Mobius 2024 Spring

Swift 5.9: разбираем интероп и «женим» с C++

Андрей Филипенков

kaspersky

План

Задача интероп

Swift → C++

Доступные платформы

Reference types

Текущие ограничения

Настройка Xcode проекта

Демонстрация

План

Задача интероп

Swift → C++

Доступные платформы

Reference types

Текущие ограничения

Настройка Xcode проекта

Демонстрация

План

Задача интеропа

Swift → C++

Доступные платформы

Reference types

Текущие ограничения

Настройка Xcode проекта

Демонстрация

План

Задача интероп

Swift → C++

Доступные платформы

Reference types

Текущие ограничения

Настройка Xcode проекта

Демонстрация

План

Задача интеропа

Swift → C++

Доступные платформы

Reference types

Текущие ограничения

Настройка Xcode проекта

Демонстрация

План

Задача интероп

Swift → C++

Доступные платформы

Reference types

Текущие ограничения

Настройка Xcode проекта

Демонстрация

План

Задача интероп

Swift → C++

Доступные платформы

Reference types

Текущие ограничения

Настройка Xcode проекта

Демонстрация

Задача интеропа

Swift → Objective-C++ → C++

C++ → Objective-C++ → Swift

Задача интеропа

Swift → ~~Objective-C++~~ → C++

C++ → ~~Objective-C++~~ → Swift

Swift → C++


```
namespace S {

std::vector<std::string> buildVec();

}
```

Swift → C++

std::vector<std::string> S::buildVec()

```
@implementation objc

+ (NSArray<NSString*>*)buildVec {
    const auto v = S::buildVec();
    auto ar = [NSMutableArray<NSString*> arrayWithCapacity:v.size()];
    for (const auto& s : v)
        [ar addObject:[NSString stringWithUTF8String:s.data()]];
    return [ar copy];
}

@end
```

Objective-C wrapper

```
@interface objc

+ (NSArray<NSString*>*)buildVec;

@end


// Swift usage
let cppVec = objc.buildVec()
```

Objective-C wrapper

```
@interface objc

+ (NSArray<NSString*>*)buildVec;

@end


// Swift usage
let cppVec = objc.buildVec()
```

C++ interop


```
namespace S {

std::vector<std::string> buildVec();

}

// Swift usage
let cppVec = S.buildVec()
```

Swift → C++

std::vector<std::string> S::buildVec()

```
public enum S {

public static func buildVec() ->
  std.vector<basic_string<Int8, char_traits<Int8>, allocator<Int8>>,
   allocator<basic_string<Int8, char_traits<Int8>, allocator<Int8>>>>
}
```

Swift → C++

std::vector<std::string> S::buildVec()

```
public enum S {

public static func buildVec() ->
  std.vector<basic_string<Int8, char_traits<Int8>, allocator<Int8>>,
   allocator<basic_string<Int8, char_traits<Int8>, allocator<Int8>>>>
}
```

Swift → C++

enum ← namespace

```
// C++
namespace S {
  const int n;

  void f();
}
```

Swift → C++

enum ← namespace

```
// C++                  // Swift
namespace S {           enum S {
  const int n;            let n = 5

  void f();               func f() {}
}                       }
```

Swift → C++

enum ← namespace

```
// C++
namespace S {
  const int n;

  void f();
}
```

```
// Swift
enum S {
  let n = 5

  func f() {}
}
```

```
// Objective-C
@interface S

@property(nonatomic, class, readonly) int n;

+ (void)f;

@end
```

Swift → C++

std::vector<std::string> S::buildVec()

```
public enum S {

public static func buildVec() ->
  std.vector<basic_string<Int8, char_traits<Int8>, allocator<Int8>>,
   allocator<basic_string<Int8, char_traits<Int8>, allocator<Int8>>>>
}
```

Swift → C++

std::vector<std::string> S::buildVec()

```
public enum S {

public static func buildVec() ->
  std.vector<T, allocator<T>>
}
```

Swift → C++

std::vector<std::string> S::buildVec()

```
public enum S {

public static func buildVec() ->
  std.vector<T, allocator<T>>
}
```

```
template<
    class T,
    class Allocator = std::allocator<T>
> class vector;
```

Swift → C++

basic_string<Int8, char_traits<Int8>, allocator<Int8>>

```
template<
    class CharT,
    class Traits = std::char_traits<CharT>,
    class Allocator = std::allocator<CharT>
> class basic_string;
```

Swift → C++

basic_string<Int8, char_traits<Int8>, allocator<Int8>>

```
template<
    class CharT,
    class Traits = std::char_traits<CharT>,
    class Allocator = std::allocator<CharT>
> class basic_string;
```

```
// можно
using string = basic_string<char>;
using u16string = basic_string<char16_t>;
```

Swift → C++

basic_string<Int8, char_traits<Int8>, allocator<Int8>>

```
template<
    class CharT,
    class Traits = std::char_traits<CharT>,
    class Allocator = std::allocator<CharT>
> class basic_string;
```

```
// можно
using string = basic_string<char>;
using u16string = basic_string<char16_t>;

// нельзя
using wstring = basic_string<wchar_t>;
using u8string = basic_string<char8_t>;
using u32string = basic_string<char32_t>;
```

Swift → C++

std.vector<basic_string<Int8>> buildVec()

```
let cppArray = S.buildVec()
```

Swift → C++

std.vector<basic_string<Int8>> buildVec()

```
let cppArray = S.buildVec()
```

Swift's **RandomAccessCollection**

Swift → C++

std.vector<basic_string<Int8>> buildVec()

```
let cppArray = S.buildVec()

print(cppArray.count, cppArray[0])

for element in cppArray {
}
```

Swift's **RandomAccessCollection**

Swift → C++

std.vector<basic_string<Int8>> buildVec()

```
let cppArray = S.buildVec()

print(cppArray.count, cppArray[0])

for element in cppArray {
}
```

Swift's **RandomAccessCollection**

map, filter etc.

for-in


```
for element in cppArray {
}
```

```
for element in cppArray {
}
```

DEEP COPY!

for-in

```
for element in cppArray {
}
```

DEEP COPY!

forEach

```
cppArray.forEach {
}
```

OK

https://github.com/apple/swift/issues/66158

Swift → C++

std.vector<basic_string<Int8>> buildVec()

```
let swiftArray = Array<String>(cppArray)
```

Swift → C++

function

```
// C++
void printWelcomeMessage(const std::string& name);


// Swift
printWelcomeMessage("Thomas")
```

C++ - **class**

```
class Color {
public:
  static Color getRandomColor();

  Color();
  Color(float red, float green,
float blue);
  Color(float value);

  float red, green, blue;
};
```

Swift

C++ - **class**

```
class Color {
public:
  static Color getRandomColor();

  Color();
  Color(float red, float green,
float blue);
  Color(float value);

  float red, green, blue;
};
```

Swift


```
let theEmptiness = Color()
let oceanBlue = Color(0.0, 0.0, 1.0)
let seattleGray = Color(0.7)


let color = Color.getRandomColor()
print("""
  R: \(color.red)
  G: \(color.green)
  B: \(color.blue)
""")
```

Swift → C++

enum

```
enum
```

Swift → C++

enum

`enum`

`enum class`

Swift → C++

enum

```
enum
```

```
enum class
```

C++ - **enum class**

```
enum class E {
    One = 1,
    Two,
};
```

Swift

C++ - **enum class**

```
enum class E {
    One = 1,
    Two,
};
```

Swift


```
public enum E : Int32 {
    case One = 1
    case Two = 2
}
```

C++ - **enum**

```
enum E {
    One = 1,
    Two,
};
```

Swift

C++ - **enum**

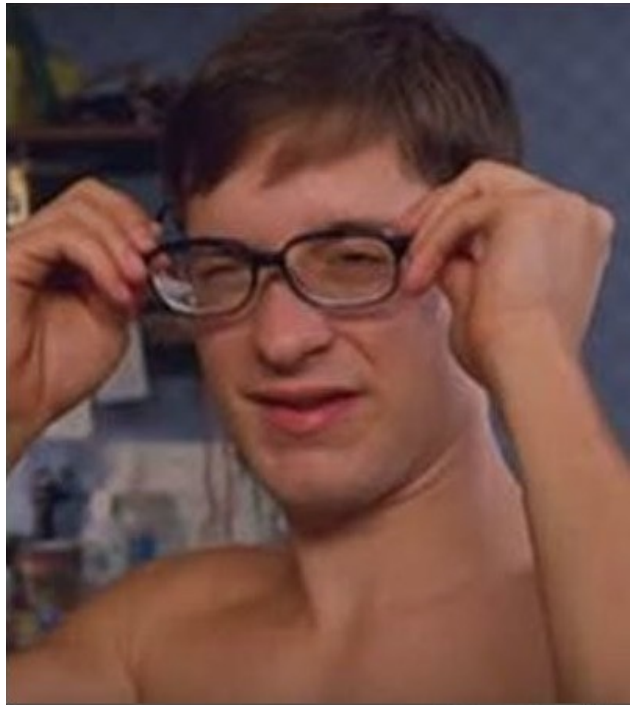
```
enum E {
    One = 1,
    Two,
};
```

Swift


```
public enum E : Int32 {
    case One = 1
    case Two = 2
}
```

C++ - **enum**

```
enum E {
    One = 1,
    Two,
};
```



Swift


```
public enum E : Int32 {
    case One = 1
    case Two = 2
}
```

C++ - **enum**

```
enum E {
    One = 1,
    Two,
};
```

Swift


```
public struct E : Equatable,
RawRepresentable {

    public init(_ rawValue:
UInt32)

    public init(rawValue:
UInt32)

    public var rawValue:
UInt32
}

public var One: E { get }
public var Two: E { get }
```

Платформы

iOS / tvOS: 11.0
macOS: 10.13

iOS / tvOS: 11.0
macOS: 10.13

Apple Clang 12+

iOS / tvOS: 11.0
macOS: 10.13

Apple Clang 12+

Ubuntu: 18.04
Windows: 10

Apple

iOS / tvOS: 11.0
macOS: 10.13

Apple Clang 12+

Другие ОС

Ubuntu: 18.04
Windows: 10

Clang: LLVM 11+

Платформы

Reference Types Imported from C++

Apple

iOS/tvOS: ~~11.0~~ **16.4**
macOS: ~~10.13~~ **13.3**

Apple Clang 12+

Другие ОС

Ubuntu: 18.04
Windows: 10

Clang: LLVM 11+

Reference Types

Default:

C++ **struct** / **class** → Swift **struct**

struct:
value type

class:
reference type

struct:
value type

class:
reference type

struct == class

std::shared_ptr

Reference types

Заголовочный файл **<swift/bridging>**

Reference types

Заголовочный файл **<swift/bridging>**

Вечноживущее: **SWIFT_IMMORTAL_REFERENCE**

Reference types

Заголовочный файл **<swift/bridging>**

Вечноживущее: **SWIFT_IMMORTAL_REFERENCE**

Просто небезопасное: **SWIFT_UNSAFE_REFERENCE**

Reference types

Свой ссылочный тип

```
class SharedObject : IntrusiveReferenceCounted<SharedObject> {
public:
    SharedObject(const SharedObject&) = delete; // non-copyable

} SWIFT_SHARED_REFERENCE(retainSharedObject, releaseSharedObject);
```

Reference types

Свой ссылочный тип

```
class SharedObject : IntrusiveReferenceCounted<SharedObject> {
public:
    SharedObject(const SharedObject&) = delete; // non-copyable

} SWIFT_SHARED_REFERENCE(retainSharedObject, releaseSharedObject);

void retainSharedObject(SharedObject*);
void releaseSharedObject(SharedObject*);
```

Ограничения

Virtual member functions

r-value reference / universal reference: T&&

deleted copy constructor / move-only types

Ограничения

Virtual member functions

r-value reference / universal reference: T&&

deleted copy constructor / move-only types

Ограничения

Virtual member functions

r-value reference / universal reference: T&&

deleted copy constructor / move-only types

Ограничения

Virtual member functions

```cpp
// C++
class A {
public:
    virtual void f() const;
};
```

```swift
// Swift
public struct A {

    public init()

    @available(*, unavailable, message:
      "virtual functions are not yet
      available in Swift")
    public func f()
}
```

Ограничения

Доступные ти́пы из C++ STL

```
std::string / std::u16string
std::array
std::vector
std::map / std::unordered_map
std::set / std::unordered_set
std::optional
std::pair / ~~std::tuple~~
```

Ограничения

Доступные ти́пы из C++ STL

```
std::string / std::u16string
std::array
std::vector
std::map / std::unordered_map
std::set / std::unordered_set
std::optional
std::pair / ~~std::tuple~~
```

Ограничения

Доступные ти́пы из C++ STL

```
std::string / std::u16string
std::array
std::vector
std::map / std::unordered_map
std::set / std::unordered_set
std::optional
std::pair / ~~std::tuple~~
```

Ограничения

Доступные ти́пы из C++ STL

```
std::string / std::u16string
std::array
std::vector
std::map / std::unordered_map
std::set / std::unordered_set
std::optional
std::pair / ~~std::tuple~~
```

Ограничения

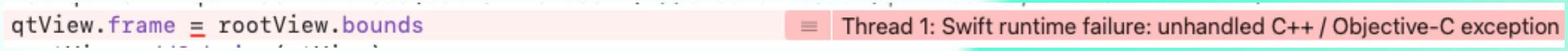
Доступные ти́пы из C++ STL

```
std::string / std::u16string
std::array
std::vector
std::map / std::unordered_map
std::set / std::unordered_set
std::optional
std::pair / ~~std::tuple~~
```

Ограничения

Исключения

- можно использовать C++ код, который бросает исключения
- но нельзя ловить исключения в Свифте
- и нельзя, чтоб C++ исключения доходили до Свифта, иначе fatal error



Ограничения

Шаблоны

```
// C++

template<typename T>
class A
{
public:
  A(T myvar) : _myvar{myvar} {}

private:
  T _myvar;
};
```

Ограничения

Шаблоны

```
// C++

template<typename T>
class A
{
public:
  A(T myvar) : _myvar{myvar} {}

private:
  T _myvar;
};
```

```
// Swift

// A<Int>
let a = A(5)
```

Ограничения

Шаблоны

```
// C++

template<typename T>
class A
{
public:
  A(T myvar) : _myvar{myvar} {}

private:
  T _myvar;
};
```

```
// Swift

// A<Int>
let a = A(5)
```

Ограничения

Шаблоны

```
// C++

A<int> f();
```

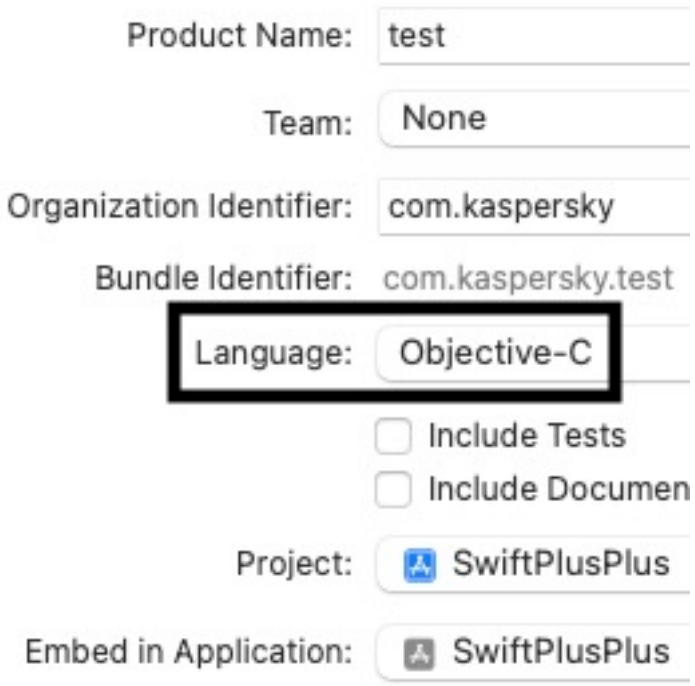
```
// Swift

// A<Int>
let a = f()
```

Как включить

Xcode проект

Как включить

Создать Framework target



Как включить

Framework public headers


```objc
#import <Foundation/Foundation.h>


//! Project version number for CppQt.
FOUNDATION_EXPORT double CppQtVersionNumber;


//! Project version string for CppQt.
FOUNDATION_EXPORT const unsigned char CppQtVersionString[];


#include <CppQt/showQt.h>
```

C++ framework

Apple Clang Module Verifier - Options

Setting	AdminKit
Enable Module Verifier	Yes
Supported Language Dialects	gnu17 gnu++20
Supported Languages	objective-c++

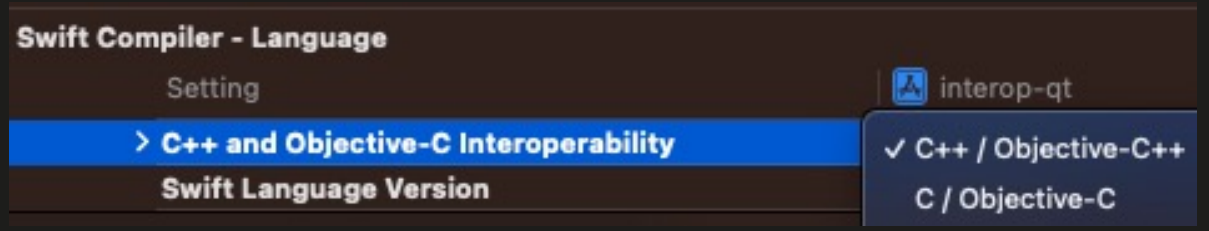
Swift target

C++ framework

Apple Clang Module Verifier - Options

Setting	AdminKit
Enable Module Verifier	Yes
Supported Language Dialects	gnu17 gnu++20
Supported Languages	objective-c++

Swift target



Демонстрация

Итоги

- В целом можно пользоваться
- Настроить Xcode проект просто
- Придется подождать пару лет, если в C++ API используется std::shared_ptr
- Придется ждать светлого будущего, если в C++ API используется «современный C++»
- Ждем потенциальных улучшений на WWDC 2024

Справочные материалы

https://gist.github.com/kambala-decapitator/3373f2dbaaa9f2ab6c0fb74868271a58

Спасибо за внимание!

Андрей Филипенков Senior iOS dev @kambala_decapitator

kaspersky