SWIFT MEMORY PERFORMANCE -For Beginners, Advanced, Experts And Fanatics

ALEX HOPPEN

COCOAHEADS AACHEN JULY 20







Stack

Cheap allocation

- Decrement and increment stack pointer
- Lifetime: Function call

HEAP

- Expensive allocation
 - Lookup of free memory in advanced data structure
- Lifetime: Dynamic (until freed)
 - Memory leaks possible

HEAP VS. STACK ALLOCATION IN C

Stack

- Cheap allocation
 - Decrement and increment stack pointer
- Lifetime: Function call



HEAP

Expensive allocation

- Lookup of free memory in advanced data structure
- Lifetime: Dynamic (until freed)
 - Memory leaks possible



Reference Counting

- Lifetime on Heap is managed through reference counting
- See slides 40 49 of "Understanding Swift Performance" from WWDC16
 - https://developer.apple.com/wwdc16/416

RETAIN/RELEASE

- Costly because they need to be thread safe
- Costly = ~ 7 ns = ~ 20 processor cycles
- Object needs to be locked before they can be retained or released

PASSING STRUCTS AROUND

- All members need to be copied
 - All referenced result in a retain call



structs

-classes

PASSING SWIFT CLASSES AND STRUCTS AROUND

VALUE-SEMANTICS VS REFERENCE-SEMANTICS

- Value-semantics
 - Better safety guarantee since the state can't change "under our feet"
- Reference-semantics
 - Allow shared state
 - Necessary if shared state is required 6



FOR ADVANCED



VALUE/REFERENCE-TYPE ≠ VALUE/REFERENCE-SEMANTICS





COPY-ON-WRITE TYPE

- Value type that contains a reference type storage
- Custom setters for each member variable
 - Copy storage if it is not uniquely referenced
 - uniquely referenced = has a retain count of 1
- Useful for structs that
 - Contain a lot of data
 - Contain a lot of reference types

COPY-ON-WRITE-TYPE



ENUMS

- Normal enums
 - Store values inline
 - \rightarrow Value types
- Indirect enums
 - Allow circular reference
 - ► → Reference types with value semantics

PASSING DIRECT AND INDIRECT ENUMS AROUND



PROTOCOLS

- Size of implementing object unknown
- Type of implementing object (value vs. reference) unknown
- Existential container performs abstraction
- There is an abstraction cost
- - https://developer.apple.com/wwdc16/416

See slides 142 – 154 of "Understanding Swift Performance" from WWDC16



Protocols

Use Enums Instead of Protocols

- If set of protocol-implementing type protocol
- Avoids existential abstraction layer

If set of protocol-implementing types is known, pass enum around instead of

protocol Shape {}

struct Line: Shape {} class Polygon: Shape {}

extension Shape { ... }

enum Shape {
 case line(Line)
 case polygon(Polygon)
}

struct Shape: ShapeMixin {}
class Polygon: ShapeMixin {}

protocol ShapeMixin {}
extension ShapeMixin { ... }

FOR PERFORMANCE-NERDS



MEASURE PERFORMANCE MPACT USING INSTRUMENTS

- Run performance critical code in Instruments
- Invert Call tree to find expensive functions
- You have a retain/release
 bottleneck if it starts showing up
 right at the top



Instruments			
		Alex's MacBoo	A Pro } ■ test Run 1 of 1 00:00:02 +
ATTRIBUTE - target INSTRUMENT - * Target Threads CPUs Instruments			
O0:00.000 O0:10.000 O0:20.000 O0:30.000 O0:40.000 O0:5 Time Profiler Instrument CPU Usage Instrument <			
me Profiler $ ightarrow$ Profile $ ightarrow$ Root			
W	/eight∽	Self Weight	Symbol Name
2.61 s 1	00.0%	0 s	▼test (17033)
2.61 s 1	00.0%	0 s	▼Main Thread 0xf31994
1.52 s	58.4%	1.52 s [swift_retain libswiftCore.dylib
.00 ms	31.7%	827.00 ms [swift_release_n libswiftCore.dylib
7.00 ms	4.4%	117.00 ms [swift_getFunctionReplacement libswiftCore.dylib
6.00 ms	3.6%	96.00 ms 🗾	▶performanceTest() test 📀
3.00 ms	0.8%	23.00 ms 🗾	►DYLD-STUB\$\$swift_retain test
9.00 ms	0.3%	9.00 ms 🚺	▶getattrlist dyld 🚭
3.00 ms	0.1%	3.00 ms 🚺	▶stat64 dyld
1.00 ms	0.0%	1.00 ms 🚺	ImageLoaderMachO::segUnaccessible(unsigned int) const dyld
1.00 ms	0.0%	1.00 ms 🚺	ImageLoaderMachOCompressed::eachBind(ImageLoader::LinkContext const&, unsigned long (ImageLoader:
1.00 ms	0.0%	1.00 ms [header_info::getHeaderInfoRW() libobjc.A.dylib
1.00 ms	0.0%	1.00 ms 🚺	ImageLoader::recursiveLoadLibraries(ImageLoader::LinkContext const&, bool, ImageLoader::RPathChain cor
1.00 ms	0.0%	1.00 ms 🚺	ImageLoaderMachO::getInstallPath() const dyld
1.00 ms	0.0%	1.00 ms 🚺	ImageLoader::hash(char const*) dyld
1.00 ms	0.0%	1.00 ms 🚺	dyld3::findInSharedCacheImage(dyld3::SharedCacheLoadInfo const&, char const*, dyld3::SharedCacheFind
1.00 ms	0.0%	1.00 ms 🚺	dyld::loadPhase5(char const*, char const*, dyld::LoadContext const&, unsigned int&, std::_1::vector <char c<="" p=""></char>
ut Filter	=	Involves Symbol	Call Tree Constraints Data Mining



LOGGING RETAIN AND RELEASE CALLS

- Set symbolic breakpoint on swift_retain, swift_retain_n and/or swift_release, swift_release_n
- po \$arg1 prints name of class being retained
- p \$arg1 prints memory address of retained object
- For _n functions p \$arg2 prints number by which retain count should be increased/decreased, for other functions it's 1

LOGGING RETAIN AND RELEASE CALLS

If object of type MyClass at 4302715776 is retained, then print object using expr -l Swift -- unsafeBitCast(4302715776, to: MyClass.self)

If class can't be found, step out into Swift context, afterwards class is known





MEASURE PERFORMANCE IMPACT AND LOG RETAIN/RELEASE CALLS

OUTPUT SIL LEVEL

- SIL (Swift Intermediate Language) contains much more low-level code
 - Including retain and release calls
- code
 - Retrieve Swift compiler call from Build Log
 - with -emit-sil

Compile with the -emit-sil instead of -c to output SIL instead of object

Might need to remove options like -incremental that are incompatible

DEBUGGING SIL

- Add -g -Xfrontend -gsilto "Other Swift Flags"
- Debugger will consider SIL as source language instead of Swift
- Stepping is based on SIL
- Set symbolic breakpoint using mangled name of interesting function
- To unmangle names use xcrun swift-demangle <demangled-name>



OUTPUT AND DEBUG SIL

Swift Calling Conventions

- @guaranteed (aka +0): Caller guarantees that argument is alive for entire function call
 - Callee needs to retain object if it is being stored
- @owned (aka +1): Caller passes the argument with a reference count of +1
 - Callee needs to release object if it doesn't store it
- @inout: Caller passes an argument, caller may modify argument
 - Callee needs to retain object if it is being stored

Use inout For Argument Passing

