Improving Your App with Instruments

Session 418
Daniel Delwood
Software Radiologist
Agenda

What's new in Instruments
Memory management
Time profiling
Performance counters
DTrace improvements
What's in Instruments
What's NEW in Instruments
Memory Management
Heap Memory

Everything Else
Objective-C's Ownership Model

Retain/Release

Reference counting ownership model based on retain, release
When the count drops to zero, object is freed
Retain/release/autorelease rules established and easy to learn

• *Advanced Memory Management Programming Guide*

Deterministic, simple, and fast
Objective-C's Ownership Model

**Managed Retain/Release**

Reference counting ownership model based on retain, release
When the count drops to zero, object is freed
Retain/release/autorelease rules established and easy to learn
- *Advanced Memory Management Programming Guide*

Deterministic, simple, and fast
Automated Reference Counting (ARC)
Objective-C's Ownership Model

Managed Retain/Release

Reference counting ownership model based on retain, release
When the count drops to zero, object is freed
Retain/release/autorelease rules established and easy to learn
  - *Advanced Memory Management Programming Guide*
Deterministic, simple, and fast
Automated Reference Counting (ARC)
  - Still have to manage autorelease pools

@autoreleasepool { /* code */ }
Swift's Ownership Model

Managed Retain/Release

Reference counting ownership model based on retain, release
When the count drops to zero, object is freed
Deterministic, simple, and fast
Automated Reference Counting (ARC)
Swift's Ownership Model

**Managed** Retain/Release

Reference counting ownership model based on retain, release

When the count drops to zero, object is freed

Deterministic, simple, and fast

Automated Reference Counting (ARC)

- Working with Objective-C? Still have to manage autorelease pools

```swift
autoreleasepool { /* code */ }```
Allocations

What does it report?

Heap allocations
- Class names — e.g. NSMutableArray, MyApp.MainViewController
- Reference types only (class, not struct)
- Retain/Release histories

Virtual Memory (VM) allocations
- Paths for mapped files

Stack traces for all
Demo
Allocations + App Extension
App Extensions
Profiling with Instruments

Specify host App
- When profiling Xcode scheme
- In Instruments

Transient, but memory matters
App Extensions

Profiling with Instruments

Specify host App

• When profiling Xcode scheme
• In Instruments

Transient, but memory matters

- Creating Extensions for iOS and OS X, Part 1  
  Mission  
  Tuesday 2:00PM

- Creating Extensions for iOS and OS X, Part 2  
  Mission  
  Wednesday 11:30AM
Memory Management with Swift

Language tools

Obj-C code can still mismatch Retain/Release
Can still form cycles in Swift
Memory Management with Swift

Language tools

Obj-C code can still mismatch Retain/Release
Can still form cycles in Swift
Manage graph, not retain/release

weak

unowned
Memory Management with Swift

Language tools

Obj-C code can still mismatch Retain/Release
Can still form cycles in Swift
Manage graph, not retain/release

```swift
weak var x : Optional<T> / T? = object

Returns T or nil when accessed, based on existence of object
```

unowned
Memory Management with Swift

Language tools

Obj-C code can still mismatch Retain/Release
Can still form cycles in Swift
Manage graph, not retain/release

weak var x : Optional<T> / T? = object
Returns T or nil when accessed, based on existence of object

unowned let / var x : T = object
Returns T always, but if object doesn't exist… deterministic
^block Captures
Here be dragons

[self.currentGame registerForStateChanges:^{
    if (self.currentGame == newGame) {
        [self.tableView reloadData];
    }
}];

'self' and 'newGame' captured strongly
^block Captures
Here be dragons

__weak typeof(newGame) weakGame = newGame;
__weak typeof(self) weakSelf = self;
[self.currentGame registerForStateChanges:^{
    if (self.currentGame == newGame) {
        [self.tableView reloadData];
    }
}];

'self' and 'newGame' captured strongly
Here be dragons

```swift
__weak typeof(newGame) weakGame = newGame;
__weak typeof(self) weakSelf = self;
[self.currentGame registerForStateChanges:^{
    if (weakSelf.currentGame == weakGame) {
        [weakSelf.tableView reloadData];
    }
}];
```
Swift Closures

Behold, the power of capture lists

currentGame.registerForStateChanges() {
    if self.currentGame == newGame {
        self.tableView!.reloadData()
    }
}

currentGame.registerForStateChanges() { [weak self, newGame] in
    if self.currentGame == newGame {
        self.tableView?.reloadData()
    }
}
Swift Closures
Behold, the power of capture lists

currentGame.registerForStateChanges() {
    [weak self, newGame] in
    if self?.currentGame == newGame {
        self?.tableView!.reloadData()
    }
}
currentGame.registerForStateChanges() { [weak self, newGame] in
    if self?.currentGame == newGame {
        self?.tableView!.reloadData()
    }
}
Time Profiling

Kris Markel
Performance Tools Engineer
Why?
Why?

To provide a great user experience
Why?

To provide a great user experience
• Faster app launch times
Why?

To provide a great user experience
- Faster app launch times
- Keep the frame rate at 60fps
Why?

To provide a great user experience

• Faster app launch times
• Keep the frame rate at 60fps
• Buttery-smooth scrolling
Why?

To provide a great user experience

• Faster app launch times
• Keep the frame rate at 60fps
• Buttery-smooth scrolling
• Responsive UI
What?

An instrument that samples stack trace information at prescribed intervals
Provides an idea of how much time is spent in each method
When?
When?

Investigate specific problems
When?

Investigate specific problems
• If you see stuttering or frame rate slowdowns
When?

Investigate specific problems
• If you see stuttering or frame rate slowdowns
• Some part of your app is taking too long
When?

Investigate specific problems
• If you see stuttering or frame rate slowdowns
• Some part of your app is taking too long

Identify and fix hotspots before they become problems
When?

Investigate specific problems
• If you see stuttering or frame rate slowdowns
• Some part of your app is taking too long

Identify and fix hotspots before they become problems
• Keep an eye on the CPU gauge in Xcode
Demo
Time Profiler in action
Review

Track view

Identify and zoom into problem areas

- Drag to apply a time range filter
- Shift+drag to zoom in
- Control+drag to zoom out
Review

New Inspector panes

Use keyboard shortcuts to quickly move between panes
Review

New Inspector panes

Use keyboard shortcuts to quickly move between panes

• ⌘ 1—Record settings
Review

New Inspector panes

Use keyboard shortcuts to quickly move between panes

• ⌘1—Record settings
• ⌘2—Display settings
Review

New Inspector panes

Use keyboard shortcuts to quickly move between panes

• ⌘1—Record settings
• ⌘2—Display settings
• ⌘3—Extended detail
Review

Strategy views
Review

Strategy views

- Cores strategy
Review

Strategy views

- Cores strategy
- Instruments strategy
Review

Strategy views

• Cores strategy
• Instruments strategy
• Threads strategy
Review

Strategy views

- Cores strategy
- Instruments strategy
- Threads strategy
  - Enable Record Waiting Threads to expose blocked threads
Review
Call Tree settings

Call Tree
- Separate by Thread
- Invert Call Tree
- Hide Missing Symbols
- Hide System Libraries
- Flatten Recursion
- Top Functions
Review

Call Tree settings

- Expensive calls are frequently near the end of the call stack
Review

Call Tree settings

- Expensive calls are frequently near the end of the call stack
- Focus on your own code
Tips

Focus and Prune

Ignore unwanted data

• Charge moves the associated cost
• Prune removes the associated cost
• Focus is “prune everything but”
Two More Guidelines

When using Time Profiler
Two More Guidelines

When using Time Profiler
- Profile Release builds
Two More Guidelines

When using Time Profiler
• Profile Release builds
• For iOS, profile on the device
Performance Counters

Joe Grzywacz
Performance Tools Engineer
What Are Counters?

Each processor core contains a small number of 64-bit hardware registers.
What Are Counters?

Each processor core contains a small number of 64-bit hardware registers
  • Typically only four to eight per core
What Are Counters?

Each processor core contains a small number of 64-bit hardware registers

• Typically only four to eight per core
• Separate from the integer and floating point registers
What Are Counters?

Each processor core contains a small number of 64-bit hardware registers

• Typically only four to eight per core
• Separate from the integer and floating point registers

Each register can be configured to either:
What Are Counters?

Each processor core contains a small number of 64-bit hardware registers

- Typically only four to eight per core
- Separate from the integer and floating point registers

Each register can be configured to either:

- Count one of a small number of events
What Are Counters?

Each processor core contains a small number of 64-bit hardware registers
- Typically only four to eight per core
- Separate from the integer and floating point registers

Each register can be configured to either:
- Count one of a small number of events
  - Instructions executed, L2 Cache Misses, Branches Taken, …
What Are Counters?

Each processor core contains a small number of 64-bit hardware registers:
- Typically only four to eight per core
- Separate from the integer and floating point registers

Each register can be configured to either:
- Count one of a small number of events
  - Instructions executed, L2 Cache Misses, Branches Taken, …
- Take a callstack every time a predetermined number of events occurs
Performance Monitoring Interrupts (PMIs)
Performance Monitoring Interrupts (PMIs)

- Time-Based Sampling
- # Branches Executed

Graph showing the number of branches executed over time.
Performance Monitoring Interrupts (PMIs)

- PMI-Based Sampling
- Time-Based Sampling

# Branches Executed
Performance Counters
How are they useful?
Performance Counters

How are they useful?

Provide a deeper understanding of your app’s performance beyond just time
Performance Counters

How are they useful?

Provide a deeper understanding of your app’s performance beyond just time

- How well CPU resources are being used
  - Caches, execution units, TLBs, …
Performance Counters

How are they useful?

Provide a deeper understanding of your app’s performance beyond just time

• How well CPU resources are being used
  - Caches, execution units, TLBs, …

• Runtime process traits
  - Branch frequency, instruction mix, …
What’s New with Counters

Formulas support

\[
\text{IPC} = \frac{\text{Instructions}}{\text{Cycles}}
\]

\[
\text{Branch Mispredict Rate} = \frac{\text{BranchesMispredicted}}{\text{BranchesExecuted}}
\]

\[
\text{L1 Cache Miss} \% = 100 \times \frac{(\text{L1CacheLoadMisses} + \text{L1CacheStoreMisses})}{(\text{L1CacheLoads} + \text{L1CacheStores})}
\]
What’s New with Counters

iOS 8 support
• 64-bit ARM devices only
What’s New with Counters

iOS 8 support
- 64-bit ARM devices only

Event Profiler instrument is deprecated
- Same PMI functionality is available within the Counters instrument
Demo

iOS Performance Counters
Counters Summary
Counters Summary

Collects data in a similar manner to Time Profiler
Counters Summary

Collects data in a similar manner to Time Profiler

- This is a statistical representation of your application
Counters Summary

Collects data in a similar manner to Time Profiler

• This is a statistical representation of your application

Counters supports Performance Monitoring Interrupts (PMI)
Counters Summary

Collects data in a similar manner to Time Profiler
• This is a statistical representation of your application
Counters supports Performance Monitoring Interrupts (PMI)
• Allows sampling based on the number of events
Counters Summary

Collects data in a similar manner to Time Profiler
• This is a statistical representation of your application

Counters supports Performance Monitoring Interrupts (PMI)
• Allows sampling based on the number of events
• Note that PMI instruction locations can be imprecise
Counters Summary

Collects data in a similar manner to Time Profiler
• This is a statistical representation of your application

Counters supports Performance Monitoring Interrupts (PMI)
• Allows sampling based on the number of events
• Note that PMI instruction locations can be imprecise

Formulas allow you to combine raw event counts in custom ways
Counters Summary

Collects data in a similar manner to Time Profiler

• This is a statistical representation of your application

Counters supports Performance Monitoring Interrupts (PMI)

• Allows sampling based on the number of events
• Note that PMI instruction locations can be imprecise

Formulas allow you to combine raw event counts in custom ways

• Be sure to save your common formulas in a template
What’s New with DTrace
What’s New with DTrace
Dynamic tracemem

Dynamic tracemem, provides a way to trace dynamically sized arrays

- \texttt{tracemem(address, nbytes\_max, nbytes)}
  - \texttt{nbytes\_max}: maximum size of the array, must be known at compile time
  - \texttt{nbytes}: the actual size of the array you want to copy

- Example:

```c
void CGContextFillRects(CGContextRef c, const CGRect rects[], size_t count);

pid$pid\_MyAppName::CGContextFillRects:entry
{
  this->array = copyin(arg1, sizeof(struct CGRect) * arg2);
  tracemem(this->array, 512, sizeof(struct CGRect) * arg2);
}
```
Improved Histograms

Histogram improvements: **agghist, aggzoom, aggpack**

Other New Features

Wait for process to start with `-W`

dtrace -Z -W MyAppName 'pid$target::*CALayer::*entry'
Other New Features

Wait for process to start with \(-W\)

\texttt{dtrace -Z -W MyAppName \textquoteleft \texttt{pid$target::*CALayer*:entry}\textquoteright} \\
Tunable internal DTrace variables

\texttt{# List the tunable variables} \\
\texttt{sysctl kern.dtrace}
Other New Features

Wait for process to start with \(-W\)

dtrace \(-Z\) \(-W\) MyAppName ‘pid$target::*CALayer::*entry’

Tunable internal DTrace variables

# List the tunable variables
sysctl kern.dtrace

Updated documentation

man dtrace
Summary

Profile Swift and Objective-C alike
Be proactive
Don't assume—profile, change, and iterate
More Information

Dave DeLong
Developer Tools Evangelist
delong@apple.com

Instruments Documentation
Instruments User Guide
Instruments User Reference
http://developer.apple.com

Apple Developer Forums
http://devforums.apple.com
# Related Sessions

<table>
<thead>
<tr>
<th>Session</th>
<th>Location</th>
<th>Date/Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creating Extensions for iOS and OS X, Part 1</td>
<td>Mission</td>
<td>Tuesday 2:00PM</td>
</tr>
<tr>
<td>Integrating Swift with Objective-C</td>
<td>Presidio</td>
<td>Wednesday 9:00AM</td>
</tr>
<tr>
<td>Creating Extensions for iOS and OS X, Part 2</td>
<td>Mission</td>
<td>Wednesday 11:30AM</td>
</tr>
<tr>
<td>Swift Interoperability In-Depth</td>
<td>Presidio</td>
<td>Wednesday 3:15PM</td>
</tr>
<tr>
<td>Advanced Swift Debugging in LLDB</td>
<td>Mission</td>
<td>Friday 9:00AM</td>
</tr>
<tr>
<td>Labs</td>
<td>Tools Lab</td>
<td>Time</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>--------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>Swift Lab</td>
<td>Tools Lab A</td>
<td>All Week</td>
</tr>
<tr>
<td>Instruments Lab</td>
<td>Tools Lab B</td>
<td>Wednesday 9:00AM</td>
</tr>
<tr>
<td>Power and Performance Lab</td>
<td>Core OS Lab B</td>
<td>Wednesday 2:00PM</td>
</tr>
<tr>
<td>Instruments Lab</td>
<td>Tools Lab B</td>
<td>Thursday 9:00AM</td>
</tr>
<tr>
<td>Power and Performance Lab</td>
<td>Core OS Lab A</td>
<td>Thursday 3:15PM</td>
</tr>
</tbody>
</table>