Содержание

Предисловие

Эта книга призвана помочь вам отточить навыки разработки для iOS в узкой области: оптимизации производительности. Она предназначена для тех, кто уже обладает базовыми навыками в разработке для платформы iOS и желает создавать лучшие приложения.

Одержимый вопросами производительности приложений, я потратил массу времени на изучение и применение различных приемов оптимизации приложений в различных платформах, таких как веб и смартфоны. Мне нравится дискутировать на эту тему с людьми. Проводя много времени на форумах и в сообществах iOS, таких как Stack Overflow и Apple Developer Forum, я понял, что большинство разработчиков iOS задают одни и те же вопросы о том, как улучшить производительность своих приложений. Я подумал, что было бы полезно собрать наиболее общие из этих вопросов в хорошо написанную и структурированную книгу, чтобы с легкостью можно было получить полное представление о проблемах оптимизации проиводительности приложений для платформы iOS. Такова была моя мотивация для написания этой книги, я сделал все возможное, чтобы охватить самые распространенные проблемы и ошибки, встречающиеся разработчикам.

Кроме того, я встречал и отмечал в своих заметках множество проблем, общих для платформ iOS, Android и Windows Phone. Последняя глава основана на этих заметках. Я думаю, что она действительно будет полезна тем, кто желает работать на этих трех платформах или перейти с одной платформы на другую.

При работе с острыми проблемами производительности полезно рассмотреть их с разных сторон и выработать компромисс между производительностью и сложностью реализации выбранного решения. Бывают такие незначительные проблемы, которые заставляют совершать множество ошибок, если заранее не знать возможных способов их решения. Я надеюсь, что эта книга поможет вам избежать таких ошибок и вместо этого потратить свое время на улучшение производительности и создание у пользователей лучших впечатлений о приложении.

Глава 1. Введение в оптимизацию производительности iOS

В данной главе приводится общая информация о книге, включая следующее:

- кому эта книга послужит лучше всего;
- вопросы, которые охватывает книга;
- общая структура и стиль книги.

Новая эра смартфонов

В настоящее время рынок iOS насчитывает сотни тысяч приложений и сотни миллионов пользователей. Все это делает его достаточно обширным как для компании, так и для отдельного разработчика. Этот рынок рос в течение многих лет и будет еще расти, наряду со спросом на интересные и мощные приложения. Имея хорошую задумку для нового приложения, необходимо продумать ее исполнение, включая создание хорошего пользовательского интерфейса. Исходя из уникальных технических характеристик *смартфонов*, высокая производительность – необходимое условие для любого приложения. Пользователи хотят видеть приложение, которое быстро откликается на их действия, мгновенно считывает данные и отображает их.

Почему важна производительность

Оптимизация производительности заключается не только в разработке оптимальных алгоритмов, структур данных и приемов эффективного управления памятью, но и в том, чтобы создать у человека ощущение, что приложение отвечает на любое его действие настолько быстро, насколько это возможно. Таким образом, оптимизация производительности приложений для iPhone играет важную роль. Пользователям необходимо чувство, что их команды воспринимаются и исполняются почти мгновенно. Что, если при нажатии кнопки

реакции пришлось бы ожидать в течение двух долгих секунд? Кого порадует такая производительность? Скорее всего, большинство пользователей были бы весьма расстроены.

Конечно, большую часть данных и их обработку можно было бы переместить в сеть с тысячами серверов, способных быстро обрабатывать и возвращать результат. Однако этого явно недостаточно. *Передача данных через сеть* – сложный процесс, и пользователям, скорее всего, придется несколько секунд ждать получения данных.

Затруднения в улучшении производительности своих продуктов испытывают и разработчики основных приложений, и разработчики игр.

Для кого предназначена эта книга

В основном данная книга написана для разработчиков начального и среднего уровней, которые уже владеют основами программирования для iPhone. Она для тех, кто стремится обеспечить высокую производительность и желает создавать инновационные, быстро реагирующие и готовые к распространению приложения. Даже опытные разработчики смогут найти что-то полезное для себя в этой книге.

Если вы намерены глубоко погрузиться в мир программирования приложений для смартфонов, эта книга даст достаточный объем знаний, чтобы можно было применить все, что вы знаете об iOS, к *средам Android и Windows Phone*.

Стиль автора

Я искренне верю, что принцип обучения на практике для программиста является лучшим способом приобрести необходимые навыки. Книга базируется именно на этом принципе. Я рассматриваю общие и частные вопросы с высоты моего двухгодичного опыта разработки для iPhone и более длительного периода разработки и преподавания *Java*. Проблемы, которые я поставлю перед вами, помогут избежать или исправить множество ошибок, связанных с производительностью, при разработке программ для iPhone. Я выбрал эти вопросы, исходя из опыта и исследования тем, наиболее популярных на форумах и в социальных сетях (таких как *Stack Overflow*). Я выделил общие подводные камни и дам вам советы, которые помогут их избежать.

Книга представляет собой совокупность основополагающих принципов, пояснительных иллюстраций и примеров исходного кода. Вместо предоставления конкретного инструмента для решения какой-либо проблемы я хочу предложить вам прочные навыки для каждодневного применения. Я использую разные подходы к изложению материала: иногда иллюстрация ценнее тысячи слов, некоторые идеи проще объяснить примером программного кода, а в некоторых случаях требуется та самая тысяча слов.

Один из лучших способов начать обучение – это разработать крутое приложение, которое бы вам очень нравилось. Такой практический опыт даст больше, чем некоторые теоретические и быстро забываемые примеры.

Совсем необязательно много знать о *фреймворке Cocoa Touch*, потому что я объясню основной синтаксис и классы, необходимые для улучшения производительности приложений. Каждая глава состоит из отдельной темы, при этом некоторые темы могут быть вам уже знакомы. Эту книгу можно также использовать как общий справочник. Как только у вас возникнет какая-то конкретная проблема, вы сможете отыскать описание и прочесть о ее решении.

Главы имеют простой формат: краткий обзор главы, за которым следуют разделы и подразделы. Каждая глава заканчивается выводами, помогающими обобщить полученные знания и напоминающими о самых важных вопросах. Затем следуют несколько простых практических упражнений, чтобы вы могли с удовольствием попрактиковать то, о чем только что узнали.

Что вам потребуется

Как разработчику для платформы iOS вам потребуется *Mac OS с установленным пакетом Xcode*. Бесплатная версия Xcode предоставляется зарегистрированным разработчикам для iOS, или ее можно загрузить непосредственно с сайта Apple Mac AppStore. Вам также понадобится экземпляр этой книги со всеми примерами программного кода, которые можно загрузить с сайта издательства «Apress». Примеры проектов были протестированы в среде разработки Xcode 4.2 с включенным режимом *автоматического подсчета ссылок* (Automatic Reference Counting – ARC), поэтому они будут запускаться в этой среде без всяких проблем.

Вы можете и должны опробовать каждый пример, чтобы лучше понять рассматриваемый вопрос. В книге также встречаются корот-

кие блоки программного кода, не привязанные ни к какому проекту, их тоже следует проработать.

Как пользоваться этой книгой

Хотя главы не совсем близки по смыслу, для более полного понимания проблем производительности и приемов оптимизации лучше читать от начала до конца. Главы могут в чем-то перекликаться между собой. Более поздние главы были написаны с учетом того, что вы уже прочли или знакомы с информацией из предыдущих глав.

Также рекомендуется читать каждую главу полностью, от начала до конца. Главы открываются коротким концептуальным введением в тему. Далее теория совмещается с практическими примерами, чтобы помочь овладеть темой полностью.

Вам следует внимательно читать заключительные разделы глав, поскольку они еще раз напоминают о ключевых вопросах, которые необходимо запомнить. Я также рекомендую выполнить все упражнения, поскольку это позволит закрепить полученные знания.

Обзор книги

Эта книга представляет собой хорошее сочетание базовых концепций и практических знаний, приемов и рекомендаций, которые помогут добиться успеха в конкурентном мире разработки приложений для iOS. Девять глав этой книги охватывают девять различных подходов к решению проблем производительности, возникающих при разработке приложений для iOS.

- *Глава 2*. Знакомит с рядом средств и инструментов, со способами их применения. Многие разработчики не используют данные средства просто потому, что не знают об их существовании.
- *Глава 3*. Как разработчику для iOS вам определенно придется пользоваться как простыми, так и сложными табличными представлениями (TableView) для отображения списков данных или опций. Проблема архитектуры `UITableView` заключается в том, что как только вы начинаете ее настраивать, страдает эффективность прокрутки. Эта проблема обязательно возникнет, пусть и в слабой мере. В данной главе будут представлены несколько способов повысить эффективность прокрутки представлений TableView.

- *Глава 4.* Может показаться, что большинство проблем с производительностью решаются при помощи облачных технологий и нескольких дополнительных серверов в системе. Даже если это и так, передача данных по сети всегда будет оставаться проблемой. Передача данных будет оставаться узким местом еще долгие годы. Вы должны знать, как кешировать данные в локальной памяти, в такой ограниченной среде, как в iOS.
- *Глава 5.* Структуры данных и алгоритмы, используемые в приложениях для iOS, сходны и в то же время отличны от структур данных и алгоритмов, применяемых в других средах. Фреймворк обеспечивает высокий уровень поддержки множества базовых структур данных, таких как массивы, множества и словари. В некоторых случаях можно просто поместить их в сеть, тогда как в других, особенно когда сбор и обработка данных выполняются для визуализации, все равно придется опираться на среду iOS.
- *Глава 6.* Под улучшением производительности также понимается уменьшение времени отклика приложения на действия пользователя. Это означает, что основной поток выполнения пользовательского интерфейса не должен блокироваться. Многопоточность может помочь увеличить не только скорость реакции, но и производительность приложения в целом. Многопоточность – сложная тема для любой платформы. Здесь она раскрывается рядом иллюстраций, примеров и доступными объяснениями.
- *Глава 7.* С появлением нового инструмента автоматического управления памятью разработчики теперь могут пользоваться его преимуществами, чтобы избежать таких типичных проблем с памятью, как утечка или нехватка памяти. Данная глава рассказывает, как лучше использовать память и когда следует загружать данные в память и выгружать их. Здесь также рассказывается о механизме автоматического подсчета ссылок (ARC), включенном в новый *набор средств разработки* (Software Development Kit – SDK), чтобы убедиться, что вы правильно понимаете и используете его.
- *Глава 8.* В версиях iOS 4 и выше все приложения могут пользоваться преимуществом многозадачности для создания благоприятных впечатлений у пользователей. В сущности, это даже не совсем многозадачность, а скорее механизм быстрого переключения между приложениями с некоторой специаль-

ной фоновой обработкой (приложения не могут выполняться в фоновом режиме). Данная глава поможет понять, какие компоненты будут поддерживаться iOS и какие задачи можно выполнять в фоновом режиме.

- *Глава 9*. Во многих приложениях для iPhone нет необходимости использовать программный код на C/C++ для реализации каких-то возможностей. Однако серьезной проблемой становятся ситуации, когда это действительно необходимо, особенно при интеграции с внешними библиотеками. Возможно, и не понадобится писать все приложение на C/C++, однако необходимо понимать, как эти языки работают, чтобы не допустить ошибок.
- *Глава 10*. На этом этапе ваше представление обо всем разнообразии аспектов производительности iPhone уже будет полностью сформировано. Вы наверняка задумаете в скором времени перенести свое приложение на устройства с Android или Windows Phone. Поэтому последняя глава посвящена сходству проблем производительности в iOS, Android и Windows Phone, что поможет сгладить практические нюансы при освоении новых платформ.

Загружаемые примеры

Примеры с исходным программным кодом можно загрузить на странице этой книги, находящейся на веб-сайте издательства «Apress» (www.apress.com), и опробовать их самостоятельно.

Обратная связь

Если у вас возникнут какие-либо вопросы, пожалуйста, напишите мне по электронной почте vodkhang@gmail.com или посетите мой веб-сайт http://vodkhang.com. Я буду рад обсудить проблемы производительности на платформе iPhone.

Глава 2. Оценка производительности с использованием эмулятора и действующего устройства

Данная глава посвящена следующим вопросам:

- различия между эмулятором и действующим устройством;
- влияние управления памятью на производительность приложения;
- инструменты и способы оценки производительности приложения, включая следующее:
  - базовые инструменты измерения производительности и расходования памяти;
  - улучшенные инструменты исследования различных аспектов управления памятью, таких как утечки или неоптимальное использование памяти;
  - улучшенные инструменты исследования различных аспектов, влияющих на производительность, таких как аккумулятор, необходимость загрузки файлов и отображение информации;
  - разделение программы на более мелкие части с целью упростить выявление узких мест в производительности.

Для улучшения производительности необходимо провести оценочное тестирование, чтобы увидеть, в чем заключается проблема. Чтобы провести тестирование с пользой, необходимо понимать причины, ведущие к замедленной работе программы или сегмента кода.

На начальном этапе придется принять два фундаментальных решения: что выбрать – эмулятор или действующее устройство и как сохранить баланс между эффективным использованием памяти и высокой производительностью программного кода.

Для начала необходимо понять разницу между эмулятором и действующим устройством.

Эмулятор и действующее устройство

Главная причина низкой производительности приложений для iPhone состоит в том, что они выполняются в среде с ограниченными ресурсами и невысокой скоростью обработки. *Эмулятор среды iPhone* работает гораздо быстрее настоящего устройства. В сущности, скорость работы *эмулятора* зависит от скорости работы компьютера, на котором он выполняется. В результате очень большим и неприятным сюрпризом может явиться тот факт, что программа действительно быстро выполняется в *эмуляторе* и гораздо медленнее – на *действующем устройстве*. Я встречал множество людей, которые винили сеть в плохой производительности приложения. Безусловно, это иногда верно. Однако в большинстве случаев производительность приложения может сильно страдать именно из-за особенностей реализации кода, а не из-за сетевых проблем. Поэтому тщательное тестирование и оценка производительности приложения основными инструментами в стандартных средах позволят получить полное представление о скорости работы приложения и его пользовательского интерфейса.

Чтобы продемонстрировать значительную разницу между эмулятором и действующим устройством, я протестировал программу в эмуляторе iPhone и на действующем устройстве iPhone. Результаты впечатляют:

- в эмуляторе iPhone вычисления занимают 0,5 сек;
- на действующем устройстве iPhone те же вычисления занимают 7 сек.

Программа была несложная: я сделал простой тест с двумя массивами по тысяче элементов в каждом. Программа проходит в цикле по обоим массивам, чтобы найти одно и то же число, и выводит слово «привет». В действительности вам может и не понадобиться обработка тысячи элементов в массиве, или вы можете не использовать обработку массивов для поиска одинаковых чисел. Однако суть не в этом. Я выбрал именно эти действия, чтобы продемонстрировать, что действующее устройство iPhone намного медленнее эмулятора iPhone.

Отсюда следует вывод, к которому я буду еще не раз возвращаться в этой книге: приложение необходимо тестировать как в эмуляторе, так и на действующем устройстве. Так почему же не тестировать только на действующем устройстве? Потому, что эмулятор обладает следующими значительными преимуществами:

- тестирование в эмуляторе занимает меньше времени, что означает для разработчиков меньше временных потерь;
- эмулятор отлично подходит для выявления утечек памяти и проблем ее распределения.

Память и производительность

Память и производительность представляют собой разные вещи. Под памятью обычно понимается *оперативная память* (Random Access Memory – RAM) и, соответственно, сколько места вы использовали и сколько осталось свободным. Производительность обозначает, насколько быстро приложение выполняет определенную операцию.

Память может оказывать большое влияние на производительность. Когда устройство обладает большим объемом оперативной памяти, можно предварительно загрузить и кешировать в ней больше данных. Оперативная память обладает более высокой скоростью доступа, по сравнению с файловым и сетевым хранилищами. Предварительно загрузив и кешировав больше данных в оперативной памяти, во многих случаях можно значительно увеличить скорость программ. Например, для игрового приложения, требующего загрузки большого количества изображений, большой объем памяти является необходимым условием, потому что позволяет заранее загрузить картинки и отображать их по мере необходимости. Загрузка из оперативной памяти в 10 раз быстрее, чем загрузка из файловой системы.

Однако более эффективное использование памяти не всегда означает лучшую производительность. Некоторые приложения не требуют много памяти, поэтому, оптимизируя использование памяти, можно столкнуться с тем, что производительность уже не повышается. Обратная ситуация не лучше: приложение может занять всю память, чтобы добиться лучшей производительности, а затем ему памяти просто не хватит.

Таким образом, необходимо всегда тщательно оценивать как память, так и скорость выполнения, чтобы убедиться в наличии строгого баланса между использованием памяти и производительностью во время выполнения.

Инструменты

Инструменты делятся на следующие три основные категории:

- основные, помимо инструментов среды разработки Xcode;

- инструменты оценки правильности и эффективности использования памяти;
- инструменты оценки производительности, позволяющие определить, насколько быстро выполняется каждая часть программы, и выявить слабые места.

Основные инструменты

В этой части в качестве основного метода измерения будет обсуждаться метод регистрации времени выполнения блоков кода.

Регистрация времени выполнения

Одним из самых простых методов является регистрация временной разницы между началом и окончанием выполнения блока кода. Обычно такая регистрация осуществляется посредством функции `NSLog`. Используя ее, разработчики могут выяснить, насколько быстро выполняется каждая строка или блок кода.

Например, при выполнении следующего фрагмента кода

```
NSDate *date1 = [NSDate date];
for (int i = 0; i < 1000; i++) {
  // Здесь выполняются вычисления
}
NSDate *date2 = [NSDate date];
NSLog (@"time: %f", [date2 timeIntervalSinceDate:date1]);
```

были получены такие результаты:

```
time: 0.0123 (измеряется в секундах)
```

Преимущества:

- простой и понятный способ измерения производительности;
- возможность проверки производительности отдельных строк и блоков кода.

Недостатки:

- отсутствие возможности оценить производительность пользовательского интерфейса (то есть время отображения, зависящее от действий пользователя);
- бессмысленная оптимизация (можно потратить много времени на какой-то определенный блок кода, увеличив скорость его работы совсем незначительно);

- в эмуляторе приложение обычно выполняется быстро. При такой скорости функция `NSLog` не поможет выявить разницу в производительности выполняющегося приложения. С другой стороны, так как `NSLog` на действующем устройстве работает медленно, она поможет выявить такую разницу.

Применение:

- когда необходимо быстро провести измерения без долгих приготовлений;
- когда необходимо быстро получить результат;
- когда необходимо изолировать небольшой блок кода, чтобы проверить предполагаемую производительность.

Инструменты оценки использования памяти

В том, что касается памяти, главной задачей является ее наиболее эффективное использование. Существуют также менее значительные проблемы с унаследованным кодом, такие как утечки и чрезмерное расходование памяти. В новых проектах предпочтение следует отдавать новому механизму *автоматического подсчета ссылок* (*ARC – Automatic Reference Counting*). Некоторые старые проекты можно попробовать преобразовать с помощью утилиты, входящей в комплект среды разработки Xcode.

Однако преобразованию поддается не каждый проект. Политика управления памятью и еще ряд проблем не позволят этого сделать. Попытки придерживаться новой политики управления могут создать еще большие проблемы. Поэтому здесь в основном рассматриваются инструменты, позволяющие выяснить распределение памяти для объектов, и более кратко инструменты, выявляющие утечки и чрезмерное расходование памяти.

> **Примечание**: все инструменты оценки использования памяти, которые представлены здесь (а здесь представлены все целевые инструменты Apple), могут использоваться в эмуляторе. Он хорош тем, что действительно быстр в исполнении и быстро загружает приложения. Однако я настоятельно рекомендую тестировать свои приложения также и на действующем устройстве, потому что эмулятор и действующее устройство не всегда одно и то же. Они имеют разные строение и архитектуру.

Инструмент Allocations (Распределение памяти)

Инструмент **Allocations** (Распределение памяти) помогает узнать, как много памяти выделяется для объектов. Он позволяет выяснить,

не слишком ли много объектов хранится в памяти и какие из них пока не могут быть освобождены, потому что продолжают использоваться.

Распределение памяти

Выберите пункт меню **Product** (Продукт) ⇒ **Profile** (Профилировать), а затем в открывшемся окне выберите ярлык **Allocations** (Распределение памяти) (как показано на рис. 2.1).

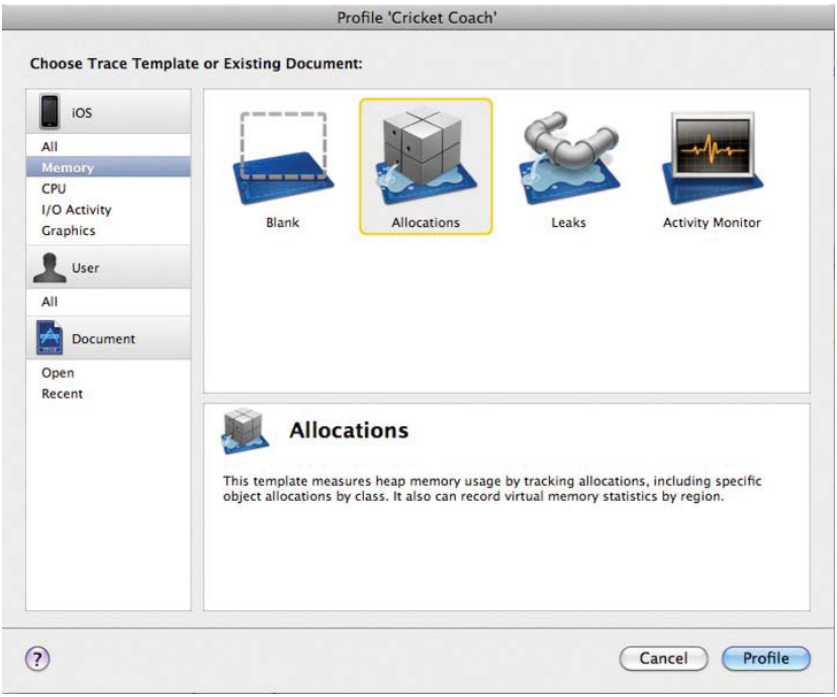

Рис. 2.1. Выбор ярлыка инструмента отображения распределения памяти

После выбора инструмента откроется его основная панель, на которой предоставлена вся необходимая информация, как показано на рис. 2.2.

Панель инструмента **Allocations** (Распределение памяти) (рис. 2.2) по умолчанию отображает объекты **Created & still living** (Созданные и существующие в настоящий момент), что позволяет

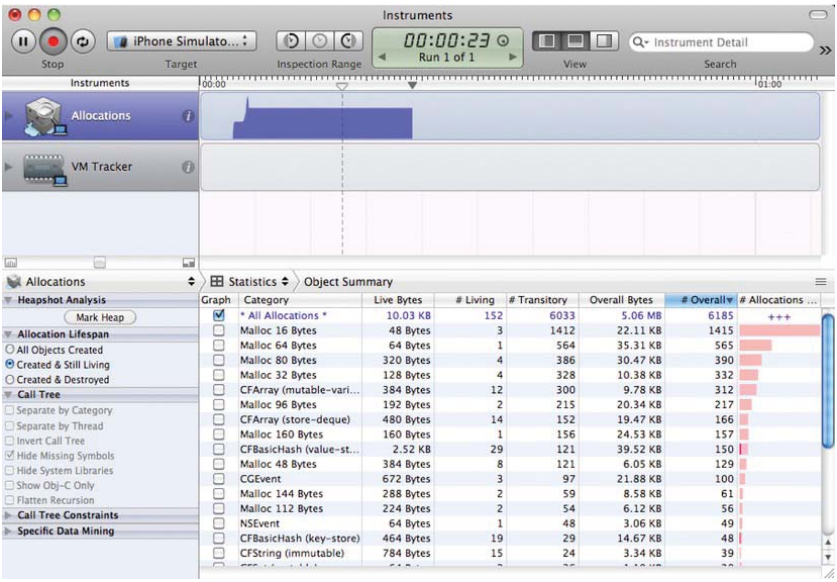

Рис. 2.2. Основная панель инструмента отображения распределения памяти

увидеть объекты, которые все еще находятся в памяти, и объекты, занимающие большую ее часть. Этот инструмент следует использовать, когда начинает появляться большое количество предупреждений от среды iOS, как, например, «Received memory warning. Level = 1» (Получено предупреждение о нехватке памяти. Уровень = 1).

Таблица с подробной информацией содержит сведения о том, какие строки кода и какие классы ответственны за создание и удержание объекта. Используя эту информацию, можно легко найти способ более эффективного использования памяти. Это хороший инструмент для исследования алгоритмов и методов кеширования (подробнее о кешировании рассказывается в главе 4).

Рисунки 2.3 и 2.4 более детально отображают, какие объекты продолжают существовать и потребляют большую часть памяти приложения. На рис. 2.3 изображена таблица с подробной информацией об объектах, созданных и продолжающих существовать внутри приложения.

На рис. 2.4 видно, какие методы вызывались для создания этих объектов.

Graph	Category	Live Bytes	# Living	# Transitory	Overall Bytes	# Overall	# Allocations (Net / Overall)
☑	* All Allocations *	27.77 KB	565	55386	17.06 MB	55951	
☐	CFRunLoopTimer	11.72 KB	125	250	35.16 KB	375	
☐	CFBasicHash (value-st...	4.50 KB	135	1146	109.23 KB	1281	
☐	CFSet (mutable)	3.91 KB	125	266	12.22 KB	391	
☐	CFBasicHash (count-s...	2.36 KB	8	57	16.86 KB	65	
☐	__NSCFDate	1.94 KB	124	247	5.80 KB	371	
☐	CGEvent	672 Bytes	3	291	64.31 KB	294	
☐	Malloc 144 Bytes	576 Bytes	4	924	130.50 KB	928	
☐	Malloc 48 Bytes	480 Bytes	10	1353	63.89 KB	1363	
☐	Malloc 80 Bytes	320 Bytes	4	4787	374.30 KB	4791	

Рис. 2.3. Подробная информация о распределении памяти

Category	Timestamp	Live	Size	Responsible Library	Responsible Caller
Malloc 80 Bytes	00:22.186.397	•	80	CoreGraphics	CGClipStackCreateMutable
Malloc 80 Bytes	00:25.680.592	•	80	CoreGraphics	CGClipStackCreateMutable
Malloc 80 Bytes	00:26.680.849	•	80	AppKit	-[NSViewHierarchyLock lock...
Malloc 80 Bytes	00:26.680.980	•	80	CoreGraphics	CGClipStackCreateMutable

Рис. 2.4. Детали распределения памяти

Преимущества:

- наличие точной и подробной информации о времени и условиях, в которых приложение потребляет больше всего памяти;
- позволяет получить достаточно полное представление о жизненном цикле объекта по отношению к жизненному циклу приложения.

Недостатки:

- результаты зависят от того, как выполняется приложение. Требуется тщательная подготовка комплексного набора тестов, чтобы охватить как можно больше вероятных ситуаций;
- возможны временные и трудовые затраты на создание набора тестов, который поможет разработчикам выявить место и время, когда приложение потребляет больше всего памяти;
- необходимо выполнять тестирование на действующем устройстве, чтобы можно было получать предупреждения о нехватке памяти. Эмулятор практически никогда не выдает таких предупреждений. Проблема заключается в том, что компьютер будет иметь оперативную память 2–4 Гб, тогда как действующее устройство – гораздо меньше.

Применение:

- это один из первых инструментов, к которым следует обратиться, если при работе с приложением появляются предупреждения о нехватке памяти.