目录

iOS逆向：Swift逆向

- 最新版本：`v1.3.0`
- 更新时间：`20240302`

简介

整理iOS逆向期间，关于Swift逆向相关的各种知识。最开始是Swift逆向概览；其中包括脑图；然后是Swift基础知识；接着是逆向常会涉及到的Swift常用函数；然后是Swift逆向常涉及的方面，包括静态分析和动态调试，静态分析包括导出头文件和IDA分析；然后是最终要的，通用逻辑，包括TypeMetadata、VWT、StructMetadata以及ClassMetadata，和通用的MetadataKind等；然后是常用的类型，包括Array数组和内存布局详情、Bool变量、Data数据和内存布局详情、Dictionary字典和内存布局详情、Enum枚举、Set集合和内存布局详情、String字符串和内存布局详情、Struct结构体和内存布局详情、Tuple元祖。

源码+浏览+下载

本书的各种源码、在线浏览地址、多种格式文件下载如下：

HonKit源码

- crifan/ios_re_swift_reverse: iOS逆向：Swift逆向

如何使用此HonKit源码去生成发布为电子书

详见：crifan/honkit_template: demo how to use crifan honkit template and demo

在线浏览

- iOS逆向：Swift逆向 book.crifan.org
- iOS逆向：Swift逆向 crifan.github.io

离线下载阅读

- iOS逆向：Swift逆向 PDF
- iOS逆向：Swift逆向 ePub
- iOS逆向：Swift逆向 Mobi

版权和用途说明

鸣谢

感谢我的老婆**陈雪**的包容理解和悉心照料，才使得我 `crifan` 有更多精力去专注技术专研和整理归纳出这些电子书和技术教程，特此鸣谢。

其他

作者的其他电子书

本人 `crifan` 还写了其他 `150+` 本电子书教程，感兴趣可移步至：

crifan/crifan_ebook_readme: Crifan的电子书的使用说明

关于作者

关于作者更多介绍，详见：

关于CrifanLi李茂 – 在路上

Swift逆向概览

iOS逆向期间，有些app二进制代码内部是：`ObjC`和Swift混合的。

所以涉及到：Swift逆向。

Swift逆向脑图

此处用脑图去表示出此教程的核心内容：

- Swift逆向脑图
  - Swift逆向脑图| ProcessOn免费在线作图,在线流程图,在线思维导图

Swift基础知识

Swift函数

SwiftObject

swift_getInitializedObjCClass

源码

swift/stdlib/public/runtime/SwiftObject.mm at main · apple/swift (github.com)

```
Class swift::swift_getInitializedObjCClass(Class c) {
  // Used when we have class metadata and we want to ensure a class has been
  // initialized by the Objective-C runtime. We need to do this because the
  // class "c" might be valid metadata, but it hasn't been initialized yet.
  // Send a message that's likely not to be overridden to minimize potential
  // side effects. Ignore the return value in case it is overridden to
  // return something different. See
  // https://github.com/apple/swift/issues/52863 for an example.
  [c self];
  return c;
}
```

总结

- Swift的 `swift_getInitializedObjCClass` 函数定义
  - `Class swift::swift_getInitializedObjCClass(Class c)`
    - 参数
      - `Class c`
    - 返回值
      - 类型：Class

UnsafeMutableBufferPointer

UnsafeMutableBufferPointer.init(start:count:)

源码

`swift/stdlib/public/core/UnsafeRawBufferPointer.swift.gyb`

```
@frozen
public struct Unsafe${Mutable}RawBufferPointer {
  @usableFromInline
  internal let _position, _end  Unsafe${Mutable}RawPointer?


  /// Creates a buffer over the specified number of contiguous bytes starting
  /// at the given pointer.
  ///
  /// - Parameters:
  ///   - start: The address of the memory that starts the buffer. If `starts`
  ///     is `nil`, `count` must be zero. However, `count` may be zero even
  ///     for a non-`nil` `start`.
  ///   - count: The number of bytes to include in the buffer. `count` must not
  ///     be negative.
  @inlinable
  public init(
    @_nonEphemeral start  Unsafe${Mutable}RawPointer?, count  Int
  ) {
    _debugPrecondition(count >= 0, "${Self} with negative count")
    _debugPrecondition(count == 0 || start != nil,
      "${Self} has a nil start and nonzero count")
    _position = start
    _end = start.map { $0 + _assumeNonNegative(count) }
  }
}
```

总结

- UnsafeMutableBufferPointer
  - `init(start:count:)`
    - 定义
      - `init(start: UnsafeMutablePointer<Element>?, count: Int)`
    - 参数
      - `start: UnsafeMutablePointer<Element>?`
      - `count: Int`
    - 返回值
      - `指针`

Array

Array.formIndex(after:)

源码

`swift/stdlib/public/core/Array.swift`

```
/// Replaces the given index with its successor.
///
/// - Parameter i: A valid index of the collection. `i` must be less than
///   `endIndex`.
@inlinable
public func formIndex(after i: inout Int) {
  // NOTE: this is a manual specialization of index movement for a Strideable
  // index that is required for Array performance.  The optimizer is not
  // capable of creating partial specializations yet.
  // NOTE: Range checks are not performed here, because it is done later by
  // the subscript function.
  i += 1
}
```

总结

- Swift函数：Array.formIndex(after:)
  - 定义
    - formIndex(after i: inout Int)
  - 参数
    - 变量名
      - 传入：after
      - 内部：i
      - 类型：inout Int == 指针类型，内部会改变值
  - 返回值
    - 传入的after==i （值已加1）

Set

_NativeSet._unsafeInsertNew

`swift/stdlib/public/core/NativeSet.swift`

```
extension _NativeSet { // Insertions
  /// Insert a new element into uniquely held storage.
  /// Storage must be uniquely referenced with adequate capacity.
  /// The `element` must not be already present in the Set.
  @inlinable
  internal func _unsafeInsertNew(_ element: __owned Element) {
    _internalInvariant(count + 1 <= capacity)
    let hashValue = self.hashValue(for: element)
    if _isDebugAssertConfiguration() {
      // In debug builds, perform a full lookup and trap if we detect duplicate
      // elements -- these imply that the Element type violates Hashable
      // requirements. This is generally more costly than a direct insertion,
      // because we'll need to compare elements in case of hash collisions.
      let (bucket, found) = find(element, hashValue: hashValue)
      guard !found else {
        #if !$Embedded
        ELEMENT_TYPE_OF_SET_VIOLATES_HASHABLE_REQUIREMENTS(Element.self)
        #else
        fatalError("duplicate elements in a Set")
        #endif
      }
      hashTable.insert(bucket)
      uncheckedInitialize(at: bucket, to: element)
    } else {
      let bucket = hashTable.insertNew(hashValue: hashValue)
      uncheckedInitialize(at: bucket, to: element)
    }
    _storage._count &+= 1
  }


  /// Insert a new element into uniquely held storage.
  /// Storage must be uniquely referenced.
  /// The `element` must not be already present in the Set.
  @inlinable
  internal mutating func insertNew(_ element: __owned Element, isUnique: Bool) {
    _ = ensureUnique(isUnique: isUnique, capacity: count + 1)
    _unsafeInsertNew(element)
  }


  @inlinable
  internal func _unsafeInsertNew(_ element: __owned Element, at bucket: Bucket) {
    hashTable.insert(bucket)
    uncheckedInitialize(at: bucket, to: element)
    _storage._count += 1
  }
```

2024-02-24 17:14:52

Swift逆向相关

Swift逆向之静态分析

导出Swift头文件

TODO：把导出ObjC和Swift混淆时会报错的问题，和解决办法（最新版的class-dump），整理过来

IDA分析Swift

TODO：把已加到IDA中的，各种类型的定义，整理过来

Swift逆向之动态调试

调试时，辅助用IDA打开Swift的各种库文件，比如：

- `libswiftCore.dylib`
- `libswiftFoundation.dylib`
- 等等

方便调试内容。

这样，通过动态调试和查看IDA中的分析，函数的伪代码实现等等，就可以有更深入的理解很多函数和变量了。

TODO：把下面帖子中有用的内容，整理过来：

- 【未解决】iOS逆向Swift：swift_getAssociatedTypeWitness
- 【未解决】iOS逆向Swift：Data.withUnsafeMutableBytes(_:)
- 【已解决】iOS逆向Swift：Data.withUnsafeBytes(_:)
- 【未解决】iOS逆向WhatsApp：swiftPodCopy_10039A1B4
- 【未解决】iOS逆向Swift：Set.init(_:)
-
- 【记录】iOS逆向Swift：IDA静态分析libswiftFoundation.dylib
  - 。。。
-
- 【已解决】iOS逆向Swift：库文件libswiftCore.dylib
  - 【已解决】iOS逆向WhatsApp：ArrayAdoptStorage_2ToW1_CB2C
    - static Swift.Array.*adoptStorage(*: __owned Swift._ContiguousArrayStorage<τ_0_0>, count: Swift.Int) -> (Swift.Array<τ_0_0>, Swift.UnsafeMutablePointer<τ_0_0>)
  - 【已解决】iOS逆向：dyld_stub_binder
  - 【已解决】iOS逆向Swift：String的WitnessTable详情
    - destroy value witness for Swift.String
  - 【已解决】iOS逆向Swift：Float对应的Builtin.Int32的VWT的具体值
    - type metadata for Swift.Float
  - 【未解决】iOS逆向Swift：_NativeSet._unsafeInsertNew
    - Swift.*NativeSet._unsafeInsertNew(*: __owned τ_0_0, at: Swift._HashTable.Bucket) -> ()
  - 【已解决】iOS逆向Swift：Optional的VWT=ValueWitnessTable
  - 【未解决】iOS逆向Swift：_SetStorage.allocate
    - static Swift._SetStorage.allocate(capacity: Swift.Int) -> Swift._SetStorage<τ_0_0>
  - 【已解决】iOS逆向Swift：Xcode中给Swift函数_NativeSet.init加断点
    - Swift._NativeSet.init(capacity: Swift.Int) -> Swift._NativeSet<τ_0_0>
  - 【未解决】iOS逆向Swift：NativeSet
  - Collection
    - 【未解决】iOS逆向WhatsApp：Collection.map(_:)(void)
    - 【未解决】iOS逆向Swift：swift_getAssociatedTypeWitness
      - Swift.Collection.isEmpty.getter : Swift.Bool

        - 【基本解决】iOS逆向Swift：protocol requirements base descriptor的含义
            - protocol requirements base descriptor for Swift.Collection
    - 【已解决】iOS逆向：Swift的pod_copy
        - pod_copy(swift::OpaqueValue*, swift::OpaqueValue*, swift::TargetMetadata<_inprocess class="copyright"> const*)
    - 【已解决】iOS逆向：Swift._ContiguousArrayStorage
        - type metadata accessor for Swift._ContiguousArrayStorage
    - 【已解决】iOS逆向：swift_getTypeByMangledNameInContext
    - 【未解决】iOS逆向：__swift_instantiateConcreteTypeFromMangledName
    - 【已解决】iOS逆向WhatsApp：Swift函数String.append(_:)的字符串拼接的实现逻辑
        - Swift.String.append(Swift.String) -> ()

Swift通用逻辑

- 其他
  - Protocol
    - `PWT` = `Protocol Witness Table` = `ProtocolWitnessTable`
    - `ValueBuffer`
  - Exitential Container?
    - Box?
  - Opaque ?

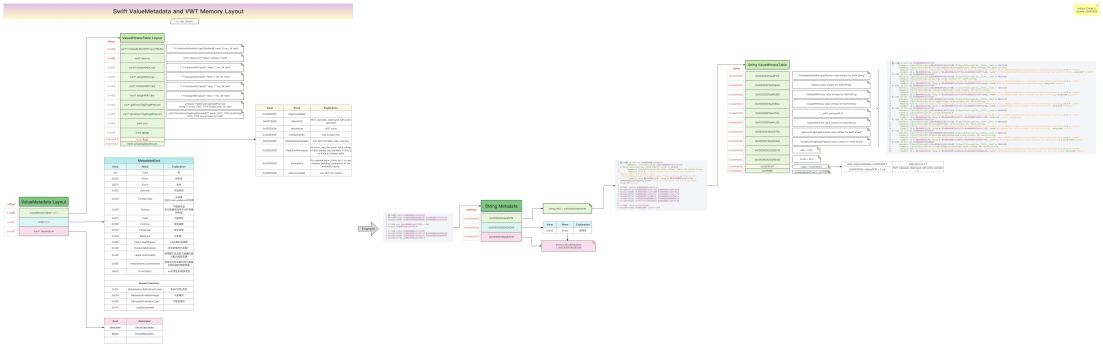
TypeMetadata

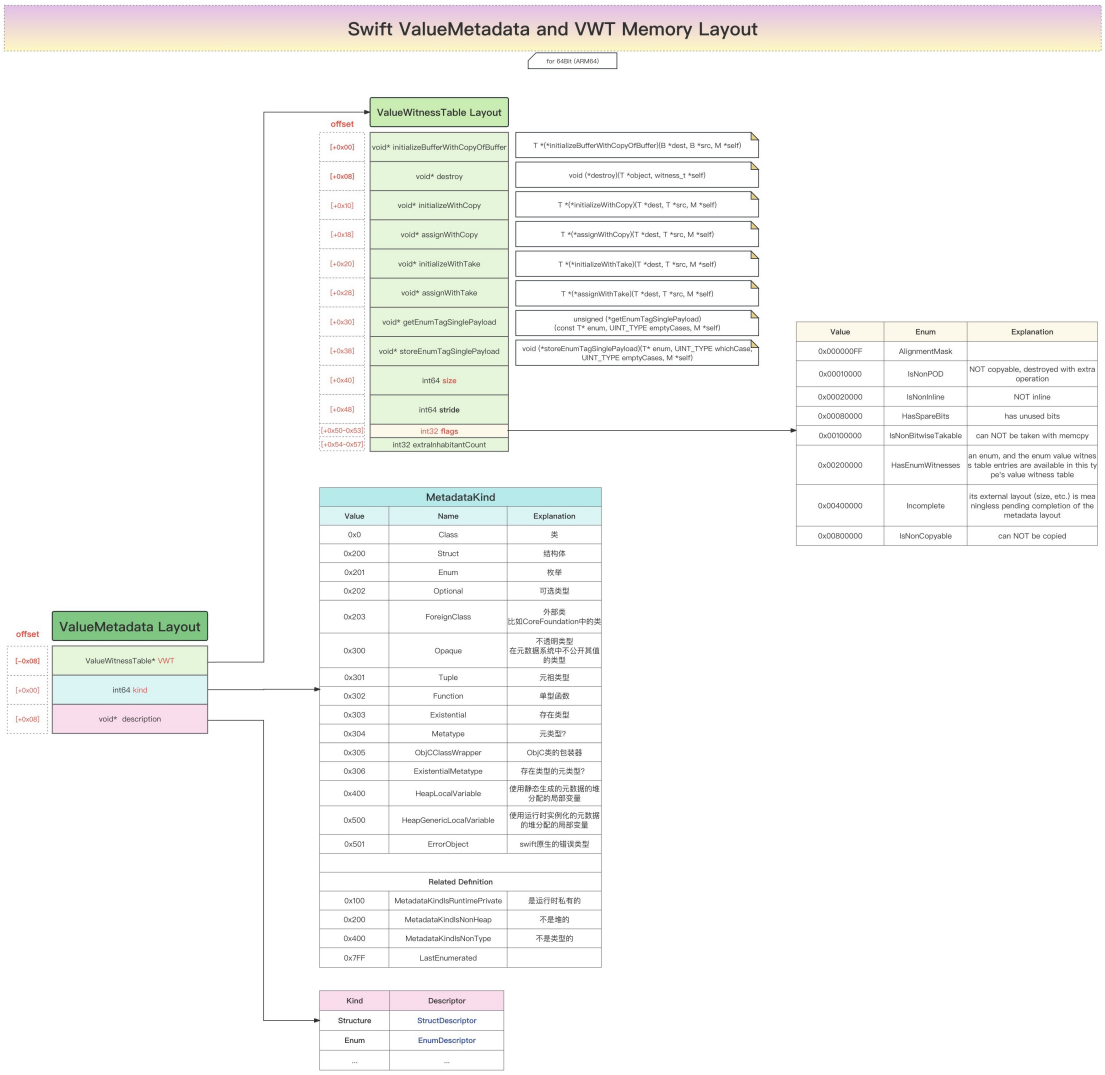
- Swift中有几种Metadata
  - 主要有2类
    - Class的：TargetClassMetadata == ClassMetadata
    - 非Class的=（Struct、Enum、Optional等）Value的：TargetValueMetadata = ValueMetadata
  - 通用逻辑
    - TargetClassMetadata和TargetValueMetadata，都继承自：TargetMetadata
      - TargetMetadata
        - 属性=字段
          - Kind 另外一种表述：
- Swift中的 `Metadata`
  - **TargetMetadata**
    - **ClassMetadata**
      - == `TargetClassMetadata`
      - 适用于：**Class类**
    - **ValueMetadata**
      - == `TargetValueMetadata`
      - 适用于：**非Class ==** `Struct` 、 `Enum` 、 `Optional` 等**Value**

ValueMetadata

内存布局

图

- Swift的ValueMetadata和VWT的内存布局结构图 = Swift ValueMetadata and VWT Memory Layout
  - 在线预览
    - Swift的ValueMetadata和VWT的内存布局结构图| ProcessOn免费在线作图,在线流程图,在线思维导图
  - 离线查看
    - 
  - 核心定义



文字

- Swift中的：TargetValueMetadata
  - 继承关系
    - TargetValueMetadata = ValueMetadata
      - TargetMetadata
  - 字段=属性=内存布局
    - TargetMetadata
      - StoredPointer Kind
        - the kind. Only valid for non-class metadata; getKind() must be used to get the kind value
    - TargetValueMetadata
      - TargetSignedPointer* Description
        - An out-of-line description of the type
        - 说明
          - 此处的Description根据具体类型不同，则是不同的内容
        - 举例
          - Struct
            - TargetStructDescriptor=StructDescriptor

加到IDA中的定义

```
struct ValueMetadata
{
  __int64 kind;
  void *description;
};
```

`VWT = ValueWitnessTable`

概述

VWT结构布局

文字

- `Swift 的 VWT = Value Witness Table` 结构布局=字段=属性
  - [+0x00] = void* `initializeBufferWithCopyOfBuffer`
  - [+0x08] = void* `destroy`
  - [+0x10] = void* `initializeWithCopy`
  - [+0x18] = void* `assignWithCopy`
  - [+0x20] = void* `initializeWithTake`
  - [+0x28] = void* `assignWithTake`
  - [+0x30] = void* `getEnumTagSinglePayload`
  - [+0x38] = void* `storeEnumTagSinglePayload`
  - [+0x40] = void* int64 `size`
  - [+0x48] = void* int64 `stride`
  - [+0x50~0x53] = int32 `flags`
  - [+0x54~0x57] = int32 `extraInhabitantCount`

图

详见：

ValueMetadata

加入到IDA中的定义

- struct `ValueWitnessTable`

```
struct __cppobj ValueWitnessTable
{
  void (__fastcall *initializeBufferWithCopyOfBuffer)(void *dst, void *src, void *metad
ataSelf);
  void (__fastcall *destroy)(void *object, void *witnessSelf);
  void (__fastcall *initializeWithCopy)(void *dst, void *src, void *metadataSelf);
  void (__fastcall *assignWithCopy)(void *dst, void *src, void *metadataSelf);
  void (__fastcall *initializeWithTake)(void *dst, void *src, void *metadataSelf);
  void (__fastcall *assignWithTake)(void *dst, void *src, void *metadataSelf);
  unsigned __int64 (__fastcall *getEnumTagSinglePayload)(void *enumPtr, __int64 emptyCa
ses, void *metadataSelf);
  void (__fastcall *storeEnumTagSinglePayload)(void *enumPtr, __int64 whichCase, void *
metadataSelf);
  __int64 size;
  __int64 stride;
```

```
  TargetValueWitnessFlags flags;
  __int32 extraInhabitantCount;
};
```

- enum `TargetValueWitnessFlags`

```
000000FF ; enum TargetValueWitnessFlags, mappedto_10385, bitfield, width 4 bytes
000000FF AlignmentMask    EQU 0xFF
00010000 IsNonPOD         EQU 0x10000
00020000 IsNonInline      EQU 0x20000
00080000 HasSpareBits     EQU 0x80000
00100000 IsNonBitwiseTakable  EQU 0x100000
00200000 HasEnumWitnesses  EQU 0x200000
00400000 Incomplete       EQU 0x400000
00800000 IsNonCopyable    EQU 0x800000
00800000
```

详解

内存布局

- `Swift` 的 `VWT` = `Value Witness Table` 结构布局=字段=属性
  - [+0x00] = initializeBufferWithCopyOfBuffer
    - 定义：`T *(*initializeBufferWithCopyOfBuffer)(B *dest, B *src, M *self);`
  - [+0x08] = destroy
    - 定义：`void (*destroy)(T *object, witness_t *self);`
  - [+0x10] = initializeWithCopy
    - 定义：`T *(*initializeWithCopy)(T *dest, T *src, M *self);`
  - [+0x18] = assignWithCopy
    - 定义：`T *(*assignWithCopy)(T *dest, T *src, M *self);`
  - [+0x20] = initializeWithTake
    - 定义：`T *(*initializeWithTake)(T *dest, T *src, M *self);`
  - [+0x28] = assignWithTake
    - 定义：`T *(*assignWithTake)(T *dest, T *src, M *self);`
  - [+0x30] = getEnumTagSinglePayload
    - 定义：`unsigned (*getEnumTagSinglePayload)(const T* enum, UINT_TYPE emptyCases, M *self);`
  - [+0x38] = storeEnumTagSinglePayload
    - 定义：`void (*storeEnumTagSinglePayload)(T* enum, UINT_TYPE whichCase, UINT_TYPE emptyCases, M *self);`
  - [+0x40] = size
    - 定义：`SIZE_TYPE size;`
  - [+0x48] = stride
    - 定义：`SIZE_TYPE stride;`
  - [+0x50~0x53] = flags
    - 定义：`UINT_TYPE flags;`

    - [+0x54~0x57] = extraInhabitantCount
        - 定义：`UINT_TYPE extraInhabitantCount;`
- 说明
    - `SIZE_TYPE` = `StoredSize` = `size_t` ? = `int64`
    - `UINT_TYPE` = `unsigned` = `int32`

Swfit代码

swift-language/include/swift/ABI/ValueWitness.def at master · eaplatanios/swift-language

```
///   T *(*initializeBufferWithCopyOfBuffer)(B *dest, B *src, M *self);
/// Given an invalid buffer, initialize it as a copy of the
/// object in the source buffer.
FUNCTION_VALUE_WITNESS(initializeBufferWithCopyOfBuffer,
                       InitializeBufferWithCopyOfBuffer,
                       MUTABLE_VALUE_TYPE,
                       (MUTABLE_BUFFER_TYPE, MUTABLE_BUFFER_TYPE, TYPE_TYPE))

BEGIN_VALUE_WITNESS_RANGE(ValueWitness,
                          InitializeBufferWithCopyOfBuffer)
BEGIN_VALUE_WITNESS_RANGE(RequiredValueWitness,
                          InitializeBufferWithCopyOfBuffer)
BEGIN_VALUE_WITNESS_RANGE(RequiredValueWitnessFunction,
                          InitializeBufferWithCopyOfBuffer)

///   void (*destroy)(T *object, witness_t *self);
///
/// Given a valid object of this type, destroy it, leaving it as an
/// invalid object.  This is useful when generically destroying
/// an object which has been allocated in-line, such as an array,
/// struct, or tuple element.
FUNCTION_VALUE_WITNESS(destroy,
                       Destroy,
                       VOID_TYPE,
                       (MUTABLE_VALUE_TYPE, TYPE_TYPE))

///   T *(*initializeWithCopy)(T *dest, T *src, M *self);
///
/// Given an invalid object of this type, initialize it as a copy of
/// the source object.  Returns the dest object.
FUNCTION_VALUE_WITNESS(initializeWithCopy,
                       InitializeWithCopy,
                       MUTABLE_VALUE_TYPE,
                       (MUTABLE_VALUE_TYPE, MUTABLE_VALUE_TYPE, TYPE_TYPE))

///   T *(*assignWithCopy)(T *dest, T *src, M *self);
///
/// Given a valid object of this type, change it to be a copy of the
/// source object.  Returns the dest object.
FUNCTION_VALUE_WITNESS(assignWithCopy,
                       AssignWithCopy,
                       MUTABLE_VALUE_TYPE,
                       (MUTABLE_VALUE_TYPE, MUTABLE_VALUE_TYPE, TYPE_TYPE))
```

```
///   T *(*initializeWithTake)(T *dest, T *src, M *self);
///
/// Given an invalid object of this type, initialize it by taking
/// the value of the source object.  The source object becomes
/// invalid.  Returns the dest object.
FUNCTION_VALUE_WITNESS(initializeWithTake,
                       InitializeWithTake,
                       MUTABLE_VALUE_TYPE,
                       (MUTABLE_VALUE_TYPE, MUTABLE_VALUE_TYPE, TYPE_TYPE))

///   T *(*assignWithTake)(T *dest, T *src, M *self);
///
/// Given a valid object of this type, change it to be a copy of the
/// source object.  The source object becomes invalid.  Returns the
/// dest object.
FUNCTION_VALUE_WITNESS(assignWithTake,
                       AssignWithTake,
                       MUTABLE_VALUE_TYPE,
                       (MUTABLE_VALUE_TYPE, MUTABLE_VALUE_TYPE, TYPE_TYPE))

/// unsigned (*getEnumTagSinglePayload)(const T* enum, UINT_TYPE emptyCases)
/// Given an instance of valid single payload enum with a payload of this
/// witness table's type (e.g Optional<ThisType>) , get the tag of the enum.
FUNCTION_VALUE_WITNESS(getEnumTagSinglePayload,
                       GetEnumTagSinglePayload,
                       UINT_TYPE,
                       (IMMUTABLE_VALUE_TYPE, UINT_TYPE, TYPE_TYPE))

/// void (*storeEnumTagSinglePayload)(T* enum, UINT_TYPE whichCase,
///                                   UINT_TYPE emptyCases)
/// Given uninitialized memory for an instance of a single payload enum with a
/// payload of this witness table's type (e.g Optional<ThisType>), store the
/// tag.
FUNCTION_VALUE_WITNESS(storeEnumTagSinglePayload,
                       StoreEnumTagSinglePayload,
                       VOID_TYPE,
                       (MUTABLE_VALUE_TYPE, UINT_TYPE, UINT_TYPE, TYPE_TYPE))

END_VALUE_WITNESS_RANGE(RequiredValueWitnessFunction,
                        StoreEnumTagSinglePayload)

///   SIZE_TYPE size;
///
/// The required storage size of a single object of this type.
DATA_VALUE_WITNESS(size,
                   Size,
                   SIZE_TYPE)

BEGIN_VALUE_WITNESS_RANGE(TypeLayoutWitness,
                          Size)

BEGIN_VALUE_WITNESS_RANGE(RequiredTypeLayoutWitness,
                          Size)

///   SIZE_TYPE stride;
///
```

```
/// The required size per element of an array of this type. It is at least
/// one, even for zero-sized types, like the empty tuple.
DATA_VALUE_WITNESS(stride,
                   Stride,
                   SIZE_TYPE)


///   UINT_TYPE flags;
///
/// The ValueWitnessAlignmentMask bits represent the required
/// alignment of the first byte of an object of this type, expressed
/// as a mask of the low bits that must not be set in the pointer.
/// This representation can be easily converted to the 'alignof'
/// result by merely adding 1, but it is more directly useful for
/// performing dynamic structure layouts, and it grants an
/// additional bit of precision in a compact field without needing
/// to switch to an exponent representation.
///
/// The ValueWitnessIsNonPOD bit is set if the type is not POD.
///
/// The ValueWitnessIsNonInline bit is set if the type cannot be
/// represented in a fixed-size buffer or if it is not bitwise takable.
///
/// The ExtraInhabitantsMask bits represent the number of "extra inhabitants"
/// of the bit representation of the value that do not form valid values of
/// the type.
///
/// The Enum_HasSpareBits bit is set if the type's binary representation
/// has unused bits.
///
/// The HasEnumWitnesses bit is set if the type is an enum type.
DATA_VALUE_WITNESS(flags,
                   Flags,
                   UINT_TYPE)

///   UINT_TYPE extraInhabitantCount;
///
/// The number of extra inhabitants in the type.
DATA_VALUE_WITNESS(extraInhabitantCount,
                   ExtraInhabitantCount,
                   UINT_TYPE)

END_VALUE_WITNESS_RANGE(RequiredTypeLayoutWitness,
                        ExtraInhabitantCount)

END_VALUE_WITNESS_RANGE(RequiredValueWitness,
                        ExtraInhabitantCount)

END_VALUE_WITNESS_RANGE(TypeLayoutWitness,
                        ExtraInhabitantCount)

#endif /* WANT_REQUIRED_VALUE_WITNESSES */

#if WANT_ENUM_VALUE_WITNESSES

// The following value witnesses are conditionally present if the witnessed
```

```
// type is an enum.

///   unsigned (*getEnumTag)(T *obj, M *self);
///
/// Given a valid object of this enum type, extracts the tag value indicating
/// which case of the enum is inhabited. Returned values are in the range
/// [0..NumElements-1].
FUNCTION_VALUE_WITNESS(getEnumTag,
                       GetEnumTag,
                       INT_TYPE,
                       (IMMUTABLE_VALUE_TYPE, TYPE_TYPE))

BEGIN_VALUE_WITNESS_RANGE(EnumValueWitness,
                          GetEnumTag)

///   void (*destructiveProjectEnumData)(T *obj, M *self);
/// Given a valid object of this enum type, destructively extracts the
/// associated payload.
FUNCTION_VALUE_WITNESS(destructiveProjectEnumData,
                       DestructiveProjectEnumData,
                       VOID_TYPE,
                       (MUTABLE_VALUE_TYPE, TYPE_TYPE))

///   void (*destructiveInjectEnumTag)(T *obj, unsigned tag, M *self);
/// Given an enum case tag and a valid object of case's payload type,
/// destructively inserts the tag into the payload. The given tag value
/// must be in the range [-ElementsWithPayload..ElementsWithNoPayload-1].
FUNCTION_VALUE_WITNESS(destructiveInjectEnumTag,
                       DestructiveInjectEnumTag,
                       VOID_TYPE,
                       (MUTABLE_VALUE_TYPE, UINT_TYPE, TYPE_TYPE))

END_VALUE_WITNESS_RANGE(EnumValueWitness,
                        DestructiveInjectEnumTag)

END_VALUE_WITNESS_RANGE(ValueWitness,
                        DestructiveInjectEnumTag)
```

Swift的VWT和C++对应关系

Swift Value Witness Operation	C++ equivalent
`initializeWithCopy`	copy constructor
`assignWithCopy`	copy assignment operator
`initializeWithTake`	move constructor, followed by a call to destructor on the source
`assignWithTake`	move assignment operator, followed by a call to destructor on the source
`destroy`	destructor
`size`	`sizeof(T)` minus trailing padding
`stride`	`sizeof(T)`

`flags`	among other information, contains alignment, i.e., `alignof(T)`

举例

Builtin.NativeObject的VWT

- Builtin.NativeObject的VWT内存布局
  - [+0x00] = initializeBufferWithCopyOfBuffer
    - `name="swift::metadataimpl::BufferValueWitnesses<swift::metadataimpl::ValueWitnesses< :metadataimpl::SwiftRetainableBox>, true, 8ul, 8ul, (swift::metadataimpl::FixedPacking)1>::initializeBufferWithCopyOfBuffer(swift::TargetValueBuffer< :InProcess>*, swift::TargetValueBuffer< :InProcess>*, swift::TargetMetadata< :InProcess> const*)"`
      - `mangled="_ZN5swift12metadataimpl20BufferValueWitnessesINS0_14ValueWitnessesINS0_18SwiftRetainableBoxEEELb1ELm8ELm8ELNS0_12FixedPackingE1EE32initializeBufferWithCopyOfBufferEPNS_17TargetValueBufferINS_9InProcessEEESA_PKNS_14TargetMetadataIS8_EE"`
  - [+0x08] = destroy
    - `name="swift::metadataimpl::ValueWitnesses< :metadataimpl::SwiftRetainableBox>::destroy(swift::OpaqueValue*, swift::TargetMetadata< :InProcess> const*)"`
      - `mangled="_ZN5swift12metadataimpl14ValueWitnessesINS0_18SwiftRetainableBoxEE7destroyEPNS_11OpaqueValueEPKNS_14TargetMetadataINS_9InProcessEEE"`
  - [+0x10] = initializeWithCopy
    - `name="swift::metadataimpl::ValueWitnesses< :metadataimpl::SwiftRetainableBox>::initializeWithCopy(swift::OpaqueValue*, swift::OpaqueValue*, swift::TargetMetadata< :InProcess> const*)"`
      - `mangled="_ZN5swift12metadataimpl14ValueWitnessesINS0_18SwiftRetainableBoxEE18initializeWithCopyEPNS_11OpaqueValueES5_PKNS_14TargetMetadataINS_9InProcessEEE"`
  - [+0x18] = assignWithCopy
    - `name="swift::metadataimpl::ValueWitnesses< :metadataimpl::SwiftRetainableBox>::assignWithCopy(swift::OpaqueValue*, swift::OpaqueValue*, swift::TargetMetadata< :InProcess> const*)"`
      - `mangled="_ZN5swift12metadataimpl14ValueWitnessesINS0_18SwiftRetainableBoxEE14assignWithCopyEPNS_11OpaqueValueES5_PKNS_14TargetMetadataINS_9InProcessEEE`
  - [+0x20] = initializeWithTake
    - `name="swift::metadataimpl::ValueWitnesses< :metadataimpl::SwiftRetainableBox>::initializeWithTake(swift::OpaqueValue*, swift::OpaqueValue*, swift::TargetMetadata< :InProcess> const*)"`
      - `mangled="_ZN5swift12metadataimpl14ValueWitnessesINS0_18SwiftRetainableBoxEE18initializeWithTakeEPNS_11OpaqueValueES5_PKNS_14TargetMetadataINS_9InProcessEEE"`
  - [+0x28] = assignWithTake
    - `name="swift::metadataimpl::ValueWitnesses< :metadataimpl::SwiftRetainable`

Box>::assignWithTake(swift::OpaqueValue*, swift::OpaqueValue*, swift::TargetMetadata< :InProcess> const*)"

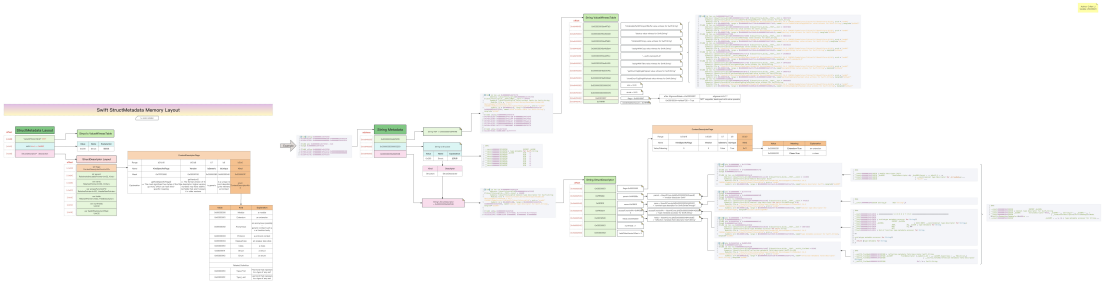
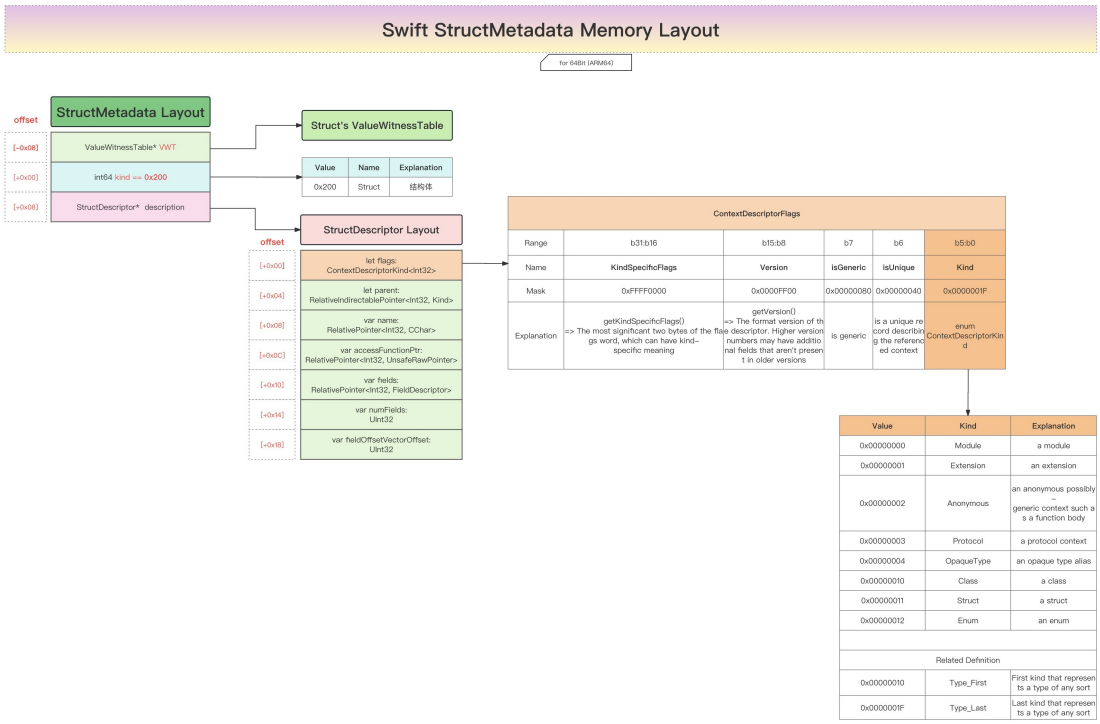
- mangled="_ZN5swift12metadataimpl14ValueWitnessesINS0_18SwiftRetainableBoxEE14assignWithTakeEPNS_11OpaqueValueES5_PKNS_14TargetMetadataINS_9InProcessEEE

- [+0x30] = getEnumTagSinglePayload
  - name="swift::metadataimpl::FixedSizeBufferValueWitnesses<swift::metadataimpl::ValueWitnesses< :metadataimpl::SwiftRetainableBox>, true, 8ul, 8ul, true>::getEnumTagSinglePayload(swift::OpaqueValue const*, unsigned int, swift::TargetMetadata< :InProcess> const*)"
    - mangled="_ZN5swift12metadataimpl29FixedSizeBufferValueWitnessesINS0_14ValueWitnessesINS0_18SwiftRetainableBoxEEELb1ELm8ELm8ELb1EE23getEnumTagSinglePayloadEPKNS_11OpaqueValueEjPKNS_14TargetMetadataINS_9InProcessEEE"
- [+0x38] = storeEnumTagSinglePayload
  - name="swift::metadataimpl::FixedSizeBufferValueWitnesses<swift::metadataimpl::ValueWitnesses< :metadataimpl::SwiftRetainableBox>, true, 8ul, 8ul, true>::storeEnumTagSinglePayload(swift::OpaqueValue*, unsigned int, unsigned int, swift::TargetMetadata< :InProcess> const*)"
    - mangled="_ZN5swift12metadataimpl29FixedSizeBufferValueWitnessesINS0_14ValueWitnessesINS0_18SwiftRetainableBoxEEELb1ELm8ELm8ELb1EE25storeEnumTagSinglePayloadEPNS_11OpaqueValueEjjPKNS_14TargetMetadataINS_9InProcessEEE"
- [+0x40] = int64 size
  - 0x0000000000000008
- [+0x48] = int64 stride
  - 0x0000000000000008
- [+0x50~0x53] = int32 flags
  - 0x00010007
- [+0x54~0x57] = int32 extraInhabitantCount
  - 0x7fffffff

StructMetadata

StructMetadata的内存布局

图

- Swift的StructMetadata内存布局图 = Swift ValueMetadata Memory Layout
  - 在线预览
    - Swift的StructMetadata内存布局图| ProcessOn免费在线作图,在线流程图,在线思维导图
  - 离线查看
    - 
  - 核心定义
    - 

文字

- Swift的StructMetadata的内存布局
  - [-0x08] = ValueWitnessTable* vwt;
  - [+0x00] = int64 kind // 是固定的：0x200
  - [+0x08] = StructDescriptor* description
    - StructDescriptor
      - FieldDescriptor

- FieldRecord

举例

`ConcreteConfigPrimitiveBox` 的 `StructDescriptor`

内存值：

```
(lldb) x/16gw 0x0000000010771431c
0x10771431c: 0x000000d1 0xfffffffdc 0xffe75bcc 0xfd6669b4
0x10771432c: 0x00000000 0x00000001 0x00000004 0x00a2c760
0x10771433c: 0xffe75bd4 0x00010001 0x00000002 0x00000080
0x10771434c: 0x00000080 0x000a5870 0xfffffe78 0x000000c2
```

对应着IDA中的汇编：

```
__constg_swiftt:0000000102DF831C ; nominal type descriptor for MainAppLibrary.ConcreteC
onfigPrimitiveBox
__constg_swiftt:0000000102DF831C                 s14MainAppLibrary26ConcreteConfigPrimitiveBoxVMn Stru
ctDescriptor <0xD1, stru_102DF82FC - ., aConcreteconfig - ., \
__constg_swiftt:0000000102DF831C                                         ; DATA XREF: t
ype metadata accessor for ConcreteConfigPrimitiveBox↑o
__constg_swiftt:0000000102DF831C                                         ; __swift5_typ
es:0000000102EA5EAC↓o ...
__constg_swiftt:0000000102DF831C                                 s14MainAppLibrary26
ConcreteConfigPrimitiveBoxVMa - .,\ ; type metadata accessor for ConcreteConfigPrimitiv
eBox ...
__constg_swiftt:0000000102DF831C                                 s14MainAppLibrary26
ConcreteConfigPrimitiveBoxVMn.FieldDescriptor - .,\
__constg_swiftt:0000000102DF831C                                 1, 4>
```

->

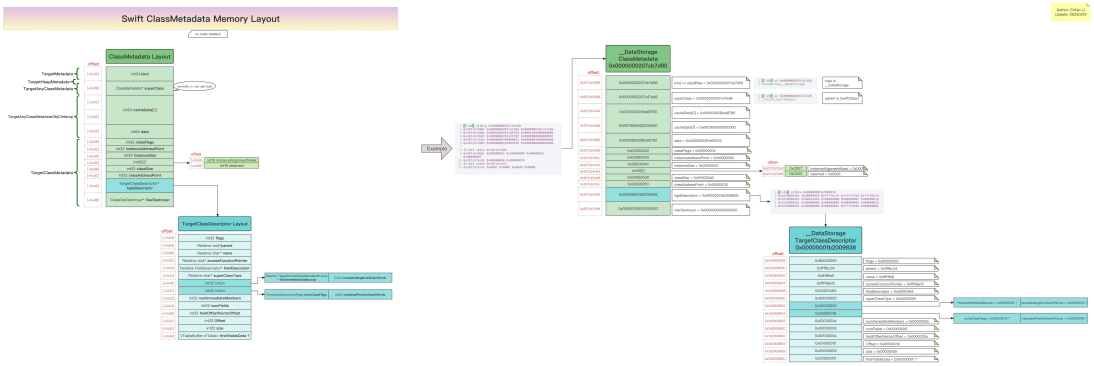
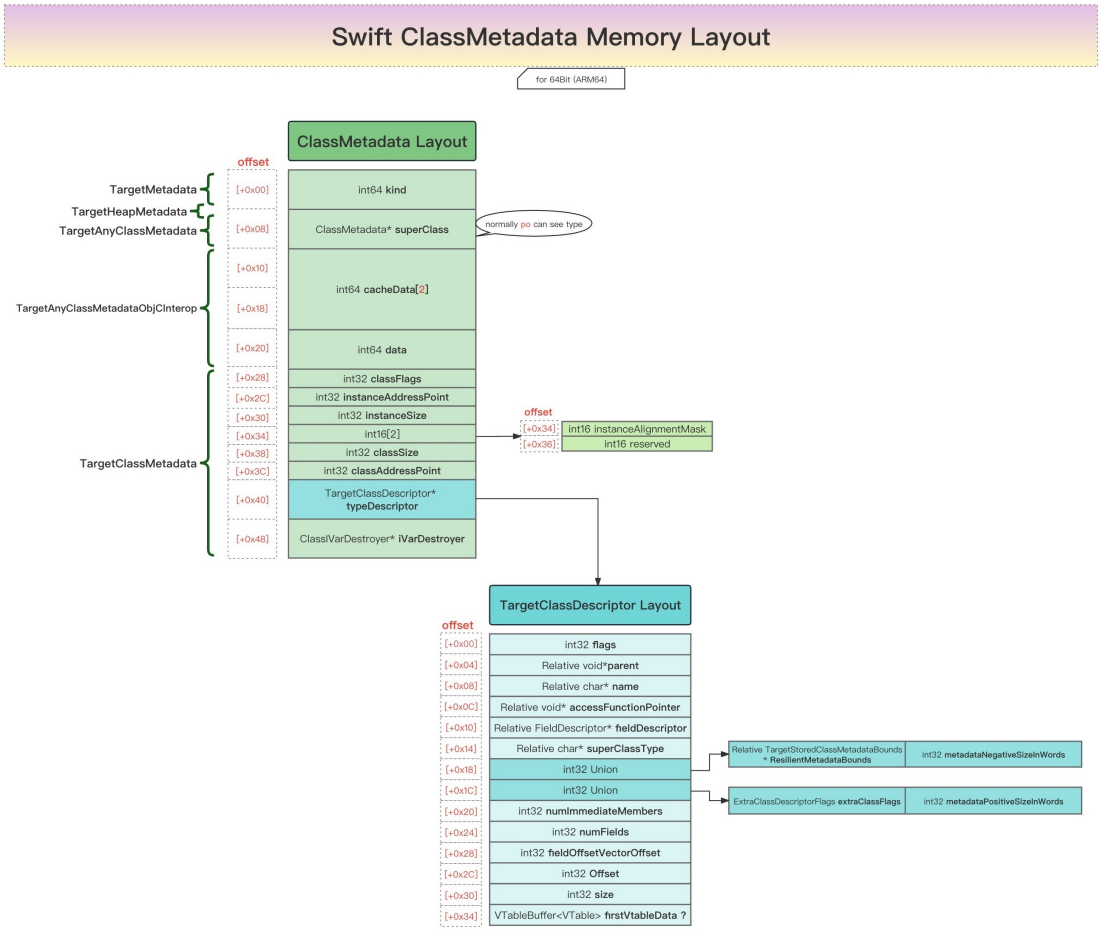
- StructDescriptor的例子：ConcreteConfigPrimitiveBox
  - struc : sizeof=0x1C == 0x0000000010771431c == IDA的 0x0000000102df831c
    - [0x00~0x03] = Flags
      - 0x000000d1
    - [0x04~0x07] = Parent
      - 0xfffffffdc
        - stru_102DF82FC - .
        - = 0x102DF82FC - (0x0000000102df831c + 0x4)
        - = 0x102DF82FC - 0x0000000102df8320
        - = 0xFFFFFFDC
    - [0x08~0x0B] = Name
      - 0xffe75bcc
        - aConcreteconfig - .
          - aConcreteconfig == "ConcreteConfigPrimitiveBox"
        - = 0x102C6DEF0 - (0x0000000102df831c + 0x8)
        - = 0xFFE75BCC

- [0x0C~0x0F] = AccessFunction = AccessFunctionPtr
  - 0xfd6669b4
    - $s14MainAppLibrary26ConcreteConfigPrimitiveBoxVMa - .
    - = 0x10045ECDC - (0x0000000102df831c + 0xC)
    - = 0xFD6669B4
- [0x10~0x13] = FieldDescriptor == Fields
  - 0x00000000
    - $s14MainAppLibrary26ConcreteConfigPrimitiveBoxVMn.FieldDescriptor - .
    - = 0x0000000102df832c - (0x0000000102df831c + 0x10)
    - = 0
- [0x14~0x17] = NumFields
  - 0x00000001
    - 这个结构体只有一个字段=属性property=域field？
    - 好像是：field offset vector length = 1
      - 此处是有1个field offset vector的
- [0x18~0x1B] = FieldOffsetVectorOffset
  - 0x00000004
    - The offset of the field offset vector for this struct's stored properties in its metadata
      - 此处的field offset vector的offset是4

ClassMetadata

内存布局

Swift的ClassMetadata的内存布局结构图

- Swift的ClassMetadata的内存布局结构图 = Swift ClassMetadata Memory Layout
  - 在线预览
    - Swift的ClassMetadata内存布局结构图| ProcessOn免费在线作图,在线流程图,在线思维导图
  - 离线查看
    - 
  - 核心定义
    - 

ClassMetadata的内存布局

概述

ClassMetadata = Class的TypeMetadata

- Class的TypeMetadata
  - [+0x00] = var kind: Int
  - [+0x08] = var superClass: UnsafeMutablePointer
  - [+0x10] = Int cacheData[2]
  - [+0x20] = var data: Int
  - [+0x28] = var classFlags: Int32
  - [+0x2C] = var instanceAddressPoint: UInt32
  - [+0x30] = var instanceSize: UInt32
  - [+0x34] = var instanceAlignmentMask: UInt16
  - [+0x36] = var reserved: UInt16
  - [+0x38] = var classSize: UInt32
  - [+0x3C] = var classAddressPoint: UInt32
  - [+0x40] = var typeDescriptor: UnsafeMutablePointer
  - [+0x48] = var iVarDestroyer: UnsafeMutablePointer

struct TargetClassDescriptor

- struct TargetClassDescriptor
  - [+0x00] = var flags: UInt32
  - [+0x04] = var parent: TargetRelativeDirectPointer
  - [+0x08] = var name: TargetRelativeDirectPointer
  - [+0x0C] = var accessFunctionPointer: TargetRelativeDirectPointer
  - [+0x10] = var fieldDescriptor: TargetRelativeDirectPointer
  - [+0x14] = var superClassType: TargetRelativeDirectPointer
  - [+0x18] = Int32 Union
    - var ResilientMetadataBounds: RelativeDirectPointer
    - var metadataNegativeSizeInWords: UInt32
  - [+0x1C] = Int32 Union
    - var extraClassFlags: ExtraClassDescriptorFlags
    - var metadataPositiveSizeInWords: UInt32
  - [+0x20] = var numImmediateMembers: UInt32
  - [+0x24] = var numFields: UInt32
  - [+0x28] = var fieldOffsetVectorOffset: UInt32
  - [+0x2C] = var Offset: UInt32
  - [+0x30] = var size: UInt32
  - [+0x34] = var firstVtableData: VTableBuffer

详解

Swift中类的继承关系

- Swift中类的继承关系
  - 详解
    - TargetClassMetadataType == ClassMetadata
      - = TargetClassMetadata>
    - TargetClassMetadataObjCInterop
      - = TargetClassMetadata>
    - ->
    - TargetAnyClassMetadataType
      - ObjCInterop=true == 兼容ObjC类 == 支持Objective-C类互操作
        - TargetAnyClassMetadataObjCInterop
      - ObjCInterop=false == 不兼容ObjC类
        - TargetAnyClassMetadata
    - TargetAnyClassMetadataObjCInterop
      - TargetAnyClassMetadata
    - TargetClassMetadata
      - TargetAnyClassMetadataVariant
    - ->
    - TargetAnyClassMetadata
      - TargetHeapMetadata == HeapMetadata
        - TargetMetadata
  - 概述
    - ClassMetadata
      - TargetClassMetadata
        - TargetAnyClassMetadataObjCInterop
          - TargetAnyClassMetadata
            - TargetHeapMetadata
              - TargetMetadata

Swift中的：ClassMetadata 字段定义

- Swift中的：ClassMetadata 字段定义
  - TargetMetadata
    - var kind: Int
      - 在oc中放的就是isa，在swift中kind大于0x7FF表示的就是类
  - TargetHeapMetadata
    - 没属性=字段
  - TargetAnyClassMetadata
    - var superClass: UnsafeMutablePointer
      - 父类的Metadata，如果是null说明是最顶级的root类了
  - TargetAnyClassMetadataObjCInterop
    - Int cacheData[2]
      - 缓存数据用于某些动态查找，它由运行时拥有，通常需要与Objective-C的使用进行互操

作。（说到底就是OC的东西）
        - var data: Int
            - 除了编译器设置低位以表明这是Swift元类型（因此存在对应的类型元数据的头信息）外，这个data里存的指针，用于行外元数据，通常是不透明的（应该也是OC的）
    - TargetClassMetadata
        - var classFlags: Int32
            - Swift-specific class flags
        - var instanceAddressPoint: UInt32
            - The address point of instances of this type
        - var instanceSize: UInt32
            - The required size of instances of this type.(实例对象在堆内存的大小)
        - var instanceAlignmentMask: UInt16
            - The alignment mask of the address point of instances of this type. (根据这个mask来获取内存中的对齐大小)
        - var reserved: UInt16
            - Reserved for runtime use.（预留给运行时使用）
        - var classSize: UInt32
            - The total size of the class object, including prefix and suffix extents.
        - var classAddressPoint: UInt32
            - The offset of the address point within the class object.
        - var typeDescriptor: UnsafeMutablePointer
            - 一个对类型的超行的swift特定描述，如果这是一个人工子类，则为null。目前不提供动态创建非人工子类的机制。
        - var iVarDestroyer: UnsafeMutablePointer
            - 销毁实例变量的函数，用于在构造函数早期返回后进行清理。如果为null，则不会执行清理操作，并且所有的ivars都必须是简单的。

struct TargetClassDescriptor

- struct TargetClassDescriptor
    - var flags: UInt32
        - 存储在任何上下文描述符的第一个公共标记
    - var parent: TargetRelativeDirectPointer
        - 复用的RelativeDirectPointer这个类型，其实并不是，但看下来原理一样;
        - 父级上下文，如果是顶级上下文则为null
    - var name: TargetRelativeDirectPointer
        - 获取类的名称
    - var accessFunctionPointer: TargetRelativeDirectPointer
        - 这里的函数类型是一个替身，需要调用getAccessFunction()拿到真正的函数指针（这里没有封装），会得到一个MetadataAccessFunction元数据访问函数的指针的包装器类，该函数提供operator()重载以使用正确的调用约定来调用它（可变长参数），意外发现命名重整会调用这边的方法（目前不太了解这块内容）。
    - var fieldDescriptor: TargetRelativeDirectPointer
        - 一个指向类型的字段描述符的指针(如果有的话)。类型字段的描述，可以从里面获取结构体的属性。
    - var superClassType: TargetRelativeDirectPointer

        - The type of the superclass, expressed as a mangled type name that can refer to the generic arguments of the subclass type.
    - Int32 Union (下面两个属性在源码中是union类型，所以取size大的类型作为属性（这里貌似一样），具体还得判断是否have a resilient superclass)
        - var ResilientMetadataBounds: RelativeDirectPointer
            - 有resilient superclass，用ResilientMetadataBounds，表示对保存元数据扩展的缓存的引用
        - var metadataNegativeSizeInWords: UInt32
            - 没有resilient superclass使用MetadataNegativeSizeInWords，表示该类元数据对象的负大小(用字节表示)
    - Int32 Union
        - var extraClassFlags: ExtraClassDescriptorFlags
            - 有resilient superclass，用ExtraClassFlags，表示一个Objective-C弹性类存根的存在
        - var metadataPositiveSizeInWords: UInt32
            - 没有resilient superclass使用MetadataPositiveSizeInWords，表示该类元数据对象的正大小(用字节表示)
    - var numImmediateMembers: UInt32
        - 此类添加到类元数据的其他成员的数目。默认情况下，这些数据对运行时是不透明的，而不是在其他成员中公开;它实际上只是NumImmediateMembers *sizeof(void)*字节的数据。
        - 这些字节是添加在地址点之前还是之后，取决于areImmediateMembersNegative()方法。
    - var numFields: UInt32
        - 属性个数，不包含父类的
    - var fieldOffsetVectorOffset: UInt32
        - 存储这个结构的字段偏移向量的偏移量（记录你属性起始位置的开始的一个相对于metadata的偏移量，具体看metadata的getFieldOffsets方法），如果为0，说明你没有属性
        - 如果这个类含有一个弹性的父类，那么从他的弹性父类的metaData开始偏移
    - var Offset: UInt32
    - var size: UInt32 //VTable数量
    - var firstVtableData: VTableBuffer //VTable

Swift源码

IDA定义

```
struct ClassMetadata
{
  __int64 kind;
  void *superClass;
  __int64 cacheData[2];
  void *data;
  __int32 classFlags;
  __int32 instanceAddressPoint;
  __int32 instanceSize;
  __int16 instanceAlignmentMask;
  __int16 reserved;
  __int32 classSize;
  __int32 classAddressPoint;
  void *typeDescriptor;
  void *iVarDestroyer;
};
```

举例

__DataStorage

具体详见：

ClassMetadata 内存布局 图

对应调试细节：

ClassMetadata

```
(lldb) x/10xg 0x0000000207cb7d90
0x207cb7d90: 0x0000000207cb7d58 0x0000000207c47ed8
0x207cb7da0: 0x00000001c29c8490 0x0000805000000000
0x207cb7db0: 0x0000000281e95702 0x0000000000000002
0x207cb7dc0: 0x0000000700000041 0x00000010000000d0
0x207cb7dd0: 0x00000001b2009838 0x0000000000000000

(lldb) po 0x0000000207cb7d58
Foundation.__DataStorage
(lldb) po 0x0000000207c47ed8
_TtCs12_SwiftObject

(lldb) x/2xh 0x207cb7db8
0x207cb7db8: 0x0002 0x0000

(lldb) x/2xh 0x207cb7dc0
0x207cb7dc0: 0x0041 0x0000

(lldb) x/2xw 0x207cb7db8
0x207cb7db8: 0x00000002 0x00000000

(lldb) x/2xw 0x207cb7dc0
0x207cb7dc0: 0x00000041 0x00000007

(lldb) x/2xh 0x207cb7dc4
0x207cb7dc4: 0x0007 0x0000

(lldb) x/2xw 0x207cb7dc8
0x207cb7dc8: 0x000000d0 0x00000010
```

->

- __DataStorage的Class的TypeMetadata
  - [+0x00] = var kind: Int
    - 0x0000000207cb7d58
      - Foundation.__DataStorage
  - [+0x08] = var superClass: UnsafeMutablePointer
    - 0x0000000207c47ed8

        - _TtCs12_SwiftObject
    - [+0x10] = Int cacheData[2]
        - 0x00000001c29c8490
        - 0x0000805000000000
    - [+0x20] = var data: Int
        - 0x0000000281e95702
    - [+0x28] = var classFlags: Int32
        - 0x00000002
    - [+0x2C] = var instanceAddressPoint: UInt32
        - 0x00000000
    - [+0x30] = var instanceSize: UInt32
        - 0x00000041
    - [+0x34] = var instanceAlignmentMask: UInt16
        - 0x0007
    - [+0x36] = var reserved: UInt16
        - 0x0000
    - [+0x38] = var classSize: UInt32
        - 0x000000d0
    - [+0x3C] = var classAddressPoint: UInt32
        - 0x00000010
    - [+0x40] = var typeDescriptor: UnsafeMutablePointer
        - 0x00000001b2009838
    - [+0x48] = var iVarDestroyer: UnsafeMutablePointer
        - 0x0000000000000000

以及：

struct TargetClassDescriptor

```
(lldb) x 8xg 0x00000001b2009838
0x1b2009838: 0xffffbc3480000050 0xfff49e10ffffffe8
0x1b2009848: 0x0000000000010464 0x0000001800000002
0x1b2009858: 0x000000060000000e 0x000000100000000a
0x1b2009868: 0x0000000100000008 0x00000001fff33690
(lldb) x 16xw 0x00000001b2009838
0x1b2009838: 0x80000050 0xffffbc34 0xffffffe8 0xfff49e10
0x1b2009848: 0x00010464 0x00000000 0x00000002 0x00000018
0x1b2009858: 0x0000000e 0x00000006 0x0000000a 0x00000010
0x1b2009868: 0x00000008 0x00000001 0xfff33690 0x00000001
```

->

- Foundation.__DataStorage TypeDescriptor == struct TargetClassDescriptor
    - [+0x00] = var flags: UInt32
        - 0x80000050
    - [+0x04] = var parent: TargetRelativeDirectPointer
        - 0xffffbc34
    - [+0x08] = var name: TargetRelativeDirectPointer

        - 0xffffffe8
    - [+0x0C] = var accessFunctionPointer: TargetRelativeDirectPointer
        - 0xfff49e10
    - [+0x10] = var fieldDescriptor: TargetRelativeDirectPointer
        - 0x00010464
    - [+0x14] = var superClassType: TargetRelativeDirectPointer
        - 0x00000000
    - [+0x18] = Int32 Union
        - var ResilientMetadataBounds: RelativeDirectPointer
        - var metadataNegativeSizeInWords: UInt32
            - 估计是：0x00000002
    - [+0x1C] = Int32 Union
        - var extraClassFlags: ExtraClassDescriptorFlags
        - var metadataPositiveSizeInWords: UInt32
        - 暂不确定是哪个
        - 此处值：0x00000018
    - [+0x20] = var numImmediateMembers: UInt32
        - 0x0000000e
    - [+0x24] = var numFields: UInt32
        - 0x00000006
    - [+0x28] = var fieldOffsetVectorOffset: UInt32
        - 0x0000000a
    - [+0x2C] = var Offset: UInt32
        - 0x00000010
    - [+0x30] = var size: UInt32
        - 0x00000008
    - [+0x34] = var firstVtableData: VTableBuffer
        - 0x00000001

MetadataKind = Metadata的type

Swift中Metadata的Type == MetadataKind 的定义

图

- MetadataKind的定义 表格
  -

MetadataKind		
Value	**Name**	**Explanation**
0x0	Class	类
0x200	Struct	结构体
0x201	Enum	枚举
0x202	Optional	可选类型
0x203	ForeignClass	外部类 比如CoreFoundation中的类
0x300	Opaque	不透明类型 在元数据系统中不公开其值的类型
0x301	Tuple	元祖类型
0x302	Function	单型函数
0x303	Existential	存在类型
0x304	Metatype	元类型?
0x305	ObjCClassWrapper	ObjC类的包装器
0x306	ExistentialMetatype	存在类型的元类型?
0x400	HeapLocalVariable	使用静态生成的元数据的堆分配的局部变量
0x500	HeapGenericLocalVariable	使用运行时实例化的元数据的堆分配的局部变量
0x501	ErrorObject	swift原生的错误类型
Related Definition		
0x100	MetadataKindIsRuntimePrivate	是运行时私有的
0x200	MetadataKindIsNonHeap	不是堆的
0x400	MetadataKindIsNonType	不是类型的
0x7FF	LastEnumerated	

- == Swift的ValueMetadata和VWT的内存布局图中的 `MetadataKind` 的表格

核心代码

```
const unsigned MetadataKindIsNonType = 0x400;
const unsigned MetadataKindIsNonHeap = 0x200;
const unsigned MetadataKindIsRuntimePrivate = 0x100;

LastEnumerated = 0x7FF,

NOMINALTYPEMETADATAKIND(Class, 0)
NOMINALTYPEMETADATAKIND(Struct, 0 | MetadataKindIsNonHeap)
NOMINALTYPEMETADATAKIND(Enum, 1 | MetadataKindIsNonHeap)
NOMINALTYPEMETADATAKIND(Optional, 2 | MetadataKindIsNonHeap)
METADATAKIND(ForeignClass, 3 | MetadataKindIsNonHeap)
METADATAKIND(Opaque, 0 | MetadataKindIsRuntimePrivate | MetadataKindIsNonHeap)
METADATAKIND(Tuple, 1 | MetadataKindIsRuntimePrivate | MetadataKindIsNonHeap)
METADATAKIND(Function, 2 | MetadataKindIsRuntimePrivate | MetadataKindIsNonHeap)
METADATAKIND(Existential, 3 | MetadataKindIsRuntimePrivate | MetadataKindIsNonHeap)
METADATAKIND(Metatype, 4 | MetadataKindIsRuntimePrivate | MetadataKindIsNonHeap)
METADATAKIND(ObjCClassWrapper, 5 | MetadataKindIsRuntimePrivate | MetadataKindIsNonHeap)

METADATAKIND(ExistentialMetatype, 6 | MetadataKindIsRuntimePrivate | MetadataKindIsNonH
eap)
METADATAKIND(HeapLocalVariable, 0 | MetadataKindIsNonType)
METADATAKIND(HeapGenericLocalVariable, 0 | MetadataKindIsNonType | MetadataKindIsRuntim
ePrivate)
METADATAKIND(ErrorObject, 1 | MetadataKindIsNonType | MetadataKindIsRuntimePrivate)
```

表格

名称	枚举值	说明
Class	0x0	类
Struct	0x200	结构体
Enum	0x201	枚举
Optional	0x202	可选类型
ForeignClass	0x203	外部类，比如CoreFoundation中的类
Opaque	0x300	在元数据系统中不公开其值的类型
Tuple	0x301	元祖类型
Function	0x302	A monomorphic function
Existential	0x303	An existential type
Metatype	0x304	A metatype
ObjCClassWrapper	0x305	An ObjC class wrapper
ExistentialMetatype	0x306	An existential metatype

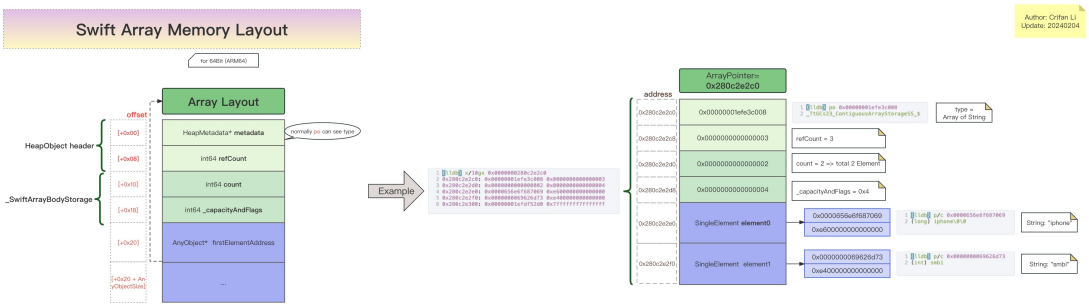
HeapLocalVariable	0x400	使用静态生成的元数据的堆分配的局部变量
HeapGenericLocalVariable	0x500	使用运行时实例化的元数据的堆分配的局部变量
ErrorObject	0x501	swift原生的错误类型
LastEnumerated	0x7FF	最大的非isa指针元数据类型值
下面是通用全局定义		
MetadataKindIsNonType	0x400	
MetadataKindIsNonHeap	0x200	
MetadataKindIsNonHeap	0x200	
LastEnumerated	0x7FF	

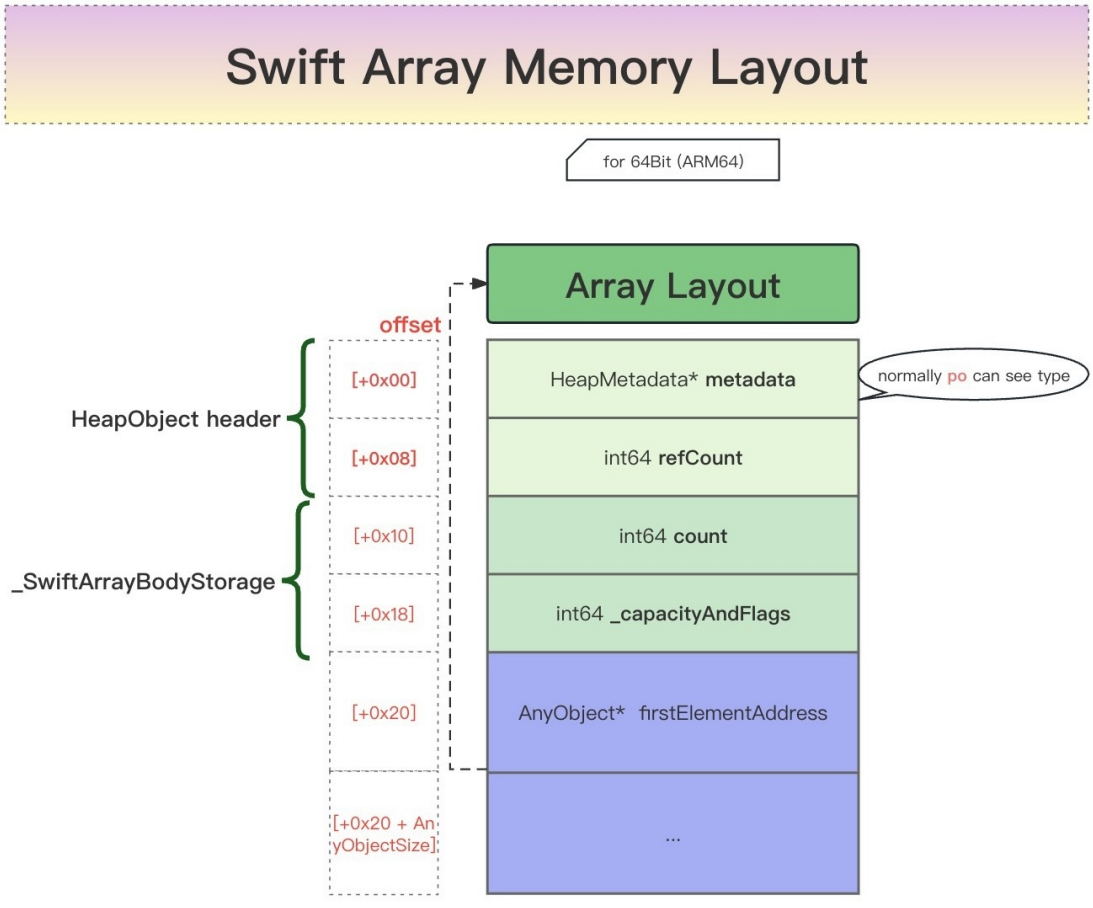
常用类型

Array数组

内存布局

Swift的Array的内存布局图

- Swift的Array的内存布局图 = Swift Array Memory Layout
  - 在线预览
    - Swift的Array内存布局结构图| ProcessOn免费在线作图,在线流程图,在线思维导图
  - 离线查看
    - 

  - 核心内容
    - 

文字

- Array的内存布局=字段=属性
  - [+0x00] = HeapMetadata* metadata
  - [+0x08] = int64 refCount
  - [+0x10] = int64 count
  - [+0x08] = int64 _capacityAndFlags
  - [+0x20] = AnyObject* firstElementAddress

Swift源码

IDA定义

举例

String的Array

```
(lldb) x 10gx 0x0000000280c2e2c0
0x280c2e2c0: 0x00000001efe3c008 0x0000000000000003
0x280c2e2d0: 0x0000000000000002 0x0000000000000004
0x280c2e2e0: 0x0000656e6f687069 0xe600000000000000
0x280c2e2f0: 0x0000000069626d73 0xe400000000000000
0x280c2e300: 0x00000001efdf52d0 0x7fffffff7fffffff
(lldb) po 0x00000001efe3c008
_TtGCs23_ContiguousArrayStorageSS_$
(lldb) p/c 0x0000656e6f687069
(long) iphone\0\0
(lldb) p/c 0x0000000069626d73
(int) smbi
```

->

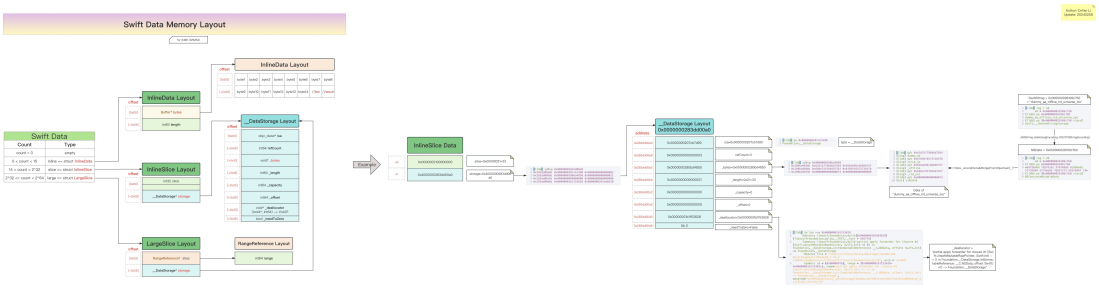
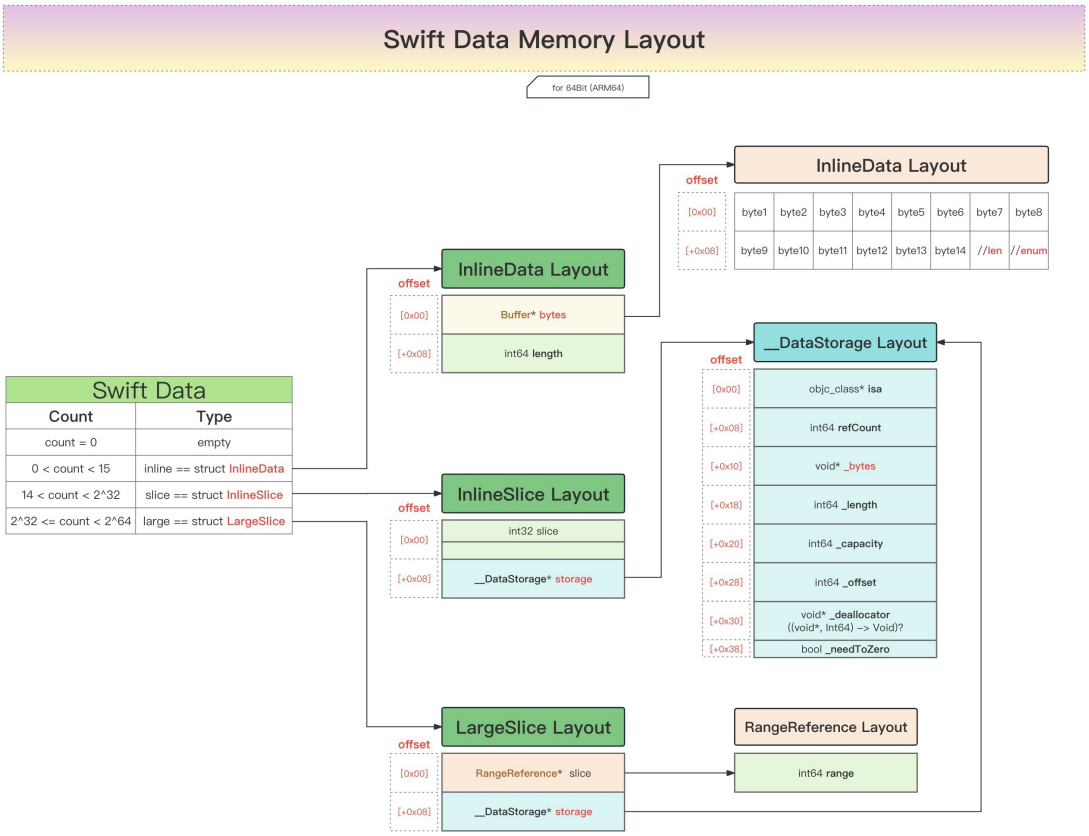
- Array的内存布局=字段
  - HeapMetadata* metadata
    - 0x00000001efe3c008
      - `_TtGCs23_ContiguousArrayStorageSS_$`
        - String的Array
  - int64 refCount
    - 0x0000000000000003
  - int64 count
    - 0x0000000000000002
  - int64 _capacityAndFlags
    - 0x0000000000000004
  - 第一个 = [0x280c2e2e0]
    - 0x0000656e6f687069 0xe600000000000000
      - "iphone"
  - 第二个 = [0x280c2e2f0]
    - 0x0000000069626d73 0xe400000000000000
      - "smbi"

Bool布尔

Struct结构体

Data数据的内存布局

Swift的Data数据的内存布局结构图

- Swift的Data数据的内存布局结构图 = Swift Data Memory Layout
  - 在线预览
    - Swift的Data内存布局结构图| ProcessOn免费在线作图,在线流程图,在线思维导图
  - 离线查看
    - 
  - 核心内容
    - 

文字

概述

- Swift的Data（对于64位，暂忽略32位） 的内存布局
  - 根据数据字节个数多少分类
    - `count=0`
      - empty = 空数据
    - `0 < count < 15` == `0 < count <= 14`
      - `inline` == `struct InlineData`
        - `var bytes: Buffer`
          - `@usableFromInline typealias Buffer = (UInt8, UInt8, UInt8, UInt8, UInt8, UInt8, UInt8, UInt8, UInt8, UInt8, UInt8, UInt8, UInt8, UInt8, UInt8) //len //enum`
        - `var length: UInt8`
    - `14 < count < 2^32`
      - `slice` == `struct InlineSlice`
        - `var slice: Range<Int32>`
        - `var storage: __DataStorage`
    - `2^32 <= count < 2^64`
      - `large` == `struct LargeSlice`
        - `var slice: RangeReference`
          - `var range: Range<Int64>`
        - `var storage: __DataStorage`

`__DataStorage`

```
class __DataStorage {
    var isa: objc_class? // NULL for pure Swift class
    var refCount: UInt64
    var _bytes: UnsafeMutableRawPointer?
    var _length: UInt64
    var _capacity: UInt64
    var _offset: UInt64
    var _deallocator: ((UnsafeMutableRawPointer, Int64) -> Void)?
    var _needToZero: Bool
}
```

==

- `class __DataStorage`
  - [+0x00] = objc_class* isa
  - [+0x08] = int64 refCount
  - [+0x10] = void* _bytes
  - [+0x18] = int64 _length

    - [+0x20] = int64 _capacity
    - [+0x28] = int64 _offset
    - [+0x30] = function_pointer* _deallocator
        - 函数定义：`((UnsafeMutableRawPointer, Int64) -> Void)?`
    - [+0x38] = Bool _needToZero

详解

- Swift的Data
    - 分4类
        - 空数据：empty
        - inline == struct InlineData
            - 字段
                - var bytes: Buffer
                - var length: UInt8
            - 说明
                - bytes的Buffer的字节=个数
                    - 64位架构：不超过14，即 <=14个字节
                        - @usableFromInline typealias Buffer = (UInt8, UInt8, UInt8, UInt8, UInt8, UInt8, UInt8, UInt8, UInt8, UInt8, UInt8, UInt8, UInt8, UInt8, UInt8) //len //enum
                    - 32位架构：不超过6，即 <=6个字节
                        - @usableFromInline typealias Buffer = (UInt8, UInt8, UInt8, UInt8, UInt8, UInt8) //len //enum
        - slice == struct InlineSlice
            - 字段
                - var slice: Range
                - var storage: __DataStorage
            - 说明
                - InlineData的（64bit的）14个 < InlineSlice的字节个数 < 2^64的最大值？（storage pointer + range == a signle word）
        - large == struct LargeSlice
            - 字段
                - var slice: RangeReference
                    - var range: Range
                - var storage: __DataStorage
            - 说明
                - a single word个数=2^64？ < LargeSlice的字节个数 < 双字节大小（two-word size）=2^128?

Swift源码

IDA定义

- struct `Swift_DataStorage`

```
struct Swift_DataStorage
{
  void *isa;
  __int64 refCount;
  void *_bytes;
  __int64 _length;
  __int64 _capacity;
  __int64 _offset;
  void *_deallocator;
  bool _needToZero;
};
```

- struct `SwiftData_InlineSlice`

```
struct SwiftData_InlineSlice
{
  __int32 slice;
  Swift_DataStorage *storage;
};
```

举例

__DataStorage

详见：

Data数据 内存布局 图

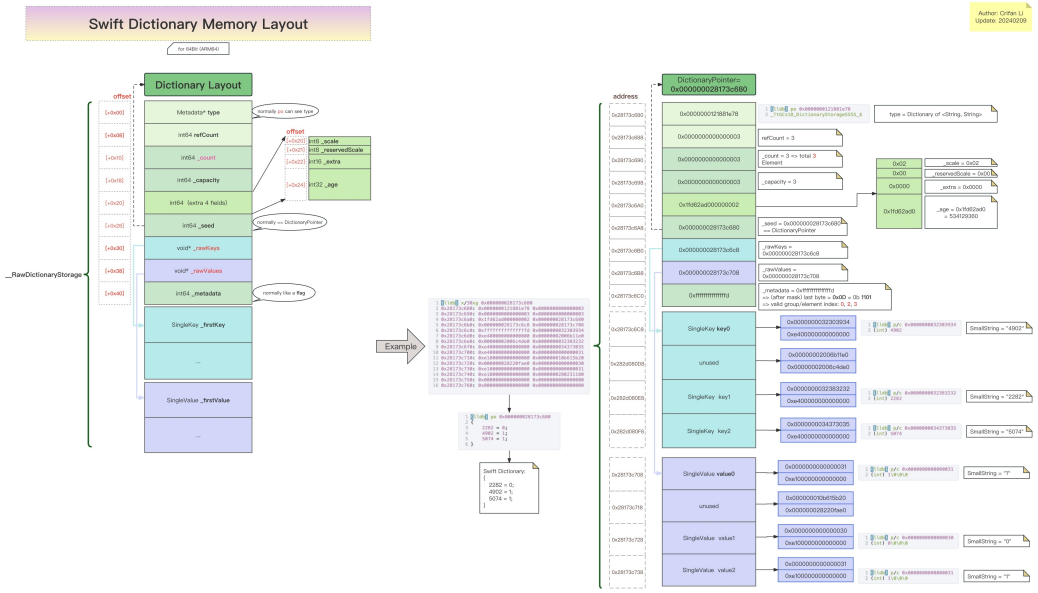
Dictionary字典

最后更新：2024-02-22 09:35:12

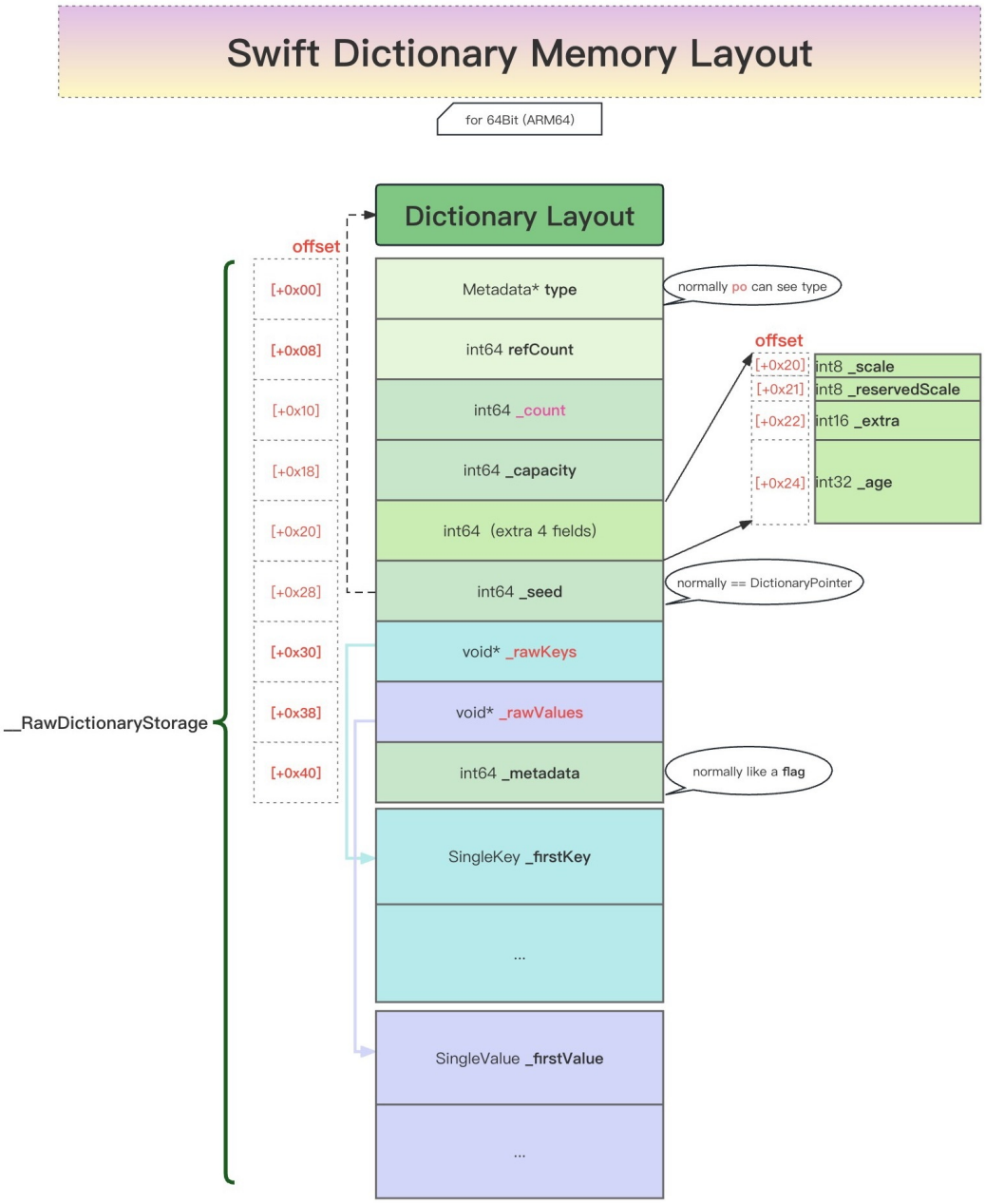
Swift的Dictionary内存布局

Swift的Dictionary内存布局结构图

- Swift的Dictionary内存布局结构图
  - 在线浏览
    - Swift的Dictionary内存布局结构图| ProcessOn免费在线作图,在线流程图,在线思维导图
  - 本地查看
    - 完整图



    - 核心内容



Swift的Dictionary内存布局 文字

- Swift的Dictionary内存布局 = 字段 = 属性
  - `[+0x00]` = `Metadata* type`
  - `[+0x08]` = `Int refCount`
  - `__RawDictionaryStorage`
    - `[+0x10]` = `var _count: Int`
    - `[+0x18]` = `var _capacity: Int`
    - `[+0x20]` = `Int`
      - `[+0x20~0x20]` = `var _scale: Int8`
      - `[+0x21~0x21]` = `var _reservedScale: Int8`
      - `[+0x22~0x23]` = `var _extra: Int16`
      - `[+0x24~0x27]` = `var _age: Int32`
    - `[+0x28]` = `var _seed: Int`
    - `[+0x30]` = `var _rawKeys: UnsafeMutableRawPointer`
      - 指向 `Key` = `键` 的数组列表
    - `[+0x38]` = `var _rawValues: UnsafeMutableRawPointer`
      - 指向 `Value` = `值` 的数组列表
    - `[+0x38]` = `int _metadata`
      - 指定了**哪几个**index元素是**有效数据**

Swift源码

IDA定义

举例

举例1：简单dict

Swift的Dict值：

```
{
    2282 = 0;
    4902 = 1;
    5074 = 1;
}
```

对应的例子，已经放到了内存布局图中了，详见：

Dictionary字典的内存布局图

此处只额外贴出：

相关调试内容

```
(lldb) po 0x000000028173c680
{
    2282 = 0;
    4902 = 1;
    5074 = 1;
}
```

->

```
(lldb) x/8xg 0x000000028173c680
0x28173c680: 0x00000001218 81e78 0x0000000000000003
0x28173c690: 0x0000000000000003 0x0000000000000003
0x28173c6a0: 0x1fd62ad000000002 0x000000028173c680
0x28173c6b0: 0x000000028173c6c8 0x000000028173c708
(lldb) x/20xg 0x000000028173c680
0x28173c680: 0x00000001218 81e78 0x0000000000000003
0x28173c690: 0x0000000000000003 0x0000000000000003
0x28173c6a0: 0x1fd62ad000000002 0x000000028173c680
0x28173c6b0: 0x000000028173c6c8 0x000000028173c708
0x28173c6c0: 0xfffffffffffffffd 0x0000000032303934
0x28173c6d0: 0xe400000000000000 0x00000002006b11e0
0x28173c6e0: 0x00000002006c4de0 0x0000000032383232
0x28173c6f0: 0xe400000000000000 0x0000000034373035
0x28173c700: 0xe400000000000000 0x0000000000000031
0x28173c710: 0xe100000000000000 0x000000010b615b20
```

可以继续了：

- 0xd = 0b 1101

- 有效index
  - 0
  - 2
  - 3

->

```
(lldb) x/40xg 0x000000028173c680
0x28173c680: 0x00000001218881e78 0x0000000000000003
0x28173c690: 0x0000000000000003 0x0000000000000003
0x28173c6a0: 0x1fd62ad000000002 0x000000028173c680
0x28173c6b0: 0x000000028173c6c8 0x000000028173c708
0x28173c6c0: 0xfffffffffffffffd 0x0000000032303934
0x28173c6d0: 0xe400000000000000 0x00000002006b11e0
0x28173c6e0: 0x00000002006c4de0 0x0000000032383232
0x28173c6f0: 0xe400000000000000 0x0000000034373035
0x28173c700: 0xe400000000000000 0x0000000000000031
0x28173c710: 0xe100000000000000 0x000000010b615b20
0x28173c720: 0x000000028220fae0 0x0000000000000030
0x28173c730: 0xe100000000000000 0x0000000000000031
0x28173c740: 0xe100000000000000 0x0000000280231180
0x28173c750: 0x0000000000000000 0x0000000000000000
0x28173c760: 0x0000000000000000 0x0000000000000000
0x28173c770: 0x0000000000000000 0x0000000000000000
0x28173c780: 0x0000000000000000 0x0000000000000000
0x28173c790: 0x0000000000000000 0x0000000000000000
0x28173c7a0: 0x0000000000000000 0x0000000000000000
0x28173c7b0: 0x0000000000000000 0x0000000000000000
(lldb)

(lldb) x/30xg 0x000000028173c680
0x28173c680: 0x00000001218881e78 0x0000000000000003
0x28173c690: 0x0000000000000003 0x0000000000000003
0x28173c6a0: 0x1fd62ad000000002 0x000000028173c680
0x28173c6b0: 0x000000028173c6c8 0x000000028173c708
0x28173c6c0: 0xfffffffffffffffd 0x0000000032303934
0x28173c6d0: 0xe400000000000000 0x00000002006b11e0
0x28173c6e0: 0x00000002006c4de0 0x0000000032383232
0x28173c6f0: 0xe400000000000000 0x0000000034373035
0x28173c700: 0xe400000000000000 0x0000000000000031
0x28173c710: 0xe100000000000000 0x000000010b615b20
0x28173c720: 0x000000028220fae0 0x0000000000000030
0x28173c730: 0xe100000000000000 0x0000000000000031
0x28173c740: 0xe100000000000000 0x0000000280231180
0x28173c750: 0x0000000000000000 0x0000000000000000
0x28173c760: 0x0000000000000000 0x0000000000000000
```

->

```
(lldb) po 0x00000001218881e78
_TtGCs18_DictionaryStorageSSSS_$

(lldb) p c 0x0000000032303934
(int) 4902
```

```
(lldb) p c 0x0000000032383232
(int) 2282
(lldb) p c 0x0000000034373035
(int) 5074
(lldb) p c 0x0000000000000031
(int) 1\0\0\0
(lldb) p c 0x0000000000000030
(int) 0\0\0\0
(lldb) x/2xw 0x28173c6a0
0x28173c6a0: 0x00000002 0x1fd62ad0
(lldb) x/4xh 0x28173c6a0
0x28173c6a0: 0x0002 0x0000 0x2ad0 0x1fd6
(lldb) x/8xb 0x28173c6a0
0x28173c6a0: 0x02 0x00 0x00 0x00 0xd0 0x2a 0xd6 0x1f
(lldb) p/d 0x1fd62ad0
(int) 534129360
```

举例2：复杂dict - key是String，value是AnyPrimitive

调试到复杂dict：

```
(lldb) reg r x0 x1 x2 x8
      x0 = 0x0000000011100b200
      x1 = 0x00000001031145b0  WhatsApp`$s10LeafFilterVN
      x2 = 0x0000000103114658  WhatsApp`LeafFilter_related_1030B4658
      x8 = 0x00000001006409a8  WhatsApp`sub_1005E09A8
(lldb) po 0x0000000011100b200
4580225536
(lldb) x/6xg 0x0000000011100b200
0x11100b200: 0x000000010fe175c8 0x0000000000000003
0x11100b210: 0x0000000000000007 0x000000000000000c
0x11100b220: 0xf3473a6200000004 0x0000000011100b200
(lldb) po 0x000000010fe175c8
_TtGCs18_DictionaryStorageSSOVO14MainAppLibrary9OfflineAB8UserInfoP10$102e672c012AnyPri
mitive_$
```

->

此处是个Dictionary，value元素是

- AnyPrimitive
  - MainAppLibrary.OfflineAB.UserInfo 的 AnyPrimitive

查看内存值：

```
(lldb) x/80xg 0x0000000011100b200
0x11100b200: 0x000000010fe175c8 0x0000000000000003
0x11100b210: 0x0000000000000007 0x000000000000000c
0x11100b220: 0xf3473a6200000004 0x0000000011100b200
0x11100b230: 0x0000000011100b248 0x0000000011100b348
0x11100b240: 0xffffffffffff9299 0x6d726f6674616c70
0x11100b250: 0xe800000000000000 0x00000001e84fab78
0x11100b260: 0x00000001e84fab78 0x00000001e84fab78
```

```
0x11100b270: 0x00000001e84fab78 0x695f656369766564
0x11100b280: 0xe900000000000064 0x69737265765f736f
0x11100b290: 0xea00000000006e6f 0x00000001e84fab78
0x11100b2a0: 0x00000001e84fab78 0x00000001e84fab78
0x11100b2b0: 0x00000001e84fab78 0x6c6975625f707061
0x11100b2c0: 0xe900000000000064 0x00000001e84fab78
0x11100b2d0: 0x00000001e84fab78 0x6e5f656369766564
0x11100b2e0: 0xeb00000000656d61 0x00000001e84fab78
0x11100b2f0: 0x00000001e84fab78 0x00000001e84fab78
0x11100b300: 0x00000001e84fab78 0x5f657361656c6572
0x11100b310: 0xef6c656e6e616863 0x00000001e84fab78
0x11100b320: 0x00000001e84fab78 0x00000001e84fab78
0x11100b330: 0x00000001e84fab78 0x737265765f707061
0x11100b340: 0xeb000000006e6f69 0x0000656e6f687069
0x11100b350: 0xe600000000000000 0x000000010fe17468
0x11100b360: 0x000000010fe16950 0x000000010fe169f8
0x11100b370: 0x00000001e84fab01 0x00000001e84fab78
0x11100b380: 0x00000001e84fab78 0x00000001e84fab78
0x11100b390: 0x00000001e84fab78 0x00000001e84fab78
0x11100b3a0: 0x00000001e84fab78 0x00000001e84fab78
0x11100b3b0: 0x00000001e84fab78 0x00000001e84fab78
0x11100b3c0: 0x00000001e84fab78 0x00000001e84fab78
0x11100b3d0: 0x00000001e84fab78 0xc000000000000024
0x11100b3e0: 0x40000002820700c0 0x00000002820700c0
0x11100b3f0: 0x000000010fe16950 0x000000010fe169f8
0x11100b400: 0x00000001e84fab01 0x000000014164cccd
0x11100b410: 0xc00000000000000a 0x40000002835413a0
0x11100b420: 0x000000010fe17798 0x000000010fe17840
0x11100b430: 0x00000001e84fab00 0x00000001e84fab78
0x11100b440: 0x00000001e84fab78 0x00000001e84fab78
0x11100b450: 0x00000001e84fab78 0x00000001e84fab78
0x11100b460: 0x00000001e84fab78 0x00000001e84fab78
0x11100b470: 0x00000001e84fab78 0x00000001e84fab78
```

->

```
(lldb) p c 0x6d726f6674616c70
(long) platform

(lldb) p c 0x695f656369766564
(long) device_i
(lldb) p c 0xe900000000000064
(unsigned long) d\0\0\0\0\0\0\xe9

(lldb) p c 0x69737265765f736f
(long) os_versi
(lldb) p c 0xea00000000006e6f
(unsigned long) on\0\0\0\0\0\xea

(lldb) p c 0x6c6975625f707061
(long) app_buil
(lldb) p c 0x64
(int) d\0\0\0

(lldb) p c 0x6e5f656369766564
```

```
(long) device_n
(lldb) p c 0x00000000656d61
(int) ame\0

(lldb) p c 0x5f657361656c6572
(long) release_
(lldb) p c 0xef6c656e6e616863
(unsigned long) channel\xef

(lldb) p c 0x737265765f707061
(long) app_vers
(lldb) p c 0xeb000000006e6f69
(unsigned long) ion\0\0\0\0\xeb
```

keys:

- platform
- device_id
- os_version
- app_build
- device_name
- release_channel
- app_version
- iphone

和：

```
(lldb) p c 0x0000656e6f687069
(long) iphone\0\0

(lldb) x s "0x40000002820700c0 + 0x20"
0x2820700e0: "5E8-8164-1AEA05E45CD9"
(lldb) po 0x00000002820700c0
2DE85147 8D62 45E8 8164 1AEA05E45CD9
...
```

values:

- iphone
- 2DE85147-8D62-45E8-8164-1AEA05E45CD9
  - 注：Swift的Shared=Bridged的String
- ...

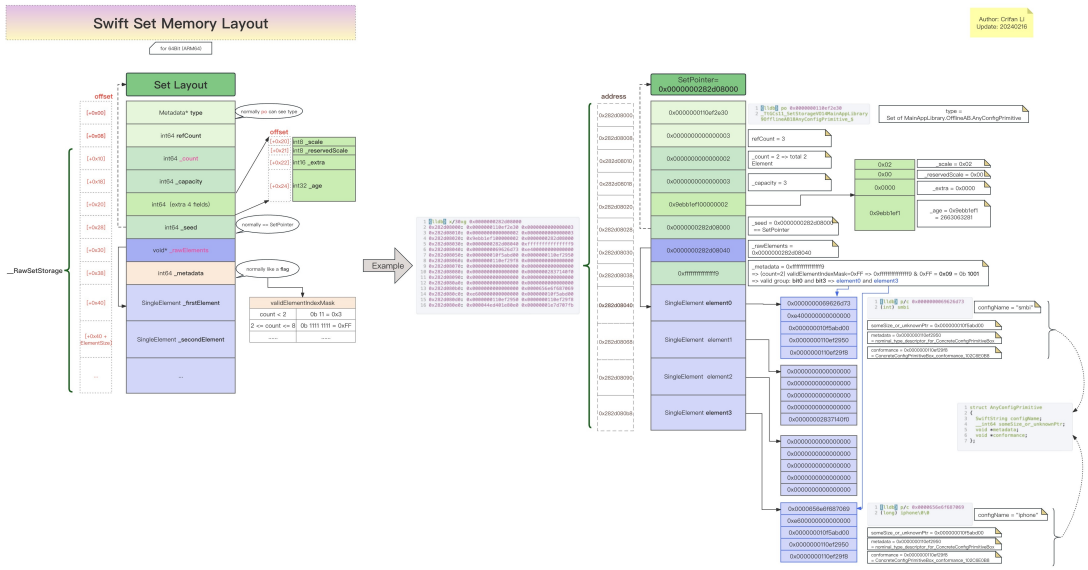
Enum枚举

Int整型

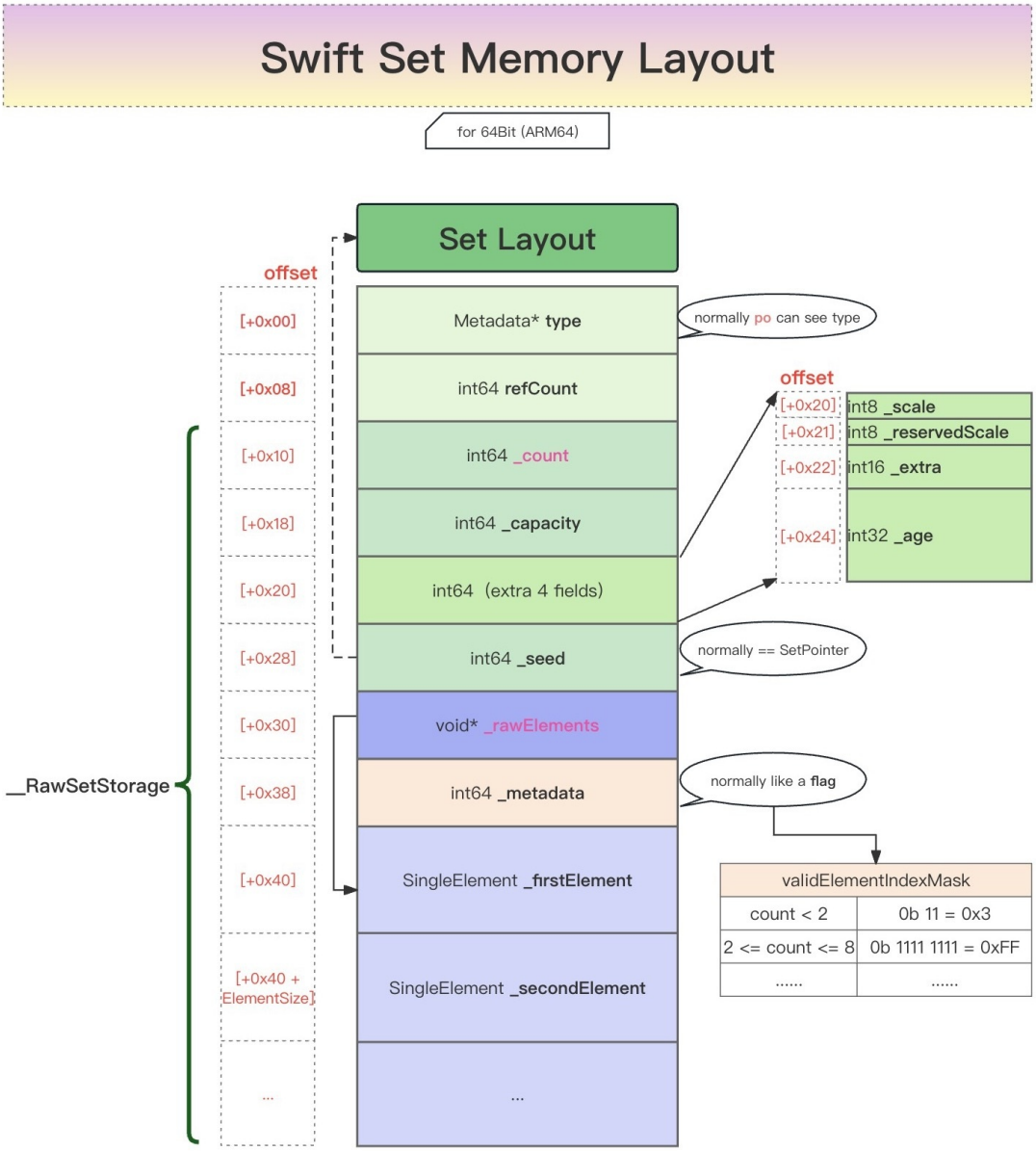
Set集合

内存布局

Swift的Set的内存布局图

- Swift的Set的内存布局图 = Swift Set Memory Layout
  - 在线预览
    - Swift的Set的内存布局结构图| ProcessOn免费在线作图,在线流程图,在线思维导图
  - 离线查看
    - 

  - 核心内容



文字

概述

- Swift的Set集合的内存布局=字段=属性
  - [+0x00] = Metadata* type
  - [+0x08] = int64 refCount
  - [+0x10] = int64 _count
  - [+0x18] = int64 _capacity
  - [+0x20] = Int64
    - [+0x20~0x20] = int8 _scale
    - [+0x21~0x21] = int8 _reservedScale
    - [+0x22~0x23] = int16 _extra
    - [+0x24~0x27] = int32 _age
  - [+0x28] = int64 _seed
  - [+0x30] = void* _rawElements
  - [+0x38] = int64 _metadata

详解

- Swift的Set集合的内存布局=字段=属性
  - [+0x00] = Metadata* type
  - [+0x08] = Int refCount
  - __RawSetStorage类型
    - [+0x10] = var _count: Int
    - [+0x18] = var _capacity: Int
    - [+0x20] = Int
      - [+0x20~0x20] = var _scale: Int8
      - [+0x21~0x21] = var _reservedScale: Int8
      - [+0x22~0x23] = var _extra: Int16
      - [+0x24~0x27] = var _age: Int32
    - [+0x28] = var _seed: Int
      - 往往是 == 当前Set指针
    - [+0x30] = var _rawElements: UnsafeMutableRawPointer
      - 指向真正数据的开始的地址 == realDataAddress
    - [+0x38] = var _metadata: UnsafeMutableRawPointer
      - == flag：哪些组数据是有效数据
        - （根据count去mask后的）最后一些bit位中的1，决定了对应位置（对应组）数据是有效数据
          - 可以写成
            - validElementIndexMask
              - count < 2：0b 11 = 0x3
              - 2<= count <= 8：0b 1111 1111 = 0xFF
              - ...

        - 举例
            - 0xfffffffffffffff9
                - 注：此处count=2
                    - mask后的值：只需要最后1个byte=8个bit的值
                - 0xfffffffffffffffe & 0xFF = 0x09 = 0b 0000 1001
                    - = bit0、bit3（的位置是1）
                    - = index0、index3是有效数据
            - 0xfffffffffffffffe
                - 注：此处count=1
                    - （此时count<2）mask后，只需要：2个bit位
                        - 对应如果有mask的话，可以理解为：
                            - validBitMask = 0x3 = 0b 11
                        - mask后的值：只需要最后2位=2个bit
                - 0xfffffffffffffffe & 0x3 = 0x2 = 0b 10
                    - = bit1（的位置是1）
                    - = index1 是有效数据
    - 说明：
        - 此处Int是64bit==int64
            - Int8=8bit
            - Int32=32bit

Swift源码

Set.swift

`swift/stdlib/public/core/Set.swift`

```
@frozen
@_eagerMove
public struct Set<Element: Hashable> {
  @usableFromInline
  internal var _variant: _Variant

  /// Creates an empty set with preallocated space for at least the specified
  /// number of elements.
  ///
  /// Use this initializer to avoid intermediate reallocations of a set's
  /// storage buffer when you know how many elements you'll insert into the set
  /// after creation.
  ///
  /// - Parameter minimumCapacity: The minimum number of elements that the
  ///   newly created set should be able to store without reallocating its
  ///   storage buffer.
  public // FIXME(reserveCapacity): Should be inlinable
  init(minimumCapacity: Int) {
    _variant = _Variant(native: _NativeSet(capacity: minimumCapacity))
  }

  /// Private initializer.
  @inlinable
  internal init(_native: __owned _NativeSet<Element>) {
    _variant = _Variant(native: _native)
  }

#if _runtime(_ObjC)
  @inlinable
  internal init(_cocoa: __owned __CocoaSet) {
    _variant = _Variant(cocoa: _cocoa)
  }

  /// Private initializer used for bridging.
  ///
  /// Only use this initializer when both conditions are true:
  ///
  /// * it is statically known that the given `NSSet` is immutable;
  /// * `Element` is bridged verbatim to Objective-C (i.e.,
  ///   is a reference type).
  @inlinable
  public // SPI(Foundation)
  init(_immutableCocoaSet: __owned AnyObject) {
    _internalInvariant(_isBridgedVerbatimToObjectiveC(Element.self),
      "Set can be backed by NSSet _variant only when the member type can be bridged ver
batim to Objective-C")
```

```
    self.init(_cocoa: __CocoaSet(_immutableCocoaSet))
  }
#endif
}
```

核心定义是：

```
public struct Set<Element: Hashable> {
  var _variant: _Variant
```

BridgeStorage.swift

`swift/stdlib/public/core/BridgeStorage.swift`

```
import SwiftShims

#if !$Embedded

@frozen
@usableFromInline
internal struct _BridgeStorage<NativeClass: AnyObject> {
  @usableFromInline
  internal typealias Native = NativeClass

  @usableFromInline
  internal typealias ObjC = AnyObject

  // rawValue is passed inout to _isUnique.  Although its value
  // is unchanged, it must appear mutable to the optimizer.
  @usableFromInline
  internal var rawValue: Builtin.BridgeObject

  @inlinable
  @inline(__always)
  internal init(native: Native, isFlagged flag: Bool) {
    // Note: Some platforms provide more than one spare bit, but the minimum is
    // a single bit.

    _internalInvariant(_usesNativeSwiftReferenceCounting(NativeClass.self))

    rawValue = _makeNativeBridgeObject(
      native,
      flag ? (1 as UInt) << _objectPointerLowSpareBitShift : 0)
  }

...
```

核心定义：

```
struct _BridgeStorage<NativeClass: AnyObject> {
  var rawValue: Builtin.BridgeObject
```

```
protocol BridgeStorage {
  associatedtype Native : AnyObject
  associatedtype ObjC : AnyObject


  init(native: Native, isFlagged: Bool)
  init(native: Native)
  init(objC: ObjC)


  mutating func isUniquelyReferencedNative() -> Bool
  mutating func isUniquelyReferencedUnflaggedNative() -> Bool
  var isNative: Bool {get}
  var isObjC: Bool {get}
  var nativeInstance: Native {get}
  var unflaggedNativeInstance: Native {get}
  var objCInstance: ObjC {get}
}

extension _BridgeStorage : BridgeStorage {}
```

SetStorage.swift

`swift/stdlib/public/core/SetStorage.swift`

```
/// An instance of this class has all `Set` data tail-allocated.
/// Enough bytes are allocated to hold the bitmap for marking valid entries,
/// keys, and values. The data layout starts with the bitmap, followed by the
/// keys, followed by the values.
// NOTE: older runtimes called this class _RawSetStorage. The two
// must coexist without a conflicting ObjC class name, so it was
// renamed. The old name must not be used in the new runtime.
@_fixed_layout
@usableFromInline
@_objc_non_lazy_realization
internal class __RawSetStorage: __SwiftNativeNSSet {
  // NOTE: The precise layout of this type is relied on in the runtime to
  // provide a statically allocated empty singleton.  See
  // stdlib/public/stubs/GlobalObjects.cpp for details.

  /// The current number of occupied entries in this set.
  @usableFromInline
  @nonobjc
  internal final var _count: Int

  /// The maximum number of elements that can be inserted into this set without
  /// exceeding the hash table's maximum load factor.
  @usableFromInline
  @nonobjc
  internal final var _capacity: Int

  /// The scale of this set. The number of buckets is 2 raised to the
  /// power of `scale`.
```

```
  @usableFromInline
  @nonobjc
  internal final var _scale: Int8

  /// The scale corresponding to the highest `reserveCapacity(_:)` call so far,
  /// or 0 if there were none. This may be used later to allow removals to
  /// resize storage.
  ///
  /// FIXME: <rdar://problem/18114559> Shrink storage on deletion
  @usableFromInline
  @nonobjc
  internal final var _reservedScale: Int8

  // Currently unused, set to zero.
  @nonobjc
  internal final var _extra: Int16

  /// A mutation count, enabling stricter index validation.
  @usableFromInline
  @nonobjc
  internal final var _age: Int32

  /// The hash seed used to hash elements in this set instance.
  @usableFromInline
  internal final var _seed: Int

  /// A raw pointer to the start of the tail-allocated hash buffer holding set
  /// members.
  @usableFromInline
  @nonobjc
  internal final var _rawElements: UnsafeMutableRawPointer

  // This type is made with allocWithTailElems, so no init is ever called.
  // But we still need to have an init to satisfy the compiler.
  @nonobjc
  internal init(_doNotCallMe: ()) {
    _internalInvariantFailure("This class cannot be directly initialized")
  }

  @inlinable
  @nonobjc
  internal final var _bucketCount: Int {
    @inline(__always) get { return 1 &<< _scale }
  }

  @inlinable
  @nonobjc
  internal final var _metadata: UnsafeMutablePointer<_HashTable.Word> {
    @inline(__always) get {
      let address = Builtin.projectTailElems(self, _HashTable.Word.self)
      return UnsafeMutablePointer(address)
    }
  }

  // The _HashTable struct contains pointers into tail-allocated storage, so
  // this is unsafe and needs `_fixLifetime` calls in the caller.
```

```
  @inlinable
  @nonobjc
  internal final var _hashTable: _HashTable {
    @inline(__always) get {
      return _HashTable(words: _metadata, bucketCount: _bucketCount)
    }
  }
}
```

和

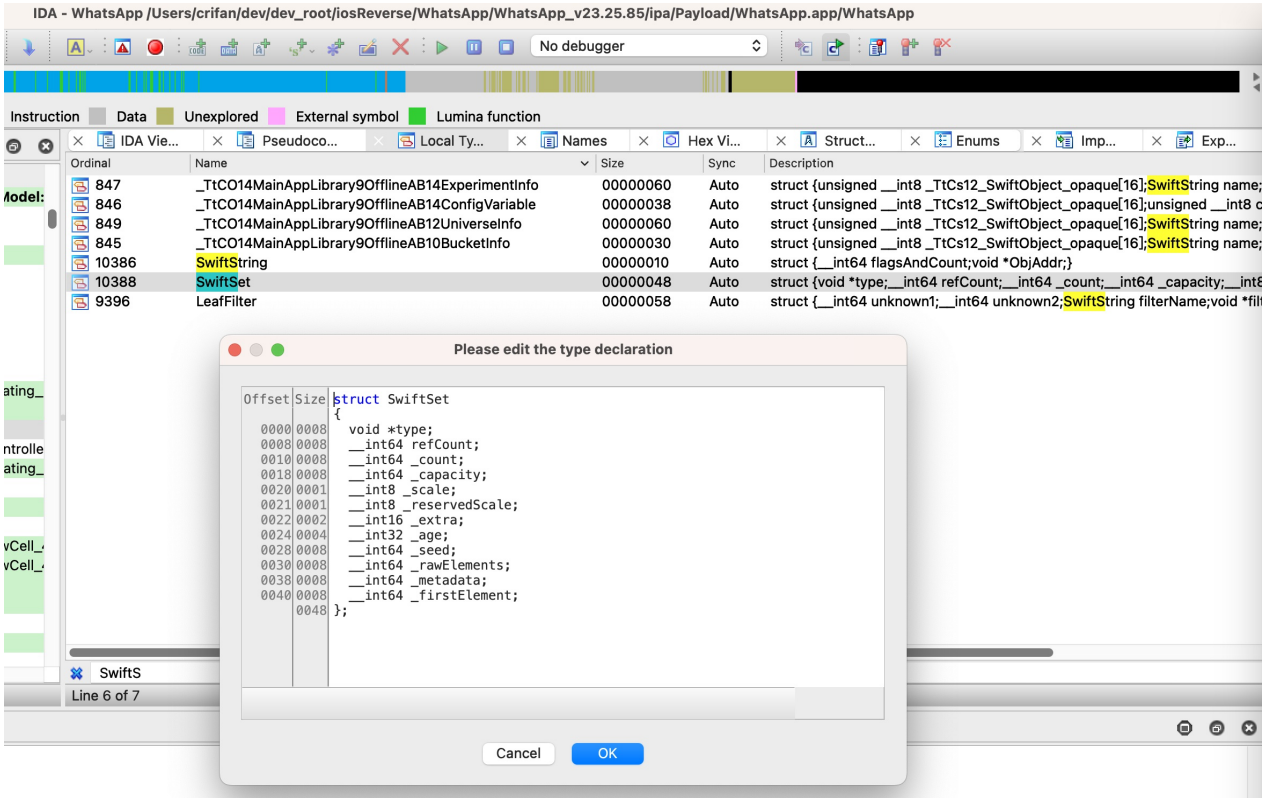
```
extension __RawSetStorage {
  /// The empty singleton that is used for every single Set that is created
  /// without any elements. The contents of the storage must never be mutated.
  @inlinable
  @nonobjc
  internal static var empty: __EmptySetSingleton {
    return Builtin.bridgeFromRawPointer(
      Builtin.addressof(&_swiftEmptySetSingleton))
  }
}
```

Runtime.swift

```
@_fixed_layout
@usableFromInline
@objc @_swift_native_objc_runtime_base(__SwiftNativeNSSetBase)
internal class __SwiftNativeNSSet {
  @nonobjc
  internal init() {}
  @objc public init(coder: AnyObject) {}
  deinit {}
}
```

IDA定义

```
struct SwiftSet
{
  void *type;
  __int64 refCount;
  __int64 _count;
  __int64 _capacity;
  __int8 _scale;
  __int8 _reservedScale;
  __int16 _extra;
  __int32 _age;
  __int64 _seed;
  __int64 _rawElements;
  __int64 _metadata;
  __int64 _firstElement;
};
```



举例

String字符串

空间占用=大小

- 空间占用：`0x10` 字节 = `16` 字节 = **2**个 `int64`

分类=类型

概述

- Swift中String
  - `Small String`
  - `Large String`
    - 三类
      - `Native`
        - 真正地址

          ```
          realStrAddr = objectAddr + 0x20
                      = objectAddr + 32
                      = objectAddr + nativeBias
          ```

      - `Shared`
        - 真正地址

          ```
          realStrAddr = objectAddr
          ```

        - 说明
          - 对于Bridge的字符串：objectAddr == NSString的地址
            - 内存布局保存的数据，是NSString中的数据
      - `Foreign`
        - 真正地址

          ```
          realStrAddr = objectAddr
          ```

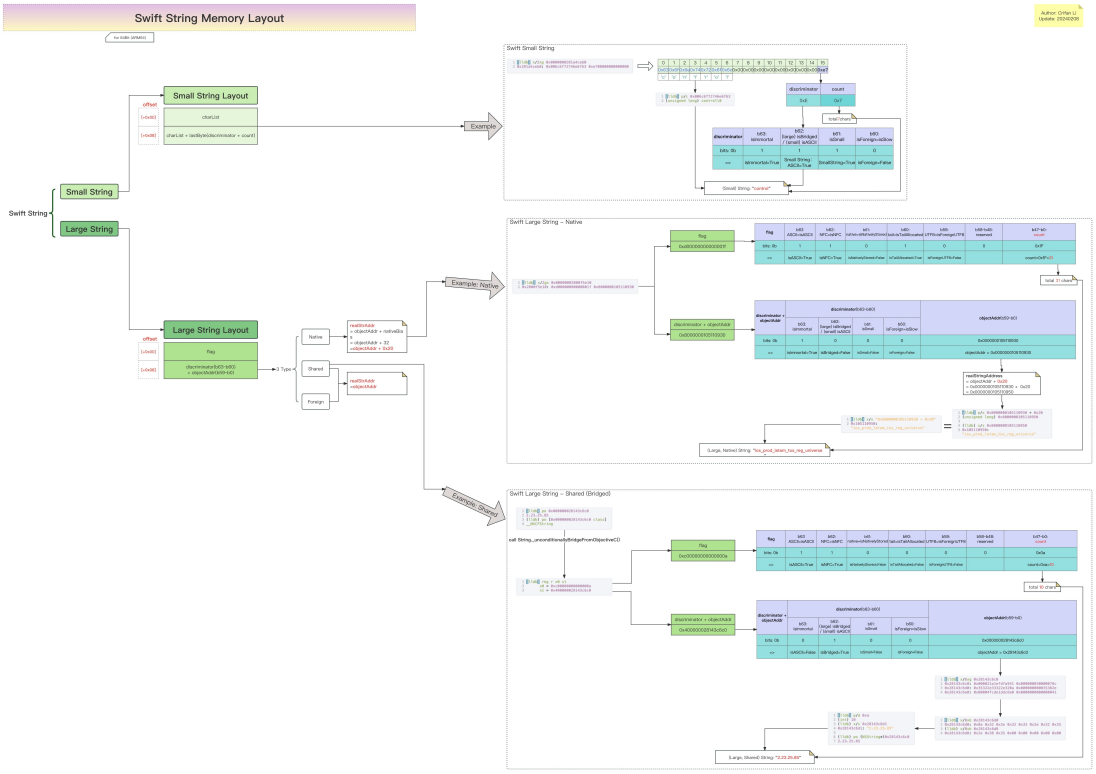
详解

- Swift中String
  - `Small String`
  - `Large String`
    - 分3类
      - native
        - Native strings have tail-allocated storage, which begins at an offset of `nativeBias` from the storage object's address. String literals, which reside in the constant section, are encoded as their start address minus `nativeBias`, unifying code paths for both literals ("immortal native") and native strings. Native Strings are

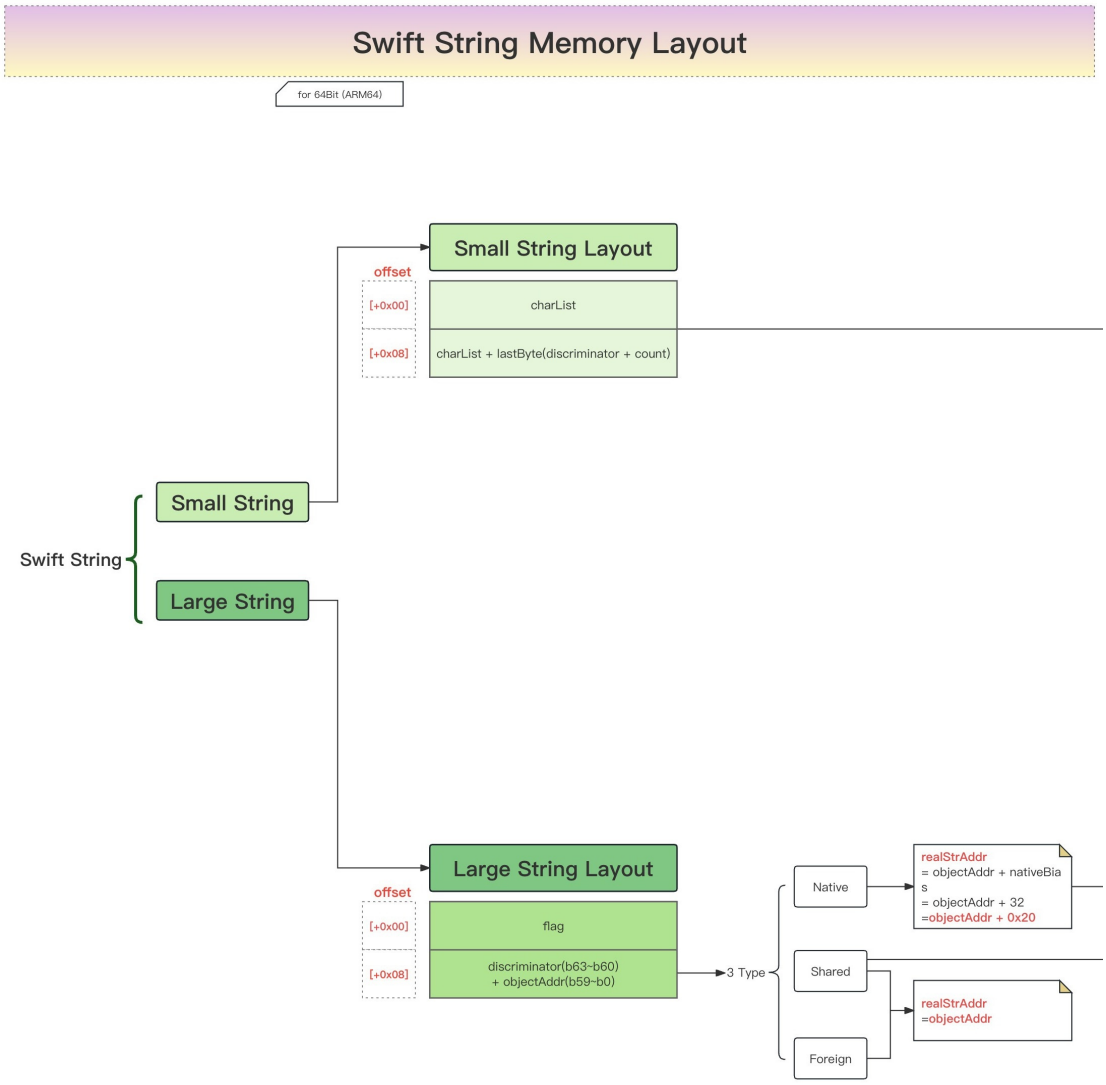
always managed by the Swift runtime.
    - shared
        - Shared strings do not have tail-allocated storage, but can provide access upon query to contiguous UTF-8 code units. Lazily-bridged NSStrings capable of providing access to contiguous ASCII/UTF-8 set the ObjC bit. Accessing shared string's pointer should always be behind a resilience barrier, permitting future evolution.
    - foreign
        - Foreign strings cannot provide access to contiguous UTF-8. Currently, this only encompasses lazily-bridged NSStrings that cannot be treated as "shared". Such strings may provide access to contiguous UTF-16, or may be discontiguous in storage. Accessing foreign strings should remain behind a resilience barrier for future evolution. Other foreign forms are reserved for the future.
- 其他说明
    - 对于Shared和foreign
        - always created and accessed behind a resilience barrier, providing flexibility for the future.

内存布局

Swift的String字符串的内存布局图

- Swift的String字符串的内存布局图 = Swift String Memory Layout
  - 在线预览
    - Swift的String内存布局结构图| ProcessOn免费在线作图,在线流程图,在线思维导图
  - 离线查看
    - 

  - 核心内容



文字

概述

- Swift中Large String
  - Native
    - 真正字符串的地址=objectAddr+0x20
  - Shared/Foreign
    - 真正字符串的地址=objectAddr

详解

- Swift的String(64位)
  - 通用（Small String和Large String都有）
    - discriminator
      - 关于名称和
        - 位置
          - == leading nibble == top nibble = 64bit的最顶部的4个bit：b63:b60 == [bit63 ~ bit60]
            - 注：nibble=半个字节 = 4bit
        - 名称
          - （此处有个专门的名字）discriminator = 辨别器：用于区分字符串的具体类型
            - 其实叫做：String type 更加容易理解
        - 所以代码和注释中下面的名称是一个意思：
          - b63:b60 = discriminator = leading nibble == top nibble
      - 具体字段和含义
        - b63 = isImmortal
          - Should the Swift runtime skip ARC
            - Small strings are just values, always immortal
            - Large strings can sometimes be immortal, e.g. literals
        - b62 = (large) isBridged / (small) isASCII
          - For large strings, this means lazily-bridged NSString: perform ObjC ARC
          - Small strings repurpose this as a dedicated bit to remember ASCII-ness
        - b61 = isSmall
          - Dedicated bit to denote small strings
        - b60 = isForeign = isSlow
          - cannot provide access to contiguous UTF-8
      - 完整的映射表 = 不同字段的组合，表示不同类型字符串

Form	b63	b62	b61	b60
Immortal, Small	1	ASCII	1	0
Immortal, Large	1	0	0	0
Native	0	0	0	0
Shared	x	0	0	0
Shared, Bridged	0	1	0	0
Foreign	x	0	0	1
Foreign, Bridged	0	1	0	1

- Small String
  - [+0x00-0x0E] = 8+7个字节 =15个字节 = 64+56位 = 120位 : realString
  - [+0xF] = 第16个字节=共8位: discriminator + count
    - byte15的[b4:b7] = discriminator
    - byte15的[b0:b3] = count
- Large String
  - [+0x00-0x07] = 8个字节 = 64位 : flag
    - b63 = ASCII = isASCII
    - b62 = NFC = isNFC
    - b61 = native = isNativelyStored
    - b60 = tail = isTailAllocated
    - b59 = UTF8 = isForeignUTF8
    - b58:48 = reserved
    - b47:0 = count
  - [+0x08-0x0F] = 8个字节 = 64位 : discriminator + objectAddr
    - b63:b60 = discriminator
      - 详见上述解释：b63 = isImmortal、b62 = (large) isBridged / (small) isASCII、b61 = isSmall、b60 = isForeign = isSlow
    - b60:b0 = objectAddr
      - realStrAddr = objectAddr + nativeBias = objectAddr + 32 = objectAddr + 0x20

Swift源码

StringObject.swift

- StringObject.swift
  - swift/stdlib/public/core/StringObject.swift at main · apple/swift (github.com)

```
extension _StringObject {
  @inlinable @inline(__always)
  internal init(_ small: _SmallString) {
    // Small strings are encoded as _StringObjects in reverse byte order
    // on big-endian platforms. This is to match the discriminator to the
    // spare bits (the most significant nibble) in a pointer.
    let word1 = small.rawBits.0.littleEndian
    let word2 = small.rawBits.1.littleEndian
#if _pointerBitWidth(_64)
    // On 64-bit, we copy the raw bits (to host byte order).
    self.init(rawValue: (word1, word2))
#elseif _pointerBitWidth(_32)
    // On 32-bit, we need to unpack the small string.
    let smallStringDiscriminatorAndCount: UInt64 = 0xFF00_0000_0000_0000


    let leadingFour = Int(truncatingIfNeeded: word1)
    let nextFour = UInt(truncatingIfNeeded: word1 &>> 32)
    let smallDiscriminatorAndCount = word2 & smallStringDiscriminatorAndCount
    let trailingTwo = UInt16(truncatingIfNeeded: word2)
    self.init(
      count: leadingFour,
      variant: .immortal(nextFour),
      discriminator: smallDiscriminatorAndCount,
      flags: trailingTwo)
#else
#error("Unknown platform")
#endif
    _internalInvariant(isSmall)
  }


  @inlinable
  internal static func getSmallCount(fromRaw x: UInt64) -> Int {
#if os(Android) && arch(arm64)
    return Int(truncatingIfNeeded: (x & 0x000F_0000_0000_0000) &>> 48)
#else
    return Int(truncatingIfNeeded: (x & 0x0F00_0000_0000_0000) &>> 56)
#endif
  }


  @inlinable @inline(__always)
  internal var smallCount: Int {
    _internalInvariant(isSmall)
```

```
    return _StringObject.getSmallCount(fromRaw: discriminatedObjectRawBits)
  }


  @inlinable
  internal static func getSmallIsASCII(fromRaw x: UInt64) -> Bool {
#if os(Android) && arch(arm64)
    return x & 0x0040_0000_0000_0000 != 0
#else
    return x & 0x4000_0000_0000_0000 != 0
#endif
  }
  @inlinable @inline(__always)
  internal var smallIsASCII: Bool {
    _internalInvariant(isSmall)
    return _StringObject.getSmallIsASCII(fromRaw: discriminatedObjectRawBits)
  }


  @inlinable @inline(__always)
  internal init(empty: ()) {
    // Canonical empty pattern: small zero-length string
#if _pointerBitWidth(_64)
    self._countAndFlagsBits = 0
    self._object = Builtin.valueToBridgeObject(Nibbles.emptyString._value)
#elseif _pointerBitWidth(_32)
    self.init(
      count: 0,
      variant: .immortal(0),
      discriminator: Nibbles.emptyString,
      flags: 0)
#else
#error("Unknown platform")
#endif
    _internalInvariant(self.smallCount == 0)
    _invariantCheck()
  }
}
```

StringBridge.swift

`swift/stdlib/public/core/StringBridge.swift`

```
extension String {
  @_spi(Foundation)
  public init(_cocoaString: AnyObject) {
    self._guts = _bridgeCocoaString(_cocoaString)
  }
}
```

->

```
@usableFromInline
```

```
@_effects(releasenone) // @opaque
internal func _bridgeCocoaString(_ cocoaString: _CocoaString) -> _StringGuts {
  switch _KnownCocoaString(cocoaString) {
  case .storage:
    return _unsafeUncheckedDowncast(
      cocoaString, to: __StringStorage.self).asString._guts
  case .shared:
    return _unsafeUncheckedDowncast(
      cocoaString, to: __SharedStringStorage.self).asString._guts
#if _pointerBitWidth(_64)
  case .tagged:
    // Foundation should be taking care of tagged pointer strings before they
    // reach here, so the only ones reaching this point should be back deployed,
    // which will never have tagged pointer strings that aren't small, hence
    // the force unwrap here.
    return _StringGuts(_SmallString(taggedCocoa: cocoaString)!)
#if arch(arm64)
  case .constantTagged
    let taggedContents = getConstantTaggedCocoaContents(cocoaString)!
    return _StringGuts(
      cocoa: taggedContents.untaggedCocoa,
      providesFastUTF8: false, //TODO: if contentsPtr is UTF8 compatible, use it
      isASCII: true,
      length: taggedContents.utf16Length
    )
#endif
#endif
  case .cocoa:
    // "Copy" it into a value to be sure nobody will modify behind
    // our backs. In practice, when value is already immutable, this
    // just does a retain.
    //
    // TODO: Only in certain circumstances should we emit this call:
    //   1) If it's immutable, just retain it.
    //   2) If it's mutable with no associated information, then a copy must
    //      happen; might as well eagerly bridge it in.
    //   3) If it's mutable with associated information, must make the call
    let immutableCopy
      = _stdlib_binary_CFStringCreateCopy(cocoaString)


#if _pointerBitWidth(_64)
    if _isObjCTaggedPointer(immutableCopy) {
      // Copying a tagged pointer can produce a tagged pointer, but only if it's
      // small enough to definitely fit in a _SmallString
      return _StringGuts(
        _SmallString(taggedCocoa: immutableCopy).unsafelyUnwrapped
      )
    }
#endif


    let (fastUTF8, isASCII): (Bool, Bool)
    switch _getCocoaStringPointer(immutableCopy) {
    case .ascii(_): (fastUTF8, isASCII) = (true, true)
    case .utf8(_): (fastUTF8, isASCII) = (true, false)
```

```
    default:  (fastUTF8, isASCII) = (false, false)
    }
    let length = _stdlib_binary_CFStringGetLength(immutableCopy)


    return _StringGuts(
      cocoa: immutableCopy,
      providesFastUTF8: fastUTF8,
      isASCII: isASCII,
      length: length)
  }
}
```

IDA定义

加到IDA中的定义：

```
struct SwiftString
{
  __int64 flagsAndCount;
  void *objAddr;
};
```

举例

Small String

"control"

寄存器中：

```
(lldb) reg r x0 x1 sp
      x0 = 0x006c6f72746e6f63
      x1 = 0xe700000000000000
```

内存中：

```
(lldb) po 0x0000000281a4cea0
MainAppLibrary.OfflineAB.BucketInfo
(lldb) x 6xg 0x0000000281a4cea0
0x281a4cea0: 0x0000000010415b460 0x0000000000000003
0x281a4ceb0: 0x006c6f72746e6f63 0xe700000000000000
0x281a4cec0: 0x0000000000002710 0x0000000281a4c930
(lldb) p c 0x006c6f72746e6f63
(long) control\0
```

->字符串本身 = realString：

```
0xe700000000000000 0x006c6f72746e6f63
```

->

```
(lldb) p c 0x006c6f72746e6f63
(unsigned long) control\0
```

->字符串是："control"

第15个字节=8位，分2部分：

- 最高4位=discriminator
  - 0xe
    - = 0b 1110
      - b63 = isImmortal -> 1
        - isImmortal=True
      - b62 = (large) isBridged / (small) isASCII -> 1
        - Small String：ASCII=True
      - b61 = isSmall -> 1
        - 是Small string
      - b60 = isForeign = isSlow -> 0
        - isForeign=False

- 最低4位=count
  - 0x7
    - 字符串长度是：7
      - = 7个字符
        - 对应着：control字符串的长度=7

Large String

Large String - Native

"ios_prod_latam_tos_reg_universe"

```
(lldb) po 0x00000002800f5e00
MainAppLibrary.OfflineAB.UniverseInfo

(lldb) x/6gx 0x00000002800f5e00
0x2800f5e00: 0x0000000105d3c398 0x0000000200000003
0x2800f5e10: 0xd00000000000001f 0x8000000105110930
0x2800f5e20: 0x6469725f72657375 0xe800000000000000
```

根据定义：

- MainAppLibrary.OfflineAB.UniverseInfo
  - [+0x10~0x18] = name
    - 类型：Swift的Large String（真正字符串地址是：name值+0x20）

此处值：

- [+0x10~0x18] = name
  - 0x2800f5e10: 0xd00000000000001f 0x8000000105110930

-》真正字符串地址是：

- 0x8000000105110930 + 0x20 = 0x8000000105110950

```
(lldb) x/s 0x8000000105110950
0x105110950: "ios_prod_latam_tos_reg_universe"
```

额外说明

（1）如果不加0x20，则看到的字符串是别的（错位后的）值：

```
(lldb) x/s 0x8000000105110930
0x105110930: "os_reg_experiment"
```

（2）去掉最开始的0x8，也是可以用x/s看到字符串的:

```
(lldb) x/s 0x0000000105110930
0x105110930: "os_reg_experiment"
(lldb) x/s 0x0000000105110950
```

```
0x105110950: "ios_prod_latam_tos_reg_universe"
```

类似的：去掉0x8前缀，可以用po查看出字符串的值：

```
(lldb) po (char *)0x0000000105110950
"ios_prod_latam_tos_reg_universe"
```

（3）不论是否加0x20，直接po查看，则都是（异常的，不是我们要的）时间类型的值：

```
(lldb) po 0x8000000105110930
2001-01-01 00:00:00 +0000
(lldb) po 0x8000000105110950
2001-01-01 00:00:00 +0000

(lldb) po (char *)0x8000000105110950
2001-01-01 00:00:00 +0000
```

后记：

另外某次去po，却连异常的时间类型的值都看不到，而是：就是数字：

```
(lldb) reg r x0 x1
      x0 = 0x000000000000007c
      x1 = 0xe100000000000000
(lldb) reg r x20 sp
     x20 = 0x000000016d658c20
      sp = 0x000000016d658c20
(lldb) x/8gx 0x000000016d658c20
0x16d658c20: 0xd000000000000021 0x80000001051946f0
0x16d658c30: 0x0000000000000002 0x00000002833c1140
0x16d658c40: 0x0000000105da1628 0x0000000105dbbcd0
0x16d658c50: 0x00000002833c1140 0x00000002828d0780
(lldb) po 0x80000001051946f0
9223372041235285744
```

而换x/s字符串查看，是可以看出：

（虽然是错误的，但是是）字符串的值的

```
(lldb) x/s 0x80000001051946f0
0x1051946f0: "dummy_aa_offline_user_rid_ios"
```

当然，真正地址+0x20的字符串：

```
(lldb) p/x 0x80000001051946f0 + 0x20
(unsigned long) 0x8000000105194710
```

->但是po也还是看不出:

```
(lldb) po 0x8000000105194710
```

```
9223372041235285776
```

只能用字符串查看：

```
(lldb) x -s 0x8000000105194710
0x105194710: "dummy_aa_offline_rid_universe_ios"
```

Large String - Shared

"2.23.25.85"

此处：

```
0xc00000000000000a 0x400000028143c6c0
```

其中：

- Large String
  - 0xc00000000000000a
    - 0xC = 12 = 0b 1100
      - b63 = ASCII = isASCII
        - True
      - b62 = NFC = isNFC
        - True
      - b61 = native = isNativelyStored
        - False
      - b60 = tail = isTailAllocated
        - False
      - b59 = UTF8 = isForeignUTF8
        - False
    - 0xa = 10： 字符串长度是10
  - 0x400000028143c6c0
    - discriminator = 0x4 = 0b 0100
      - b63 = isImmortal
        - False
      - b62 = (large) isBridged / (small) isASCII
        - True
      - b61 = isSmall
        - False
      - b60 = isForeign = isSlow
        - False
    - objectAddr = 0x28143c6c0

->

- Large String
  - length = 10

    - flag
        - isASCII = True
        - isNFC = True
        - isNativelyStored = False
        - isTailAllocated = False
        - isForeignUTF8 = False
    - discriminator
        - isImmortal = False
        - isBridged = True
        - isSmall = False
        - isForeign == isSlow = False
    - objectAddr = 0x28143c6c0

->

- Swift中的字符串
    - 是bridged桥接的
    - 字符串地址是：0x28143c6c0
    - 其他细节
        - 是ASCII的
        - 是NFC（Normal Form C）的
        - 不是尾部分配的TailAllocated

-> 此处对应着：从 `ObjC` 的 `NSString` ： `"2.23.25.85"`

```
(lldb) reg r x0
      x0 = 0x000000028143c6c0
(lldb) po $x0
2.23.25.85
(lldb) po [$x0 class]
__NSCFString
```

调用

- `libswiftFoundation.dylib`
    - `static Swift.String._unconditionallyBridgeFromObjectiveC(Swift.Optional<__C.NSString>) -> Swift.String`

而Bridge桥接过来的

->

此处：想要查看出字符串的值：

- （方式1）加上NSString强制类型转换去打印字符串

```
(lldb) po (NSString *)0x000000028143c6c0
2.23.25.85
```

- （方式2）自己查看ObjC的NSString的内存值，手动打印出字符串

此处（NSString类型的字符串的）内存值是：

```
(lldb) x/8xg 0x00000028143c6c0
0x28143c6c0: 0x000021a1efdfa941 0x000000030000078c
0x28143c6d0: 0x35322e33322e320a 0x000000000035382e
0x28143c6e0: 0x00004fcde1ddc6e0 0x0000000000000041
0x28143c6f0: 0x0000000000000000 0x0000000000000000
```

其中核心数据是：

- `0x28143c6d0: 0x35322e33322e320a 0x000000000035382e`

->可以调试查看到具体字符串的值和长度：

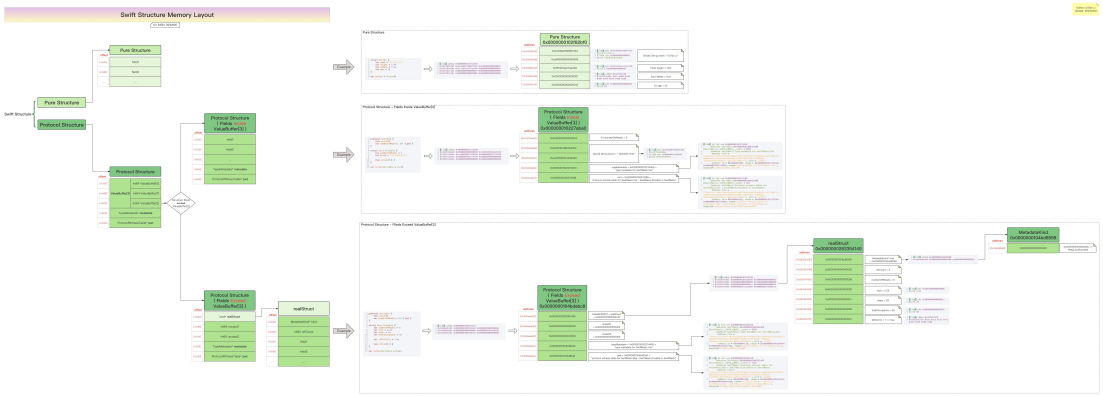
```
(lldb) p c 0x35322e33322e320a
(long) \n2.23.25
(lldb) p c 0x000000000035382e
(int) .85\0

(lldb) x/8xb 0x28143c6d0
0x28143c6d0: 0x0a 0x32 0x2e 0x32 0x33 0x2e 0x32 0x35
(lldb) x/8xb 0x28143c6d8
0x28143c6d8: 0x2e 0x38 0x35 0x00 0x00 0x00 0x00 0x00
(lldb) x/s 0x28143c6d0
0x28143c6d0: "\n2.23.25.85"
(lldb) x/s 0x28143c6d8
0x28143c6d8: ".85"

(lldb) p/d 0xa
(int) 10
(lldb) x/s 0x28143c6d1
0x28143c6d1: "2.23.25.85"
```

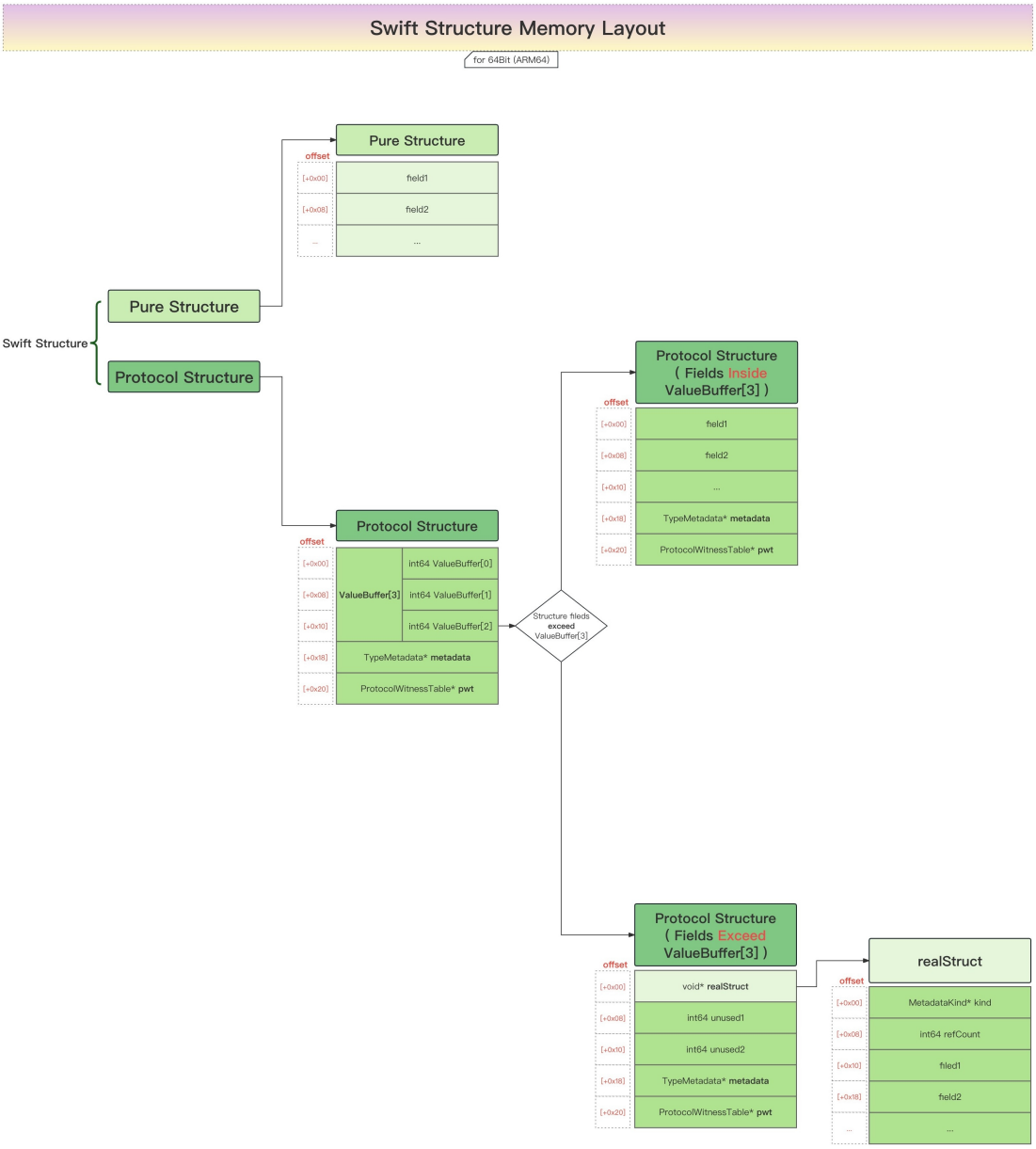
即：

- 字符串：
  - 长度：`10`
  - 值：`2.23.25.85`

内存布局

图

- Swift的Structure内存布局图 = Swift Structure Memory Layout
  - 在线预览
    - Swift的Structure内存布局图| ProcessOn免费在线作图,在线流程图,在线思维导图
  - 离线查看
    - 
  - 核心内容



文字

- Swift的Structure内存布局 概述
  - `Pure Structure`
    - 符合对齐标准，挨个字段存放
  - `Protocol Structure`
    - 字段**没超过** `ValueBuffer[3]` == `ValueBuffer[3]` 能放得下Structure的所有字段值
      - `ValueBuffer[0~3]` ：符合对齐标准，挨个字段存放
      - `[+0x18] = TypeMetadata* metadata`
      - `[+0x20] = ProtocolWitnessTable* pwt`
    - 字段**超过** `ValueBuffer[3]` == `ValueBuffer[3]` 放不下Structure的所有字段值
      - `ValueBuffer[0] = void* realStruct` = 真正结构体Structure字段
      - `ValueBuffer[1]` = 没用 = 无效数据
      - `ValueBuffer[2]` = 没用 = 无效数据
      - `[+0x18] = TypeMetadata* metadata`
      - `[+0x20] = ProtocolWitnessTable* pwt`

Swift源码

IDA定义

举例

纯Structure=不带Protocol的Structure

```
struct Person {
    let name = "Crifan Li"
    let height = 1.83
    let isMale = true
    let age = 20
}
var person = Person()
```

对应内存数据：

```
(lldb) x/8xg 0x0000000102f82bf0
0x102f82bf0: 0x4c206e6166697243 0xe900000000000069
0x102f82c00: 0x3ffd47ae147ae148 0x0000000000000001
0x102f82c10: 0x0000000000000014 0x0000000000000000
0x102f82c20: 0x0000000000000000 0x0000000000000000
(lldb) p/c 0x4c206e6166697243
(Int) Crifan L
(lldb) p/c 0x00000000000069
(Int) i\0\0\0\0\0\0
(lldb) p/f 0x3ffd47ae147ae148
(Int) 1.8300000000000001
(lldb) x/8xb 0x102f82c08
0x102f82c08: 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00
(lldb) p/d 0x0000000000000014
(Int) 20
```

->

- Person结构体
  - 大小=40=0x28
  - 内存布局
    - Person实例= person = 0x0000000102f82bf0
      - [+0x00~0x08] = String name
        - 0x4c206e6166697243 0xe900000000000069
          - "Crifan Li"
      - [+0x10] = Float height
        - 0x3ffd47ae147ae148
          - Float值：1.83
      - [+0x18] = bool isMale
        - 0x01 = true
      - [+0x20] = Int age
        - 0x0000000000000014 = 20

带Protocol的Structure

Struct结构体字段可以放得下 `ValueBuffer[3]` 的内存布局

代码：

```
protocol Drivable {
    func drive()
    var numberOfWheels: Int { get }
}

struct Car: Drivable {
    let numberOfWheels = 4
    let brand = "Tank300-Hi4T"

    func drive() { }
}

var curCar:Drivable = Car()
```

->

```
(lldb) x/8xg 0x000000010227aba0
0x10227aba0: 0x0000000000000004 0x2d3030336b6e6154
0x10227abb0: 0xec00000054346948 0x0000000102274450
0x10227abc0: 0x0000000102274188 0x0000000000000000
0x10227abd0: 0x0000000000000000 0x0000000000000000

(lldb) p/c 0x2d3030336b6e6154
(Int) Tank300-
(lldb) p/c 0x00000054346948
(Int) Hi4T\0\0\0\0
```

->

- Protocol的Structure，字段没超过ValueBuffer[3] = ValueBuffer[3]可以放得下Structure的字段 的内存布局：Car = 0x000000010227aba0
  - [+0x00] = ValueBuffer[0] = int64 numberOfWheels
    - 0x0000000000000004
  - [+0x08~0x10] = ValueBuffer[1:2] = String brand
    - 0x2d3030336b6e6154 0xec00000054346948
      - "Tank300-Hi4T"
  - [+0x18] = TypeMetadata * typeMetadata
    - 0x0000000102274450 = typeMetadata_struct_Car
      - type metadata for SwiftBasic.Car
  - [+0x20] = PWT * pwt
    - 0x0000000102274188 = PWT_Car
      - protocol witness table for SwiftBasic.Car : SwiftBasic.Drivable in SwiftBasic

其中：

- 0x0000000102274450 = typeMetadata_struct_Car

```
(lldb) im loo -va 0x0000000102274450
  Address: SwiftBasic[0x0000000100018450] (SwiftBasic.__DATA_CONST.__const + 808)
  Summary: SwiftBasic`type metadata for SwiftBasic.Car
    Module: file = "/Users/crifan/Library/Developer/Xcode/DerivedData/SwiftBasic-gbbeku
hnjvcbzueioosvbqhdnlyq/Build/Products/Debug-iphoneos/SwiftBasic.app/SwiftBasic", arch =
"arm64"
    Symbol: id = {0x000004be}, range = [0x0000000102274450-0x0000000102274468), name="t
ype metadata for SwiftBasic.Car", mangled="$s10SwiftBasic3CarVN"
```

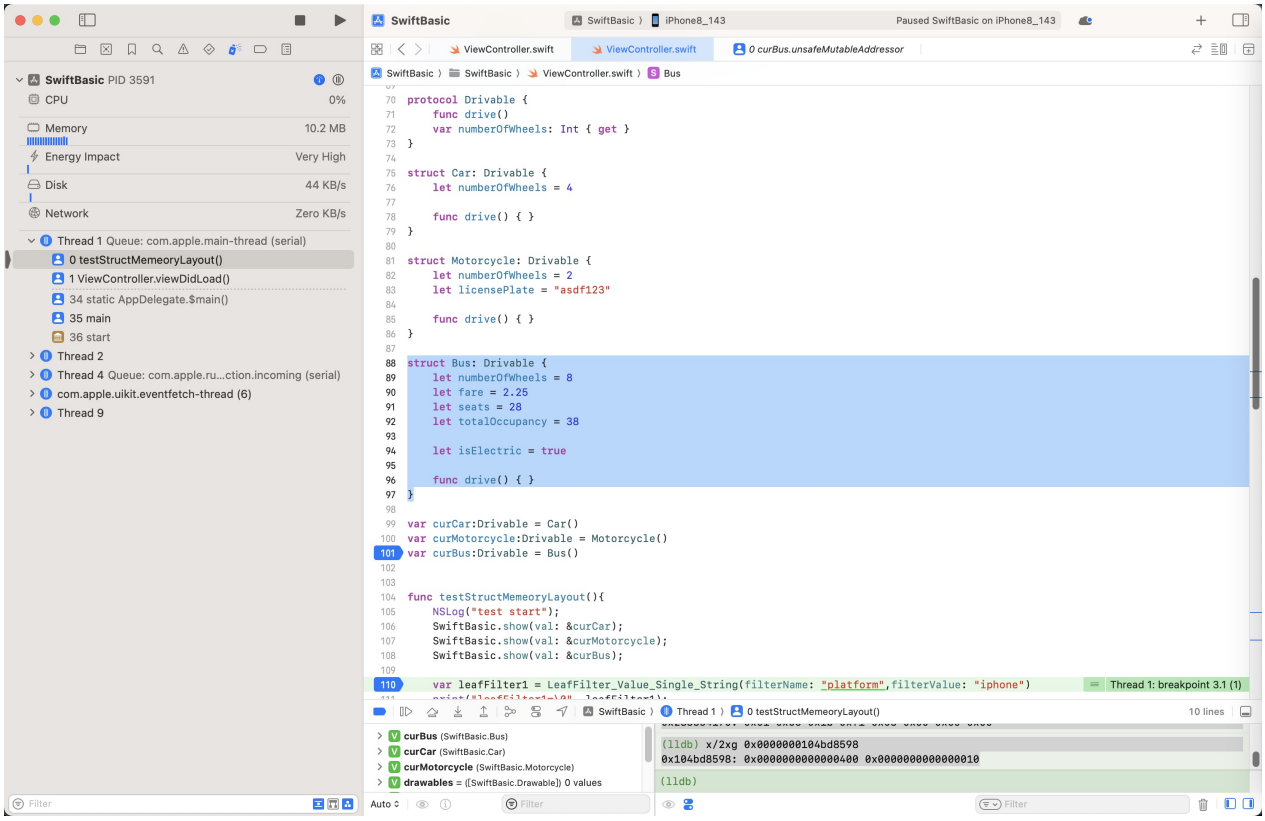
- 0x0000000102274188 = PWT_Car

```
(lldb) im loo -va 0x0000000102274188
  Address: SwiftBasic[0x0000000100018188] (SwiftBasic.__DATA_CONST.__const + 96)
  Summary: SwiftBasic`protocol witness table for SwiftBasic.Car : SwiftBasic.Drivable in
 SwiftBasic
    Module: file = "/Users/crifan/Library/Developer/Xcode/DerivedData/SwiftBasic-gbbeku
hnjvcbzueioosvbqhdnlyq/Build/Products/Debug-iphoneos/SwiftBasic.app/SwiftBasic", arch =
"arm64"
    Symbol: id = {0x000004ac}, range = [0x0000000102274188-0x00000001022741a0), name="p
rotocol witness table for SwiftBasic.Car : SwiftBasic.Drivable in SwiftBasic", mangled="
$s10SwiftBasic3CarVAA8DrivableAAWP"
```

Structue字段超过 `ValueBuffer[3]` 的Struct结构体的内存布局

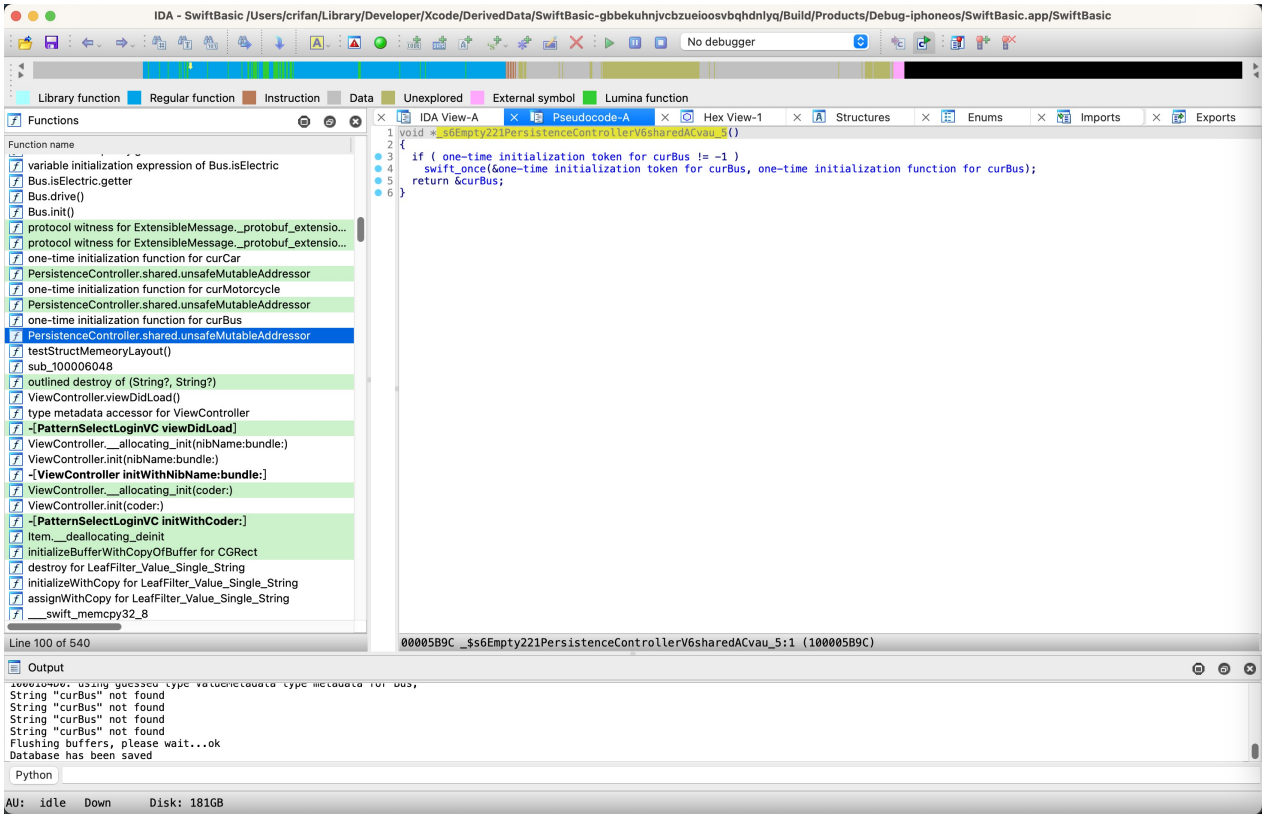
代码：

```
protocol Drivable {
    func drive()
    var numberOfWheels: Int { get }
}

struct Bus: Drivable {
    let numberOfWheels = 8
    let fare = 2.25
    let seats = 28
    let totalOccupancy = 38

    let isElectric = true

    func drive() { }
}

var curBus:Drivable = Bus()
```



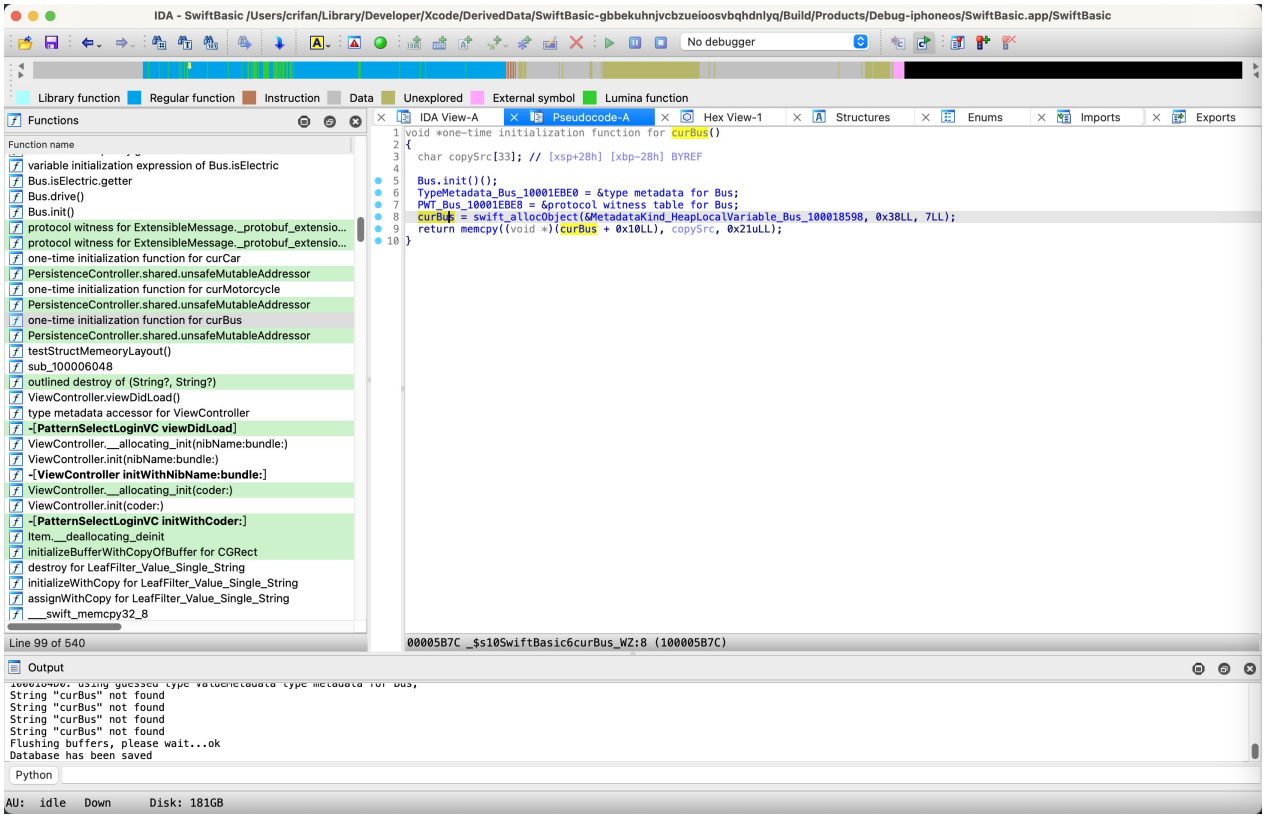
对应的生成的相关初始化代码是：

```
void *_s6Empty221PersistenceControllerV6sharedACvau_5()
{
  if ( one time initialization token for curBus != -1 )
    swift_once(&one time initialization token for curBus, one time initialization funct
ion for curBus);
  return &curBus;
}
```



和：

```
void __one_time_initialization_function_for_curBus()
{
  char copySrc[33]; // [xsp+28h] [xbp-28h] BYREF

  Bus.init()();
  TypeMetadata_Bus_10001EBE0 = &type metadata for Bus;
  PWT_Bus_10001EBE8 = &protocol witness table for Bus;
  curBus = swift_allocObject(&MetadataKind_HeapLocalVariable_Bus_100018598, 0x38LL, 7LL)
;
  return memcpy((void *)(curBus + 0x10LL), copySrc, 0x21uLL);
}
```



-》相关调试数据：

```
-------------- Drivable --------------
变量的地址：0x0000000104bdebc8
变量的内存：0x0000000283354140 0x0000000000000000 0x0000000000000000 0x0000000104bd84d0 0
x0000000104bd81a0
变量的大小：40 0x28

(lldb) x/6xg 0x0000000104bdebc8
0x104bdebc8: 0x0000000283354140 0x0000000000000000
0x104bdebd8: 0x0000000000000000 0x0000000104bd84d0
0x104bdebe8: 0x0000000104bd81a0 0x0000000000000000
(lldb) x/8xg 0x0000000283354140
0x283354140: 0x0000000104bd8598 0x0000000000000003
0x283354150: 0x0000000000000008 0x4002000000000000
0x283354160: 0x000000000000001c 0x0000000000000026
0x283354170: 0x00000003f11bc601 0x00000001f9b262ce
(lldb) im loo -va 0x0000000104bd84d0
      Address: SwiftBasic[0x00000001000184d0] (SwiftBasic.__DATA_CONST.__const + 936)
      Summary: SwiftBasic`type metadata for SwiftBasic.Bus
       Module: file = "/Users/crifan/Library/Developer/Xcode/DerivedData/SwiftBasic-gbb
ekuhnjvcbzueioosvbqhdnlyq/Build/Products/Debug-iphoneos/SwiftBasic.app/SwiftBasic", arc
h = "arm64"
       Symbol: id = {0x000004c1}, range = [0x0000000104bd84d0-0x0000000104bd84f8), name=
"type metadata for SwiftBasic.Bus", mangled="$s10SwiftBasic3BusVN"
(lldb) im loo -va 0x0000000104bd81a0
      Address: SwiftBasic[0x00000001000181a0] (SwiftBasic.__DATA_CONST.__const + 120)
      Summary: SwiftBasic`protocol witness table for SwiftBasic.Bus : SwiftBasic.Drivab
le in SwiftBasic
       Module: file = "/Users/crifan/Library/Developer/Xcode/DerivedData/SwiftBasic-gbb
ekuhnjvcbzueioosvbqhdnlyq/Build/Products/Debug-iphoneos/SwiftBasic.app/SwiftBasic", arc
```

```
h = "arm64"
       Symbol: id = {0x000004ad}, range = [0x0000000104bd81a0-0x0000000104bd81b8), name=
"protocol witness table for SwiftBasic.Bus : SwiftBasic.Drivable in SwiftBasic", mangled
="$s10SwiftBasic3BusVAA8DrivableAAWP"
(lldb) p/f 0x4002000000000000
(Int) 2.25
(lldb) p/d 0x000000000000001c
(Int) 28
(lldb) p/d 0x0000000000000026
(Int) 38
(lldb) x/8xb 0x283354170
0x283354170: 0x01 0xc6 0x1b 0xf1 0x03 0x00 0x00 0x00
(lldb) x/2xg 0x0000000104bd8598
0x104bd8598: 0x0000000000000400 0x0000000000000010
```

->含义：

- Protocol的Structure的，且Structure字段超过ValueBuffer[3]的（Bus的）内存布局 =
  - [+0x00] = ValueBuffer[0] = Structure的指针
    - 0x0000000283354140 = curBus的ptr
      - [+0x00] = MetadataKind kind
        - = 0x0000000000000400 == HeapLocalVariable
          - Bus是HeapLocalVariable类型变量
            - 内存是在堆中分配的
            - 且是个本地临时变量
      - [+0x08] = int64 refCount
        - 0x0000000000000003
      - Structure自己的属性值
        - [+0x10] = int64 numberOfWheels
          - 0x0000000000000008
        - [+0x18] = float fare
          - 0x4002000000000000 == 2.25
        - [+0x20] = int64 seats
          - 0x000000000000001c == 28
        - [+0x28] = int64 totalOccupancy
          - 0x0000000000000026 == 38
        - [+0x30] = bool(int8) isElectric
          - 0x01
  - [+0x08] = ValueBuffer[1] = 空 = 没用
  - [+0x10] = ValueBuffer[2] = 空 = 没用
  - [+0x18] = TypeMetadata * typeMetadata
    - 0x0000000104bd84d0 = typeMetadata_struct_Bus
      - type metadata for SwiftBasic.Bus
  - [+0x20] = PWT * pwt
    - 0x0000000104bd81a0 = PWT_Bus
      - protocol witness table for SwiftBasic.Bus : SwiftBasic.Drivable in SwiftBasic

其中：

- 0x0000000104bd84d0 = typeMetadata_struct_Bus

```
(lldb) im loo -va 0x0000000104bd84d0
  Address: SwiftBasic[0x00000001000184d0] (SwiftBasic.__DATA_CONST.__const + 936)
  Summary: SwiftBasic`type metadata for SwiftBasic.Bus
    Module: file = "/Users/crifan/Library/Developer/Xcode/DerivedData/SwiftBasic-gbbeku
hnjvcbzueioosvbqhdnlyq/Build/Products/Debug-iphoneos/SwiftBasic.app/SwiftBasic", arch =
"arm64"
    Symbol: id = {0x000004c1}, range = [0x0000000104bd84d0-0x0000000104bd84f8), name="t
ype metadata for SwiftBasic.Bus", mangled="$s10SwiftBasic3BusVN"
```

- 0x0000000104bd81a0 = PWT_Bus

```
(lldb) im loo -va 0x0000000104bd81a0
  Address: SwiftBasic[0x00000001000181a0] (SwiftBasic.__DATA_CONST.__const + 120)
  Summary: SwiftBasic`protocol witness table for SwiftBasic.Bus : SwiftBasic.Drivable in
 SwiftBasic
    Module: file = "/Users/crifan/Library/Developer/Xcode/DerivedData/SwiftBasic-gbbeku
hnjvcbzueioosvbqhdnlyq/Build/Products/Debug-iphoneos/SwiftBasic.app/SwiftBasic", arch =
"arm64"
    Symbol: id = {0x000004ad}, range = [0x0000000104bd81a0-0x0000000104bd81b8), name="p
rotocol witness table for SwiftBasic.Bus : SwiftBasic.Drivable in SwiftBasic", mangled="
$s10SwiftBasic3BusVAA8DrivableAAWP"
```

Tuple元祖

附录

下面列出相关参考资料。

参考资料

- 【已解决】iOS逆向Swift：Dictionary字典的内存布局结构
- 【已解决】iOS逆向Swift：找Swift的Dictionary字典的测试数据
- 【整理】iOS逆向：去画Swift的Dictionary字典的内存布局结构图
- 【未解决】iOS逆向WhatsApp：genWAABOfflineAssignSha256_10039E294
- 【已解决】iOS逆向：Swift中String字符串的内部结构和逻辑
- 【已解决】iOS逆向Swift：研究_unconditionallyBridgeFromObjectiveC后的SwiftString的值的逻辑和含义
- 【未解决】iOS逆向Swift：String._unconditionallyBridgeFromObjectiveC
- 【已解决】iOS逆向Swift：VWT=ValueWitnessTable=值见证表
- 【已解决】iOS逆向Swift：IDA优化插入ValueWitnessTable结构体定义
- 【已解决】iOS逆向Swift：IDA中自定义结构体中加上bit位的定义
- 【已解决】iOS逆向Swift：Builtin.NativeObject的VWT
- 【已解决】iOS逆向Swift：Data数据的内存布局结构
- 【已解决】iOS逆向Swift：Data的InlineSlice内存布局举例
- 【已解决】iOS逆向Swift：画Swift的Data数据的内存布局结构图
- 【未解决】iOS逆向Swift：IDA中添加Swift的Data相关定义
- 【已解决】iOS逆向Swift：画ClassMetadata的内存布局结构图
- 【整理】iOS逆向：去画Swift的ValueMetadata和VWT的内存布局结构图
- 【已解决】iOS逆向Swift：ClassMetadata的定义和内存布局
- 【未解决】iOS逆向Swift：__DataStorage的ClassMetadata内存布局值举例
- 【已解决】iOS逆向Swift：Metadata的type即元数据的类型定义
- 【已解决】iOS逆向Swift：ValueMetadata的定义和内存布局结构
- 【已解决】iOS逆向Swift：IDA中加上定义：ValueMetadata
- 【已解决】iOS逆向Swift：StructMetadata的字段定义和内存布局结构
- 【已解决】iOS逆向Swift：Swift中有几种Metadata
- 【已解决】iOS逆向Swift：TypeMetadata=Type Metadata
- 【已解决】iOS逆向Swift：IDA中添加ClassMetadata的定义
- 【已解决】iOS逆向Swift：Array数组的内存布局
- 【整理】iOS逆向：去画Swift的Array数组的内存布局结构图
- 【已解决】iOS逆向Swift：StructMetadata的字段定义和内存布局结构
- 【已解决】iOS逆向Swift：StructDescriptor
- 【已解决】iOS逆向Swift：FieldDescriptor
- 【已解决】iOS逆向Swift：画StructMetadata的内存布局图
- 【整理】iOS逆向：去画Swift的String字符串的内存布局结构图
- 【已解决】iOS逆向：Set集合的内存布局中如何判断哪组数据是有效的
- 【已解决】iOS逆向Swift：Set集合的内存布局结构
- 【已解决】iOS逆向Swift：Set集合的_rawElements之后的数据的存储逻辑
- 【整理】iOS逆向：去画Swift的Set集合的内存布局结构图
- 【未解决】iOS逆向Swift：__RawSetStorage
- 【未解决】iOS逆向Swift：__SwiftNativeNSSet
- 【已解决】iOS逆向Swift：给IDA中加上Set集合的结构体定义SwiftSet
- 【已解决】iOS逆向Swift：Set中的偏移量0x38是什么值

-
- 初探Swift底层Metadata - 掘金
- swift-evolution/proposals/0247-contiguous-strings.md at main · apple/swift-evolution (github.com)
- swift/docs/CppInteroperability/CppInteroperabilityManifesto.md at main · apple/swift
-