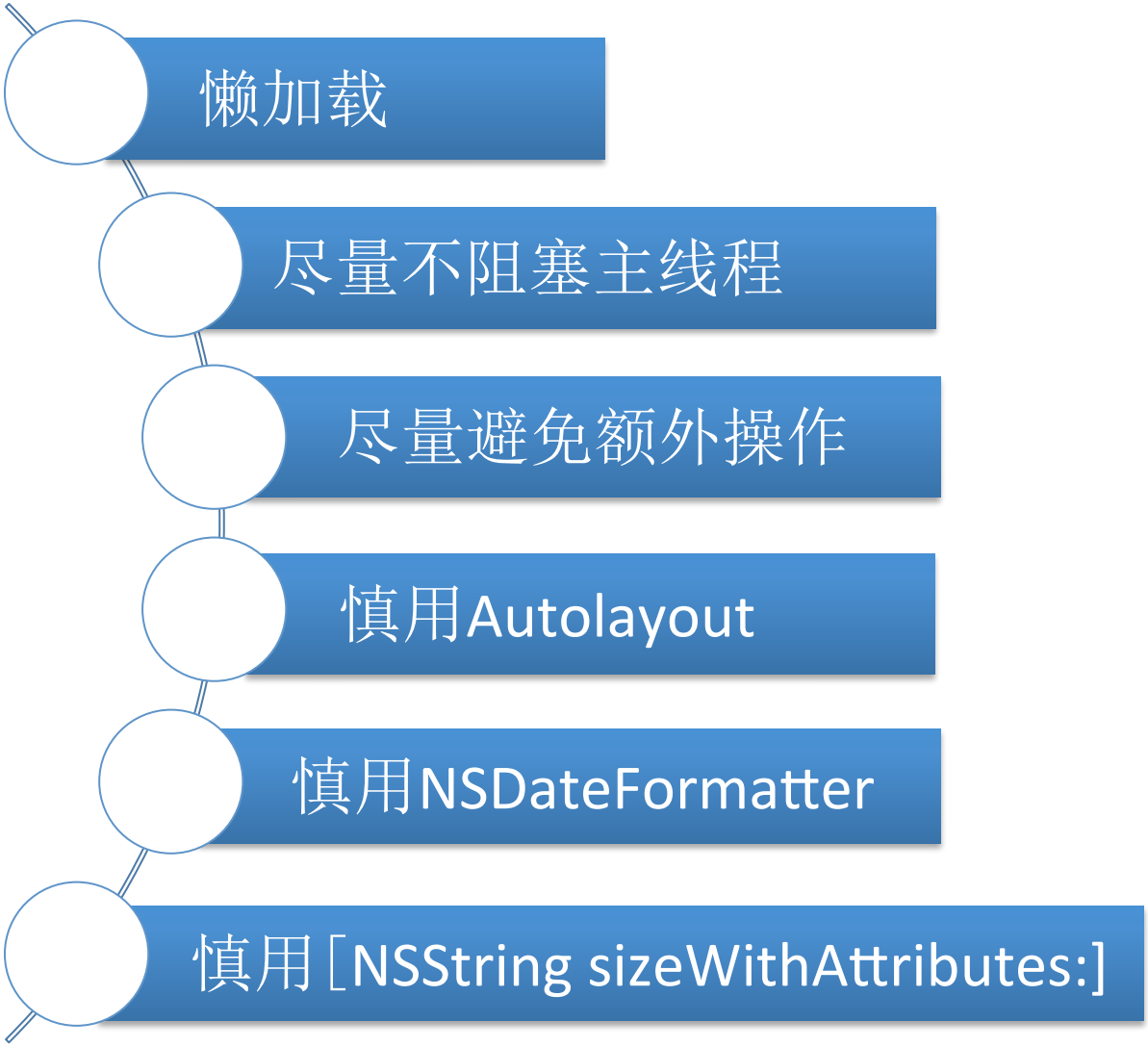
- 键盘调起速度
- 崩溃率
- 内存占用


键盘调起卡顿
键盘无法调起
影响打字速度
键盘调起速度
优化！！
如何评测？
如何定位？
如何优化？

- 评测方法
  - 对键盘调起的过程进行录像
  - 将录像过程解析出对应帧
  - 对解析的结果进行分析，统计调起过程对应的帧数
- 注意事项
  - 测试环境的一致性
  - 视频录制和解析环境的一致性
  - 测试误差的考虑

- Instruments -> Time Profiler




懒加载
尽量不阻塞主线程
尽量避免额外操作
慎用Autolayout
慎用NSDateFormatter
慎用[NSString sizeWithAttributes:]

- 懒加载
  - 延迟加载
  - 不需要的对象不立即初始化，需要用到时才初始化
  - 重写对象的getter方法
- 尽量不阻塞主线程
  - 同步 vs 异步
  - 延迟调用！！
- 尽量避免额外操作

```
@interface ViewController ()
@property (nonatomic, strong) SGICandidate *candidateView;
@end

@implementation ViewController

- (void)viewDidLoad {
    [super viewDidLoad];

    _candidateView = [[SGICandidate alloc] initWithFrame:CGRectZero];
    _candidateView.backgroundColor = [UIColor clearColor];
    [self.view addSubview:_candidateView];
}

@end
```

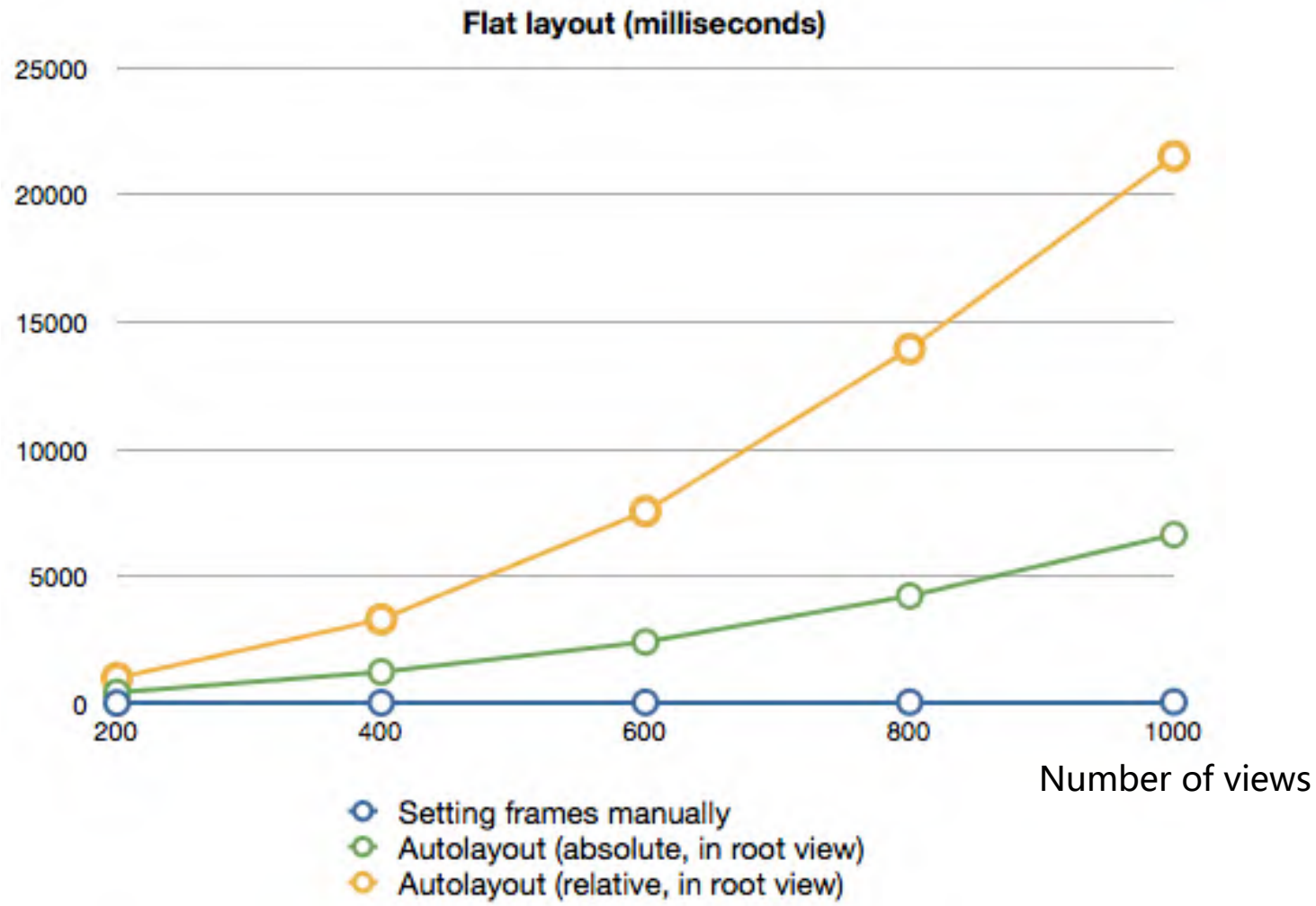
```
@implementation ViewController

- (void)viewDidLoad {
    [super viewDidLoad];
}

- (SGICandidate *)candidateView {
    if (!_candidateView) {
        _candidateView = [[SGICandidate alloc] initWithFrame:CGRectZero];
        _candidateView.backgroundColor = [UIColor clearColor];
        [self.view addSubview:_candidateView];
    }
    return _candidateView;
}

@end
```

- Autolayout
  - 简单、强大易用、可读性强
  - updateConstraints调用时机
  - 性能问题
- 原则：技术调研要到位，分析和实践相结合


- 参考链接：http://pilky.me/36/

```
- (void)testNSDateFormatter {
    NSString *dateString;
    //time0
    NSDateFormatter *newDateForMatter = [[NSDateFormatter alloc] init];
    [newDateForMatter setDateFormat:@"yyyy-MM-dd"];
    dateString = [newDateForMatter stringFromDate:[NSDate date]];
    //time1
    //NSLog(@"%@ using NSDateFormatter costs %f milliseconds\n", dateString, (time1 - time0) * 1000);

    //time2
    time_t timeInterval = [NSDate date].timeIntervalSince1970;
    struct tm *cTime = localtime(&timeInterval);
    dateString = [NSString stringWithFormat:@"%d-%02d-%02d", cTime->tm_year + 1900,
                  cTime->tm_mon + 1, cTime->tm_mday];
    //time3
    //NSLog(@"%@ using localtime costs %f milliseconds\n", dateString, (time3 - time2) * 1000);
}
```

- 输出结果：
  - using NSDateFormatter costs 1.349986 milliseconds
  - using localtime costs 0.090957 milliseconds
- 应减少对NSDateFormatter的调用，或用localtime替代

```
- (void)testNSStringSizeWithAttributes {
    //time0
    NSString *text = @"字符串";
    UIFont *font = [UIFont systemFontOfSize:20];
    CGFloat textWidth1 = [text sizeWithAttributes:@{NSFontAttributeName:font}].width;
    //time1
    //NSLog(@"Text width %0.2f costs %f milliseconds\n", textWidth1, (time1 - time0) * 1000);

    //time2
    CGFloat oneCharacterWidth = 20.0f;
    CGFloat textWidth2 = oneCharacterWidth * text.length;
    //time3
    //NSLog(@"Text width %0.2f costs %f milliseconds\n", textWidth2, (time3 - time2) * 1000);
}
```

- 输出结果：
  - Text width 61.14 costs 17.042994 milliseconds
  - Text width 60.00 costs 0.000000 milliseconds
- 减少字符串size的实时计算，或寻求替代方案

- 崩溃日志收集
  - Crash Reporter
  - itunes Connect
  - Bugly
  - Crashlytics
  - 友盟等
- 崩溃日志解析
  - symbolicatcrash
  - xcrun atos
  - dwarfdump
- 按崩溃分布定优先级解决问题


输入法使用过程中闪退
锁屏后被切成其它输入法
键盘无法调起
低内存崩溃！！
为何低内存崩溃？
如何评测内存占用？
如何跟踪低内存情况？
如何优化内存占用？

- 低内存警告的回调方法
  - 普通应用

  UIApplicationDelegate

  - (void)applicationDidReceiveMemoryWarning:(UIApplication*)application
  - 键盘扩展

  KeyboardViewController

  - (void)didReceiveMemoryWarning
- 所有的应用都应该实现这个方法，以通过释放能够被重新创建或加载的缓存数据或对象来释放尽量多的内存。

## Optimize Efficiency and Performance

App extensions should feel nimble and lightweight to users. Design your app extension to launch quickly, aiming for well under one second. An extension that launches too slowly is terminated by the system.

Memory limits for running app extensions are significantly lower than the memory limits imposed on a foreground app. On both platforms, the system may aggressively terminate extensions because users want to return to their main goal in the host app. Some extensions may have lower memory limits than others: For example, widgets must be especially efficient because users are likely to have several widgets open at the same time.

Your app extension doesn't own the main run loop, so it's crucial that you follow the established rules for good behavior in main run loops. For example, if your extension blocks the main run loop, it can create a bad user experience in another extension or app.

Keep in mind that the GPU is a shared resource in the system. App extensions do not get top priority for shared resources; for example, a Today widget that runs a graphics-intensive game might give users a bad experience. The system is likely to terminate such an extension because of memory pressure. Functionality that makes heavy use of system resources is appropriate for an app, not an app extension.

- Resident Memory
- Used Memory

# Resident Memory

```
struct task_basic_info {
    integer_t       suspend_count;  /* suspend count for task */
    vm_size_t       virtual_size;   /* virtual memory size (bytes) */
    vm_size_t       resident_size;  /* resident memory size (bytes) */
    time_value_t    user_time;      /* total user run time for
                                       terminated threads */
    time_value_t    system_time;    /* total system run time for
                                       terminated threads */
    policy_t    policy;     /* default policy for new threads */
};
typedef struct task_basic_info          task_basic_info_data_t;

+ (float)residentSizeOfMemory {
    task_basic_info_data_t taskInfo;
    mach_msg_type_number_t infoCount = TASK_BASIC_INFO_COUNT;
    kern_return_t kernReturn = task_info(mach_task_self(), TASK_BASIC_INFO,
                                         (task_info_t)&taskInfo, &infoCount);

    if (kernReturn != KERN_SUCCESS) {
        return 0.0f;
    }
    return (taskInfo.resident_size / (1024.0 * 1024.0));
}
```

# Used Memory

```
+ (float)usedSizeOfMemory {
    task_vm_info_data_t taskInfo;
    mach_msg_type_number_t infoCount = TASK_VM_INFO_COUNT;
    kern_return_t kernReturn = task_info(mach_task_self(),
                                         TASK_VM_INFO_PURGEABLE,
                                         (task_info_t)&taskInfo, &infoCount);

    if (kernReturn != KERN_SUCCESS) {
        return 0.0f;
    }
    return ((taskInfo.internal + taskInfo.compressed - taskInfo.
        purgeable_volatile_pmap) / (1024.0 * 1024.0));
}
```

```
#define TASK_VM_INFO            22
#define TASK_VM_INFO_PURGEABLE  23
struct task_vm_info {
        mach_vm_size_t  virtual_size;       /* virtual memory size (bytes) */
    integer_t   region_count;       /* number of memory regions */
    integer_t   page_size;
        mach_vm_size_t  resident_size;      /* resident memory size (bytes) */
        mach_vm_size_t  resident_size_peak; /* peak resident size (bytes) */

    mach_vm_size_t  device;
    mach_vm_size_t  device_peak;
    mach_vm_size_t  internal;
    mach_vm_size_t  internal_peak;
    mach_vm_size_t  external;
    mach_vm_size_t  external_peak;
    mach_vm_size_t  reusable;
    mach_vm_size_t  reusable_peak;
    mach_vm_size_t  purgeable_volatile_pmap;
    mach_vm_size_t  purgeable_volatile_resident;
    mach_vm_size_t  purgeable_volatile_virtual;
    mach_vm_size_t  compressed;
    mach_vm_size_t  compressed_peak;
    mach_vm_size_t  compressed_lifetime;

    /* added for rev1 */
    mach_vm_size_t  phys_footprint;
};
```

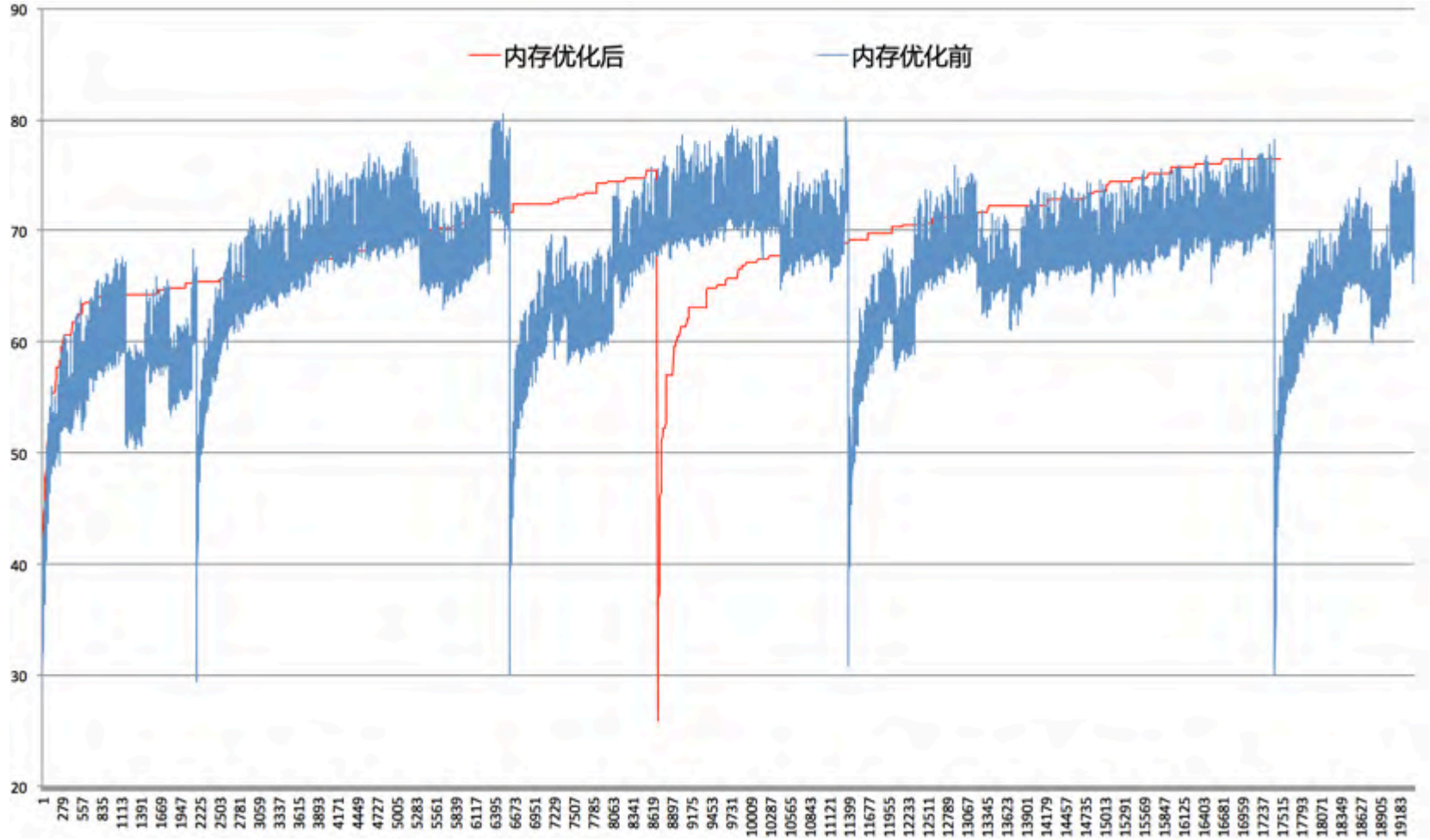
- 难点
  - 现有第三方崩溃日志收集方案无法捕获低内存崩溃
  - 低内存阈值未知
- 越狱方案
  - 遍历设备特定目录下的崩溃日志，可以大概估算出低内存情况
- 非越狱方案
  - 低内存 --> 键盘崩溃 --> 键盘重新初始化
  - 逆向推理：键盘重新初始化的可能原因？
  - 崩溃、设备重启、低内存、覆盖安装升级

- CACurrentMediaTime()
  - 基于系统内建时钟，不会以为外部时间变化而变化
  - 与系统的uptime有关，系统重启后

    CACurrentMediaTime()会被重置
- CFAbsoluteTimeGetCurrent()
  - [[NSDate data] timeIntervalSinceReferenceDate]
  - 返回的时钟时间，与网络时间同步

- 实测数据：
- rebootTime=nCurrentTime - mediaTime


- iPhone 6P，模拟用户操作评测



- autoreleasepool的使用
- 避免循环引用
- 读图方式优化
- 选择正确的缓存策略
- 降低内存占用峰值
- 内存文件映射的使用
- FastImageCache的使用

# autoreleasepool的使用

```objc
- (void)testWithoutAutoreleasepool {
    NSString *bDataString = @"test:";
    for (int i = 1; i <= 2000; i++) {
        NSString *keyName = [NSString stringWithFormat:@"a%d", i];
        bDataString = [bDataString stringByAppendingFormat:@"%@=%d", keyName, i];
        if (i < 2000)
            bDataString = [bDataString stringByAppendingString:@"&"];

        //memory used: 21372 -> 60004 = 38632(KB)
        if (1 == i || 2000 == i)
            [self NSTraceMemoryInfo];
    }
}

- (void)testWithAutoreleasepool {
    NSString *bDataString = @"test:";
    for (int i = 1; i <= 2000; i++) {
        @autoreleasepool {
            NSString *keyName = [NSString stringWithFormat:@"b%d", i];
            bDataString = [bDataString stringByAppendingFormat:@"%@=%d", keyName, i];
            if (i < 2000)
                bDataString = [bDataString stringByAppendingString:@"&"];
        }

        //memory used: 60004 -> 60124 = 120(KB)
        if (1 == i || 2000 == i)
            [self NSTraceMemoryInfo];
    }
}
```

- 第三方内存泄露检测工具
  - FBRetainCycleDetector
  - FBAllocationTracker
  - FBMemoryProfiler
- 防止使用block时出现引用闭环
  - Reactive Cocoa中用到的一种宏：weakify、strongify
  - 强弱引用转换，用于解决block与强引用self之间的循环引用问题

```
@weakify(self); // 定义了一个__weak的self_weak_变量
[RACObserve(self, name) subscribeNext:^(NSString *name) {
    @strongify(self); // 局域定义了一个__strong的self指针指向self_weak
    self.outputLabel.text = name;
}];
```

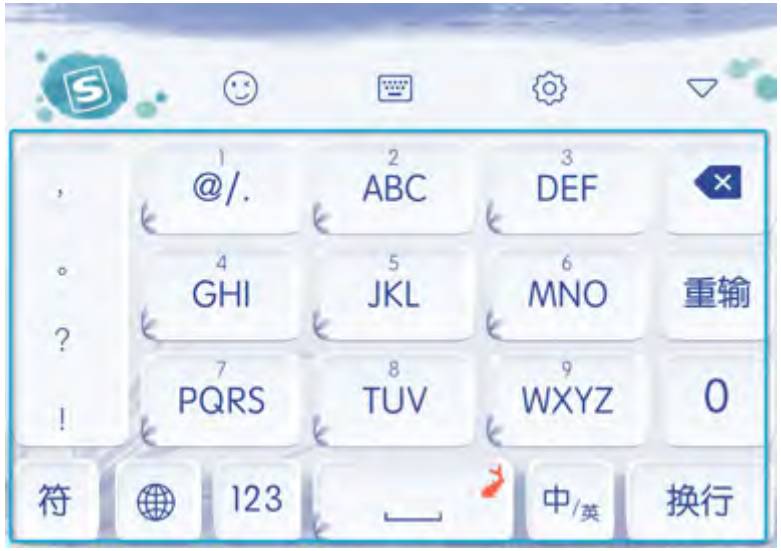
- [UIImage imageWithContentsOfFile:@""]
  - 优点
    - 不缓存图片到内存，内存可及时释放
    - 适用于大图片，使用完就释放
  - 缺点
    - 无法读取Asset Catalog里的图片
    - 读取图片需要完整文件名
- [UIImage imageNamed:@""]
  - 优点
    - 图片始终缓存，适用于某些需要在多个地方显示的图标, 其对应的UIImage对象只会被创建一次，避免频繁的沙盒读写
  - 缺点
    - 缓存图片到内存，不能及时释放

- 基本原则
  - 缓存需要的
  - 缓存常用的
  - 缓存计算或生成开销较大的
- 优化实践
  - 非常用功能退出后，立即释放相关内存占用
  - 有选择缓存图片资源、数据资源

# 输入法缓存举例


夜间模式
键盘高度
单手键盘
候选字号
繁体
云输入
英文补全
滑行输入
小字输入
更多设置

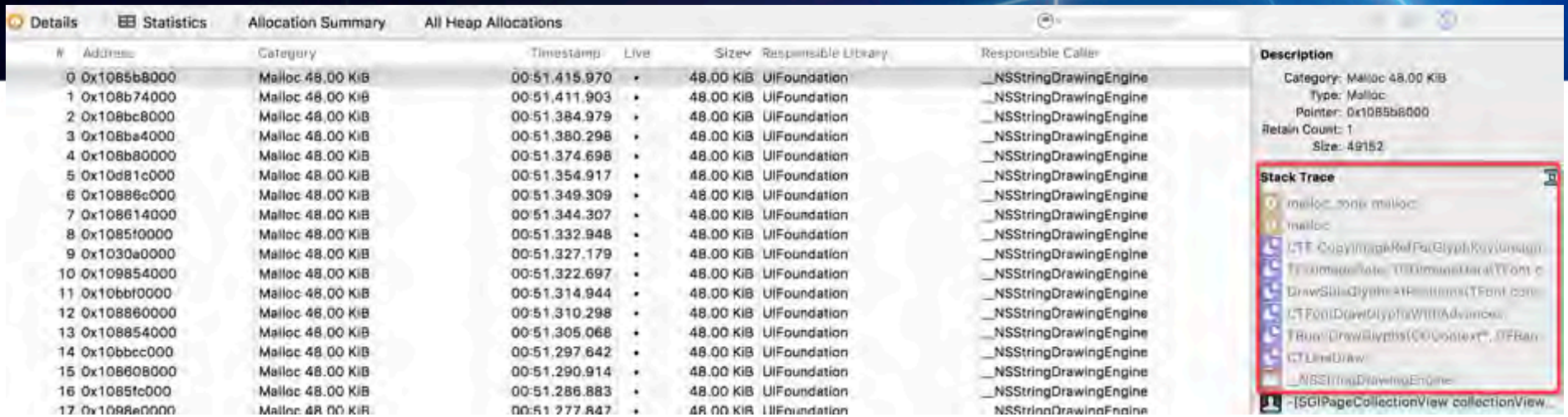

Emoji
颜文字
图片表情
我有这么快
时而生猛
洪荒之力
别说话用心...
加油
怪我咯
历史最好成绩
这么快
胜利
你太弱
返回
最近

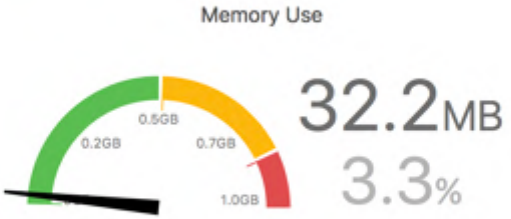

@/.
ABC
DEF
GHI
JKL
MNO
重输
PQRS
TUV
WXYZ
0
符
123
中/英
换行

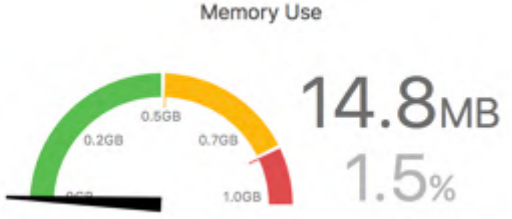
- 目标
  - 始终保持应用内存占用处于一个相对平稳的范围内
- 优化实践
  - Emoji、颜文字表情内存优化
  - 通讯录规模较大时，人名信息联想内存优化

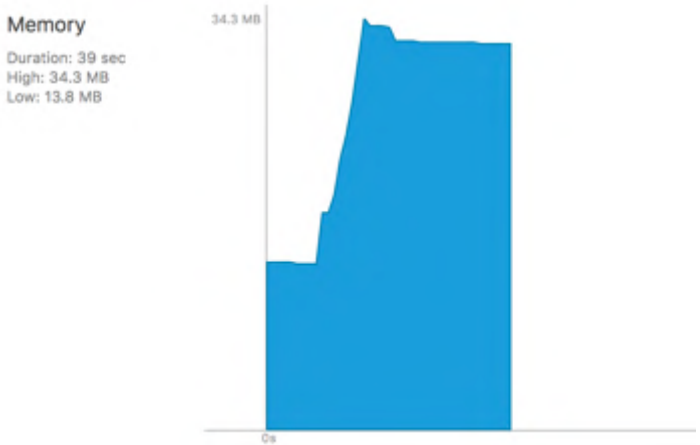
- 面临问题
  - 直接绘制方式占用内存太大，峰值过高
  - 绘制缓存无法立即释放

- 解决方法
  - 牺牲包大小，用图片替代绘制
  - 控制图片大小
  - 优化图片存储、读取方式
  - 不同iOS系统emoji表情更新带来的影响！！



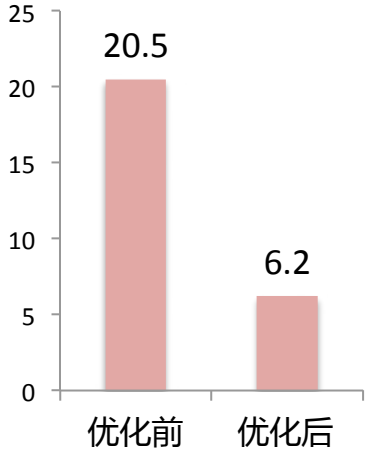



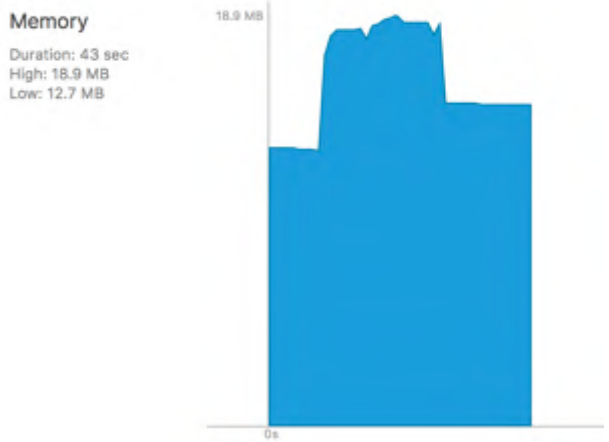




优化前

内存峰值增量

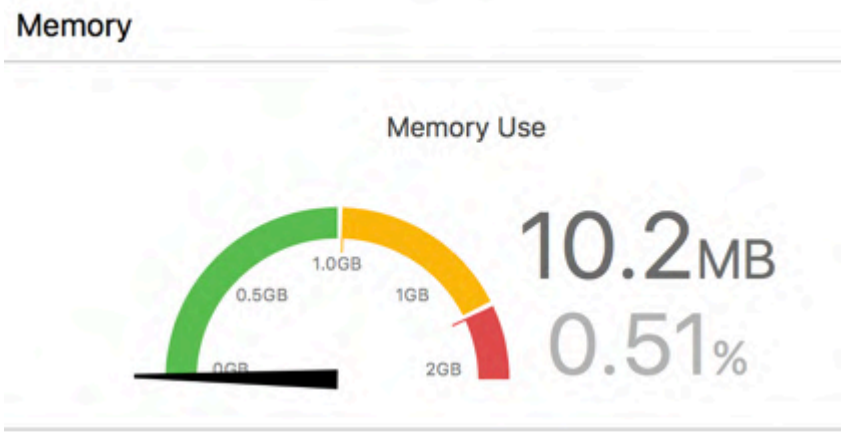


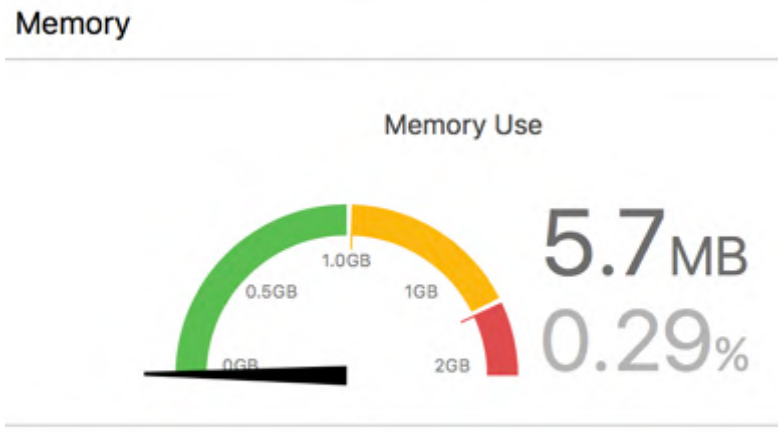



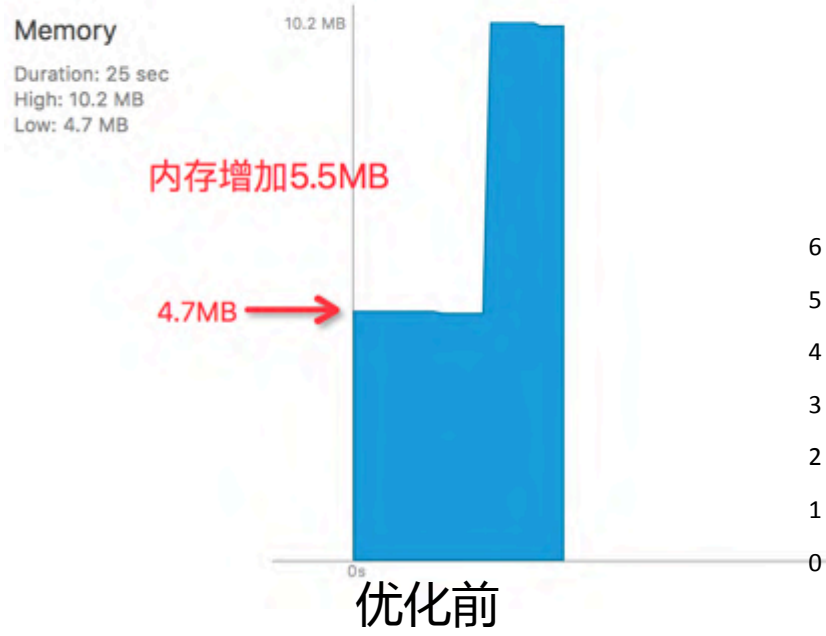

优化后

- 面临问题
  - 手机通讯录规模庞大，在多次进行通讯录信息联想时，存在较大的内存峰值，容易引发内存问题

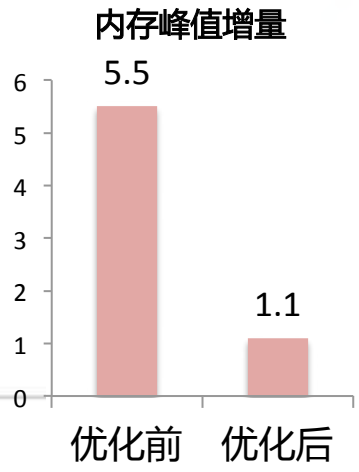
- 解决方法
  - CFArrayRef ABAddressBookCopyArrayOfAllPeople (ABAddressBookRef addressBook)
  - CFArrayRef ABAddressBookCopyPeopleWithName (ABAddressBookRef addressBook, CFStringRef name)

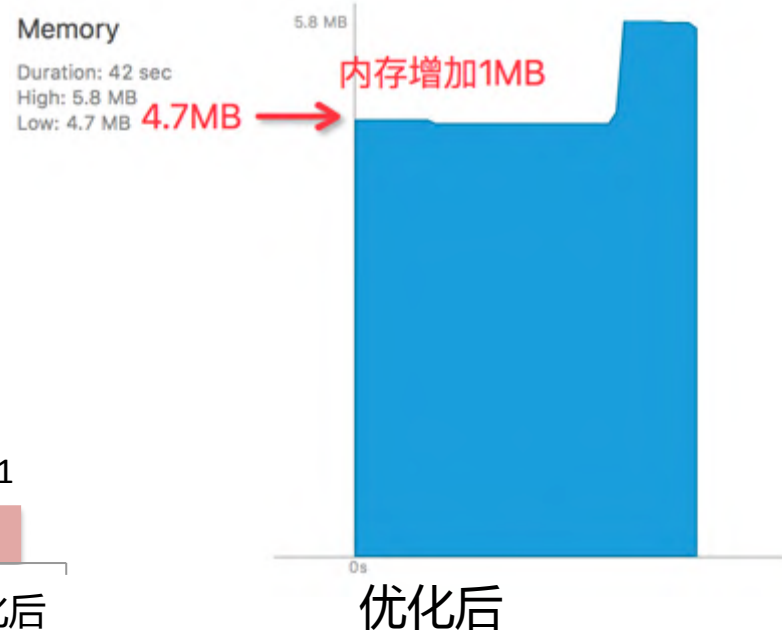

Memory
Memory Use
0.5GB
1.0GB
1GB
0GB
2GB
10.2MB
0.51%


Memory
Memory Use
0.5GB
1.0GB
1GB
0GB
2GB
5.7MB
0.29%


Memory
Duration: 25 sec
High: 10.2 MB
Low: 4.7 MB
10.2 MB
内存增加5.5MB
4.7MB
0s

优化前


内存峰值增量
5.5
1.1
6
5
4
3
2
1
0
优化前
优化后

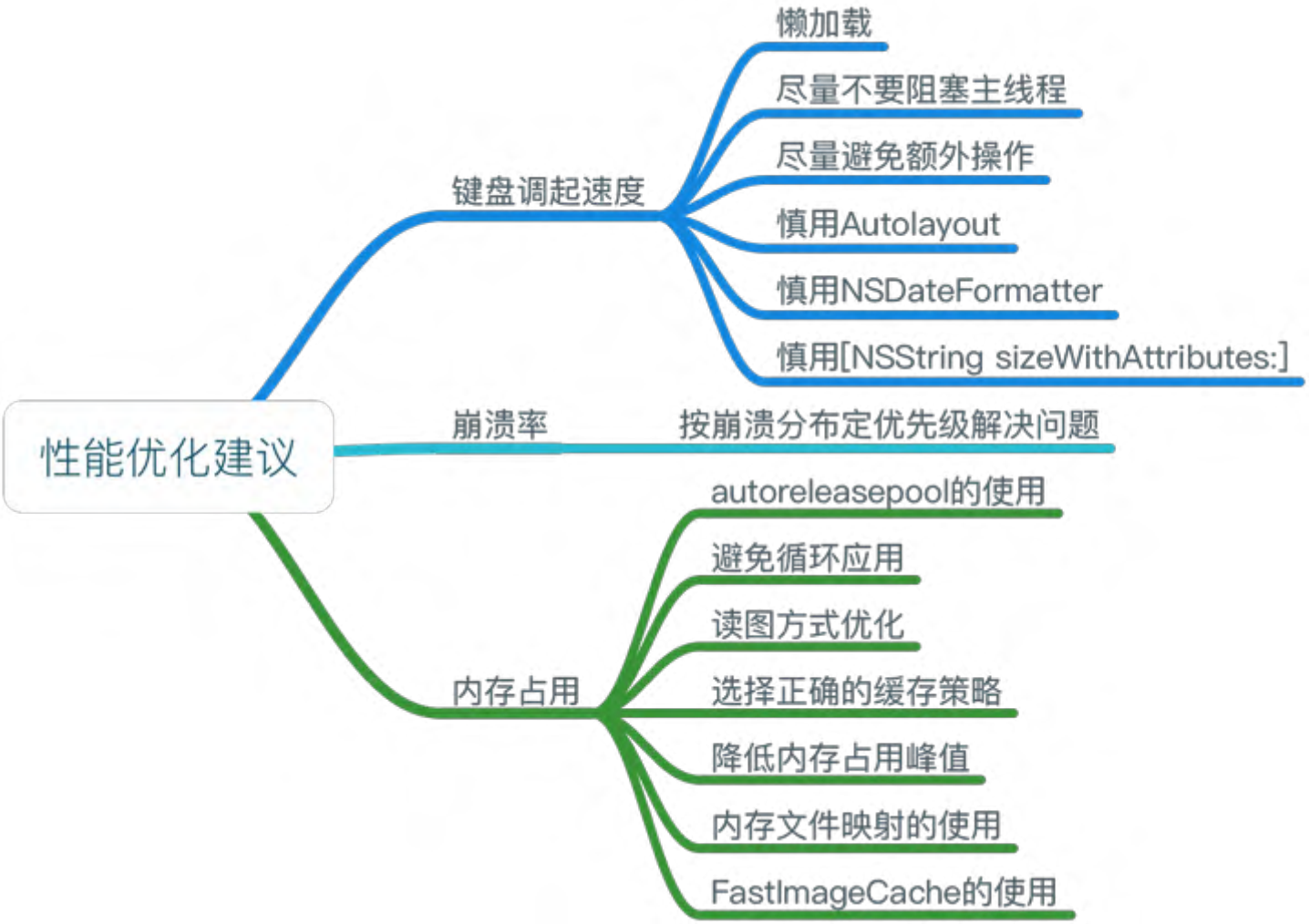

Memory
Duration: 42 sec
High: 5.8 MB
Low: 4.7 MB
5.8 MB
内存增加1MB
4.7MB
0s

优化后

- 优点
  - 内存文件映射对应external
  - 不算应用程序占用的内存，能有效减少应用程序占用的内存，避免触及低内存阈值
- 缺点
  - 性能
  - 映射文件的大小、个数需要考虑，两者权衡
- 使用文件映射
  - NSData: + dataWithContentsOfFile:options:error:;
  - mmap、munmap、msync

- FastImageCache
  - Path团队开发的一个开源库，用于提升图片的加载和渲染速度
  - Mapped memory
  - Uncompressed Image Data
  - Byte Alignment
  - https://github.com/path/FastImageCache
- 优化实践
  - 搜狗输入法：换肤优化、图片存储/读取方式优化
  - 有效优化换肤内存使用，评测结果显示减少了低内存概率

# 性能优化建议


性能优化建议
键盘调起速度
懒加载
尽量不要阻塞主线程
尽量避免额外操作
慎用Autolayout
慎用NSDateFormatter
慎用[NSString sizeWithAttributes:]
崩溃率
按崩溃分布定优先级解决问题
内存占用
autoreleasepool的使用
避免循环应用
读图方式优化
选择正确的缓存策略
降低内存占用峰值
内存文件映射的使用
FastImageCache的使用

- 性能指标监控体系的进一步完善
- 键盘调起速度的进一步优化
- 内存占用的进一步优化
- 架构扩展性优化，支持未来各种新功能的开发

# 谢谢！