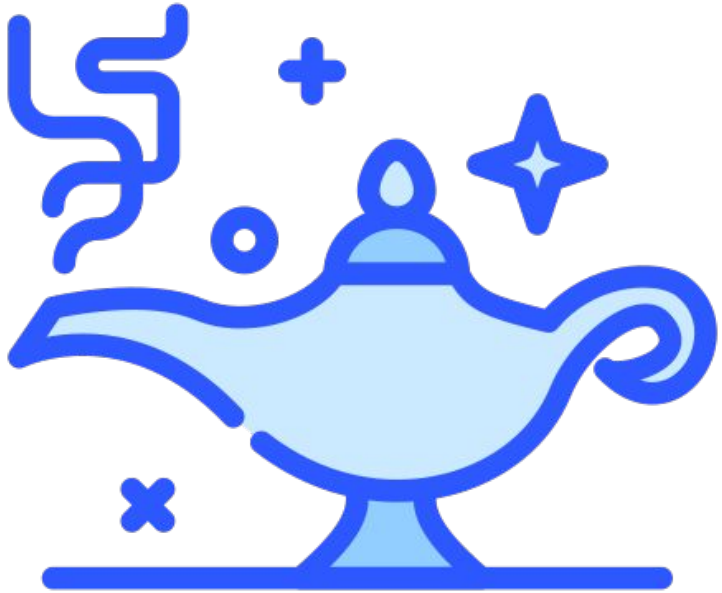
saby

IDL PyDjinni

Система генерации межъязыковых мостов



спикер

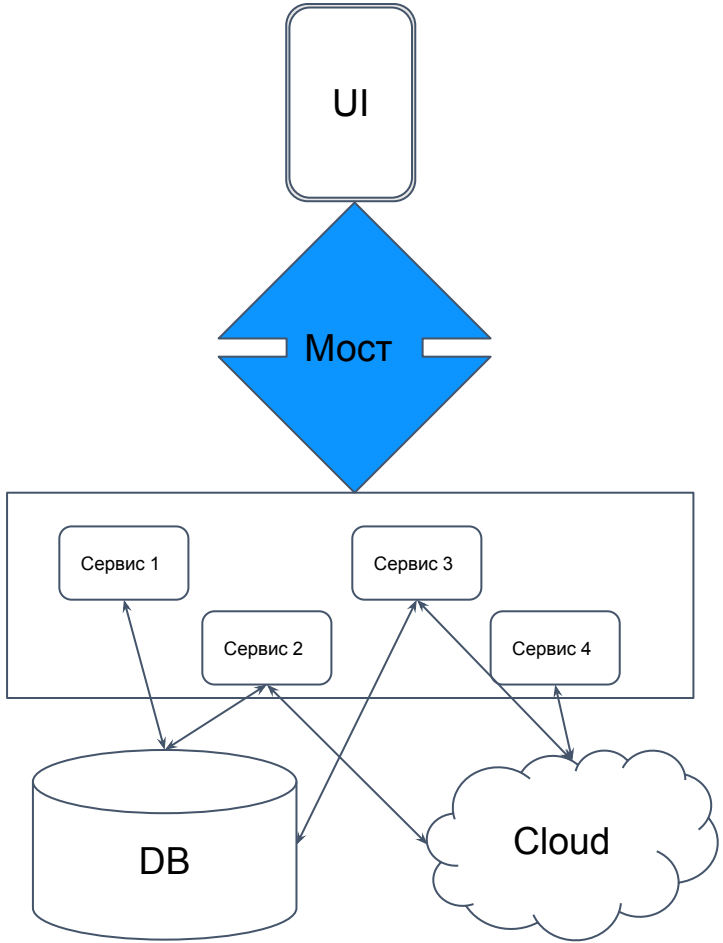
Глеб Игумнов, Тензор

вед. программист мобильной платформы

saby

Архитектура МП

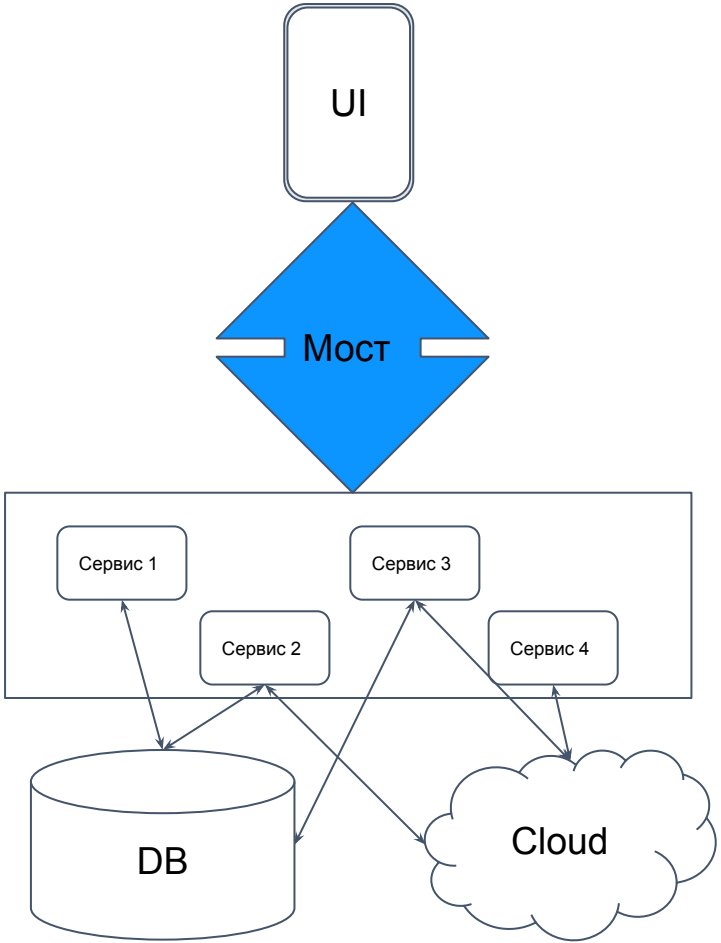
- Мобильное приложение состоит из трёх слоёв - контроллер на C++, нативный слой UI и мосты.
- Плюсы: не надо писать одну и ту же логику трижды (iOS, Android и десктоп).
- Минусы: надо писать межъязыковые мосты



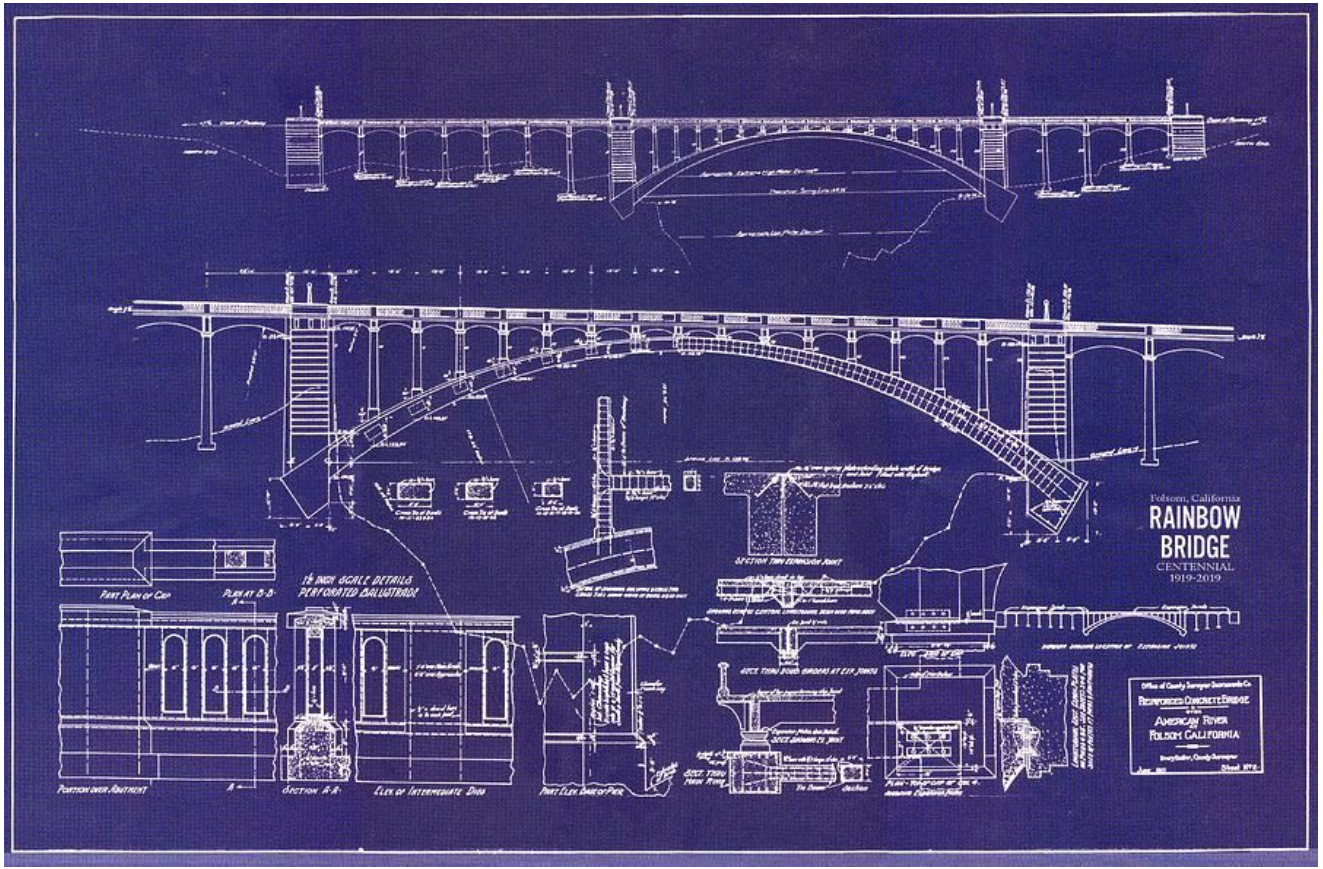
saby

Что передаём?

- Вызовы функций
- Данные
- Исключения



Устройство моста



saby

Устройство моста под Android

- Для межъязыковых вызовов используется JNI (Java Native Interface)
- Функции с реализацией на стороне Java/Kotlin вызываются через библиотеку JNI
- Функции с реализацией на стороне C++ объявляются в нативном коде **external**

saby

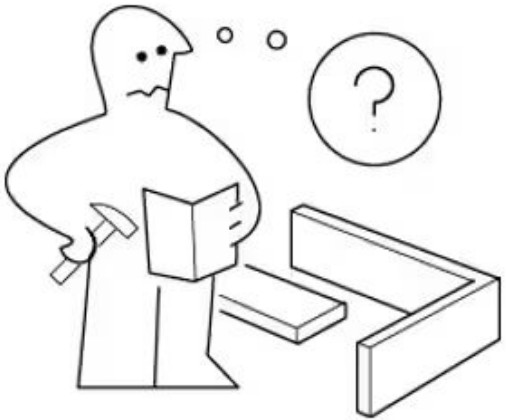
Устройство моста под iOS

- C++ <==> ObjC++ <==> ObjC <==> Swift
- Преобразование типов между ObjC и ObjC++ происходит через static_cast
  Благодаря тому, что все классы будут унаследованы от NSObject и у них совпадают сигнатуры методов — это работает
- Часть классов может остаться на уровне ObjC и импортироваться в свифт автоматически, но это может приводить к дополнительным проблемам, например с nullable-перечислениями.

saby

Общие проблемы мостостроения

- Надо знать все языки
- Много boilerplate’a
- Сложный код с кучей мест для ошибок



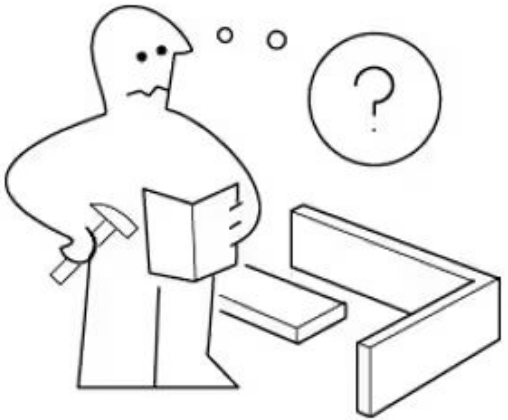
Всего один вызов через JNI

```
1.  CJNIEXPORT jstring JNICALL Java_ru_tensor_sbis_desktop_iauth_1service_generated_AuthService_00024CppProxy_native_1authByPhone(JNIEnv* jniEnv, jobject /*this*/, jlong nativeRef, jstring j_phone)
2.
3.  {
4.      try
5.      {
6.          DJINNI_FUNCTION_PROLOGUE1(jniEnv, nativeRef);
7.          auto scoped_ctx = sbis::log::ThreadContext::SetScopedWithMethodName ( sbis::GenerateUUIDRandomDevice(), L"[sbis-auth-service300:djinni][AuthService.AuthByPhone]"_sv, 1 );
8.          auto thread_registrator = ::djinni::ThreadInfoRegistrator( ::djinni::ThreadType::NATIVE );
9.          std::optional< sbis::blcore::ThreadDataUse > data_use;
10.         if( thread_registrator )
11.         {
12.             sbis::blcore::ThreadData thread_data;
13.             data_use.emplace( thread_data.PrepareThreadIfNeeded( true ) );
14.         }
15.         auto const start_time = sbis::DateTime::Now();
16.         LogMsg( sbis::logging::llDEBUG, L"[m][start] AuthService.AuthByPhone"_sv );
17.         if( _raw_NativeAuthService ) {
18.             auto& ref = ::djinni::objectRefFromHandleAddress<::sbis::desktop::iauth_service::AuthService>(nativeRef);
19.         auto r = ref->AuthByPhone(::djinni::WString::toCpp( jniEnv, j_phone ));
20.         LogMsg( sbis::logging::llDEBUG, L"[m][finish] AuthService.AuthByPhone"_sv, ( sbis::DateTime::Now() - start_time ).TotalMilliseconds(), L"ms"_sv );
21.         return ::djinni::release(::djinni::WString::fromCpp( jniEnv, r ));
22.         }
23.         else {
24.             const auto& ref = ::djinni::objectFromHandleAddress<::sbis::desktop::iauth_service::AuthService>(nativeRef);
25.         auto r = ref->AuthByPhone(::djinni::WString::toCpp( jniEnv, j_phone ));
26.         LogMsg( sbis::logging::llDEBUG, L"[m][finish] AuthService.AuthByPhone"_sv, ( sbis::DateTime::Now() - start_time ).TotalMilliseconds(), L"ms"_sv );
27.         return ::djinni::release(::djinni::WString::fromCpp( jniEnv, r ));
28.         }
29.     }
30.     catch (const ::sbis::desktop::iauth_service::LoginException& excepCpp)
31.     {
32.         const ::djinni::GlobalRef<jclass> clazz = ::djinni::jniFindClass("ru/tensor/sbis/desktop/iauth_service/generated/LoginException");
33.         const jmethodID jconstructor = ::djinni::jniGetMethodID(clazz.get(), "<init>", "(ILjava/lang/String;Ljava/lang/String;)V");
34.         auto excepJava = ::djinni::LocalRef<jobject>{jniEnv->NewObject(clazz.get(), jconstructor,
35.                                                                        ::djinni::get(::djinni::I32::fromCpp( jniEnv, excepCpp.ErrorCode() )),
36.                                                                        ::djinni::get(::djinni::WString::fromCpp( jniEnv, excepCpp.ErrorMessage() )),
37.                                                                        ::djinni::get(::djinni::WString::fromCpp( jniEnv, excepCpp.ErrorUserMessage() )))};
38.         ::djinni::jniExceptionCheck(jniEnv);
39.         jniEnv->Throw(static_cast<jthrowable>(excepJava.get()));
40.         return 0 /* value doesn't matter */;
41.     }
42. // тут еще куча catch-ей
43.     JNI_TRANSLATE_EXCEPTIONS_RETURN(jniEnv, 0 /* value doesn't matter */)
44. }
```

saby

Общие проблемы мостостроения

- Надо знать все языки
- Много boilerplate’a
- Сложный код с кучей мест для ошибок
- Преобразование типов может быть крайне неэффективно
- Невозможно использовать мосты в C++ корутинах



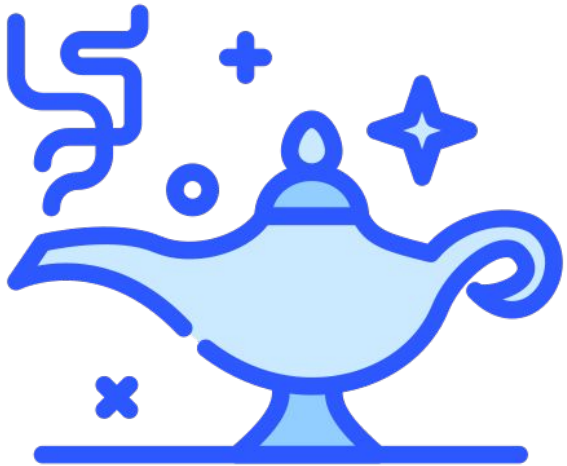
Android:

- Проблема с загрузкой классов в JNI в C++ потоке

iOS:

- Вопрос поставки swift-модулей

saby



PyDjinni

saby

Dropbox Djinni

- Был нестабилен
- Поддерживает только ObjC и Java
- Требует Scala-разработчиков
- Было решено переписать на Python

saby

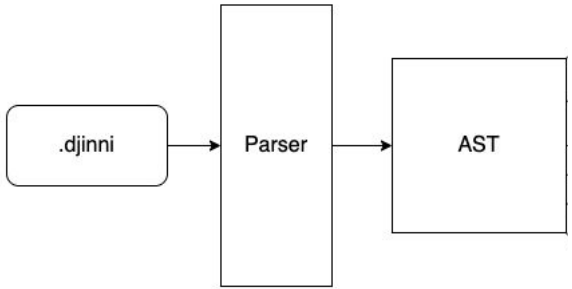
Код на PyDjinni

- Лёгкий в написании псевдокод
- Содержит все основные типы
- Поддерживает модульность
- Генерация boilerplate кода не только в мостах. Автоматически генерируются операторы сравнения, сериализации и десериалиации и т.д.

```
@package auth
user_type = enum {
  user;
  admin;
}

user = record {
  id: uuid;
  name: string;
  type: user_type;
}

bad_password_exc = exception
{
}

auth_service = interface +c {
  auth_by_password( login: string, password: string ): user
    throws bad_bassword_exc;
  event on_login( user_id: uuid );
}
```

saby

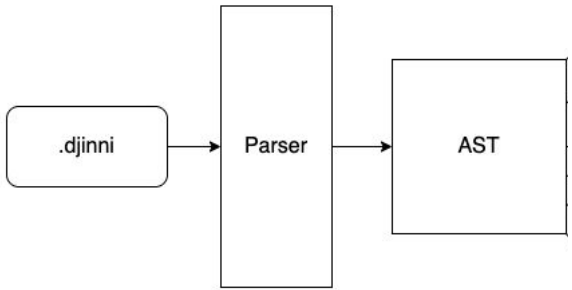
Парсинг псевдокода



- Построение дерева типов из псевдокода
- Построен на основе PLY (Python Lex-Yacc)
- На этом этапе не учитываются связи типов

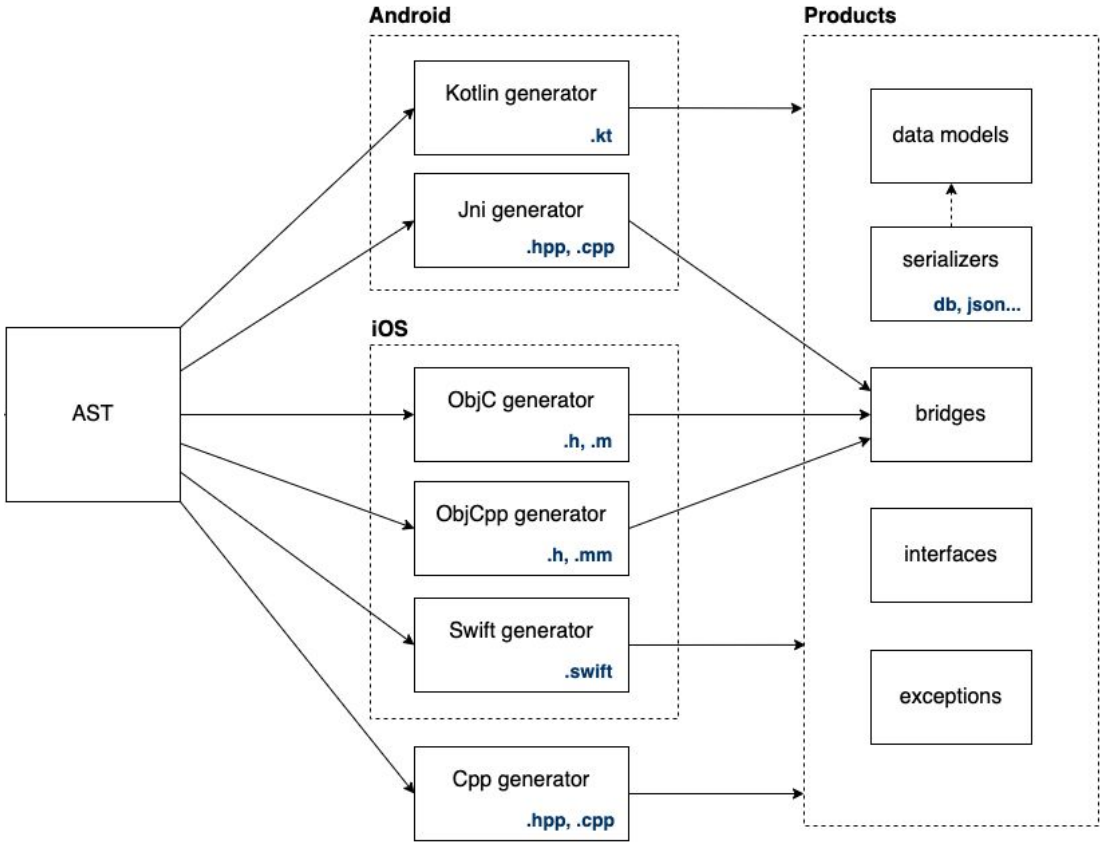
saby

Разрешение типов



- Работает с уже имеющимся деревом типов
- Разрешает зависимости
- Генерирует дополнительные типы (шаблоны, события)
- Ищет ошибки

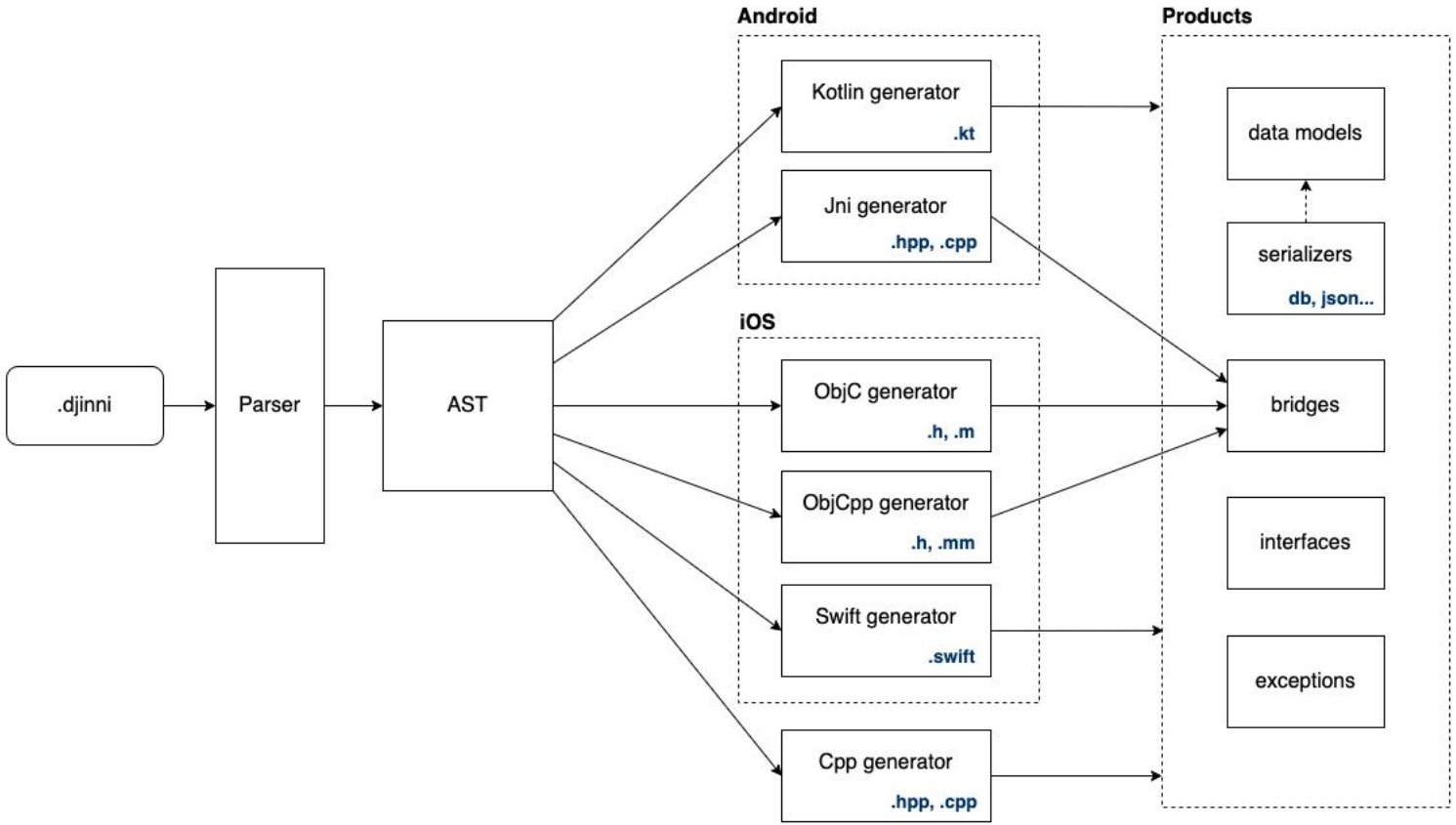
Генерация



- Отдельные модули под каждый язык
- Классы для генерации типов, которые могут быть нужны в “соседних” языках
- Генерирует итоговые артефакты – программный код, документацию в json и т.д.

saby

Архитектура PyDjinni



saby

Интерфейсы

- Не содержат данных
- Могут обладать реализацией как в контроллере, так и в нативной части
- Не копируются. Через мост передаётся указатель на прокси-объект.
- Для получения указателя используются статические фабричные методы, либо методы-регистраторы.

```
# Интерфейс с реализацией в нативной части
auth_callback = interface +j +o {
   on_login( user_id: uuid );
}

# Интерфейс с реализацией в C++ части
auth_service = interface +c {
  # Фабричный метод для получения экземпляра класса
  static instance(): auth_service;

  # Метод для регистрации нативного объекта коллбека
  register_callback( cb: auth_callback );

  # Прочие методы
  auth_by_password( login: string, password: string ): user
    throws bad_bassword_exc;
}
```

saby

Модели и перечисления

- Данные копируются между языками, модели не изменяются синхронно
- Не разрешаем интерфейсы ради использования в межпроцессных вызовах

```
@package auth
# Простое перечисление
user_type = enum {
  user;
  admin;
}
# Перечисление - битовая маска
contact_type = enum_flags {
  mobile;
  email;
  vk;
}
# Модель данных
user = record {
  id: uuid;
  name: string;
  type: user_type;
  available_contacts: contact_type;
  contacts: map< contact_type, string >;
}
```

saby

Битовые маски C++

- Частый прикладной сценарий – множество bool полей в модели, что замедляет конвертацию.
- В C++ используется собственный тип, с целым числом внутри и enum по степеням двойки для удобства использования.

```
file_properties = enum_flags {
  ## во внешнем хранилище или локальном
  is_external;
  ## архив или нет
  is_zip;
  ## картинка или нет
  is_image;
}
```

C++:

```
FileProperties our_props;
// Добавление флага
our_props |= FilePropertiesFlags::IsZip;
our_props |= FilePropertiesFlags::IsImage;
// Удаление флага
our_props &= ~FilePropertiesFlags::IsImage;
// Проверка наличия флага
if( our_props & FilePropertiesFlags::IsZip ) { ... }
```

saby

Битовые маски Swift

- Частый прикладной сценарий – множество bool полей в модели, что замедляет конвертацию.
- В ObjC генерируется enum NS_OPTIONS, который автоматически импортируется в Swift как Optionset

```
file_properties = enum_flags {
  ## во внешнем хранилище или локальном
  is_external;
  ## архив или нет
  is_zip;
  ## картинка или нет
  is_image;
}
```

Swift:

```
let opt: FileProperties
// Вставка значения
opt.insert(.isZip)
opt.insert(.isImage)
// Удаление флага
opt.remove(.isZip)
// Проверка флага
if opt.contains(.isZip) { ... }
```

saby

Битовые маски Kotlin

- Частый прикладной сценарий – множество bool полей в модели, что замедляет конвертацию.
- В Kotlin пришлось написать свой enum со значением и обёртку над ним

Kotlin:

```
interface EnumWithValue {
   val value: Long
}

enum class FilePropertiesFlags: EnumWithValue
{
   IS_EXTERNAL{ override val value = 1L },
   IS_ZIP{ override val value = 2L },
   IS_IMAGE{ override val value = 4L },
}

open class EnumFlags< T: EnumWithValue >( var rawValue: Long ) {
   infix fun contains( flag: T ): Boolean
   infix fun append( flag: T )
   infix fun remove( flag: T )
}
```

saby

Битовые маски Kotlin

- Частый прикладной сценарий – множество bool полей в модели, что замедляет конвертацию.
- В Kotlin пришлось написать свой enum со значением и обёртку над ним

Kotlin:

```
val our_props: FileProperties
// Добавление флага
our_props append FilePropertiesFlags.IS_ZIP
our_props append FilePropertiesFlags.IS_IMAGE
// Удаление флага
our_props remove FilePropertiesFlags.IS_IMAGE
// Проверка наличия флага
if( our_props contains FilePropertiesFlags.IS_ZIP ) { ... }
```

saby

Variant

- Ещё один частый сценарий – модель с взаимоисключающими optional полями
- В C++ используется std::variant
- В Swift - enum со значением

```
# Категория в меню.
menu_folder = record { id: uuid; }
# Блюдо
menu_item = record { name: string; }

item_variant = variant< menu_folder, menu_item>

# Позиция в меню.
menu_list_item = record {  value: item_variant; }
```

saby

Variant

- Ещё один частый сценарий – модель с взаимоисключающими optional полями
- В C++ используется std::variant
- В Swift - enum со значением
- В Kotlin - генерируемый класс с методами определения типа и получения значения

Kotlin:

```
// Задание значения variant'a
var item: MenuListItem
item.value.setValue( MenuItem("имя блюда") )

// Проверка конкретного типа.
item.value.isMenuListItemFolder()

// Равноценный способ проверки.
item.value.isOfType( MenuListVariant.typeMenuListItemElement )

// Получение значения
when( item.value.getType() )
{
   MenuListVariant.typeMenuFolder ->
      item.value.getValue() as MenuFolder
   MenuListVariant.typeMenuItem ->
      item.value.getValue() as MenuItem
}
```

saby

Исключения Android

- На Android передача исключений идёт через стандартные механизмы JNI
- Есть проблема при переполнении стека

```
#Создадим два исключения
custom_djinni_exception = exception {}

other_custom_djinni_exception = exception {
  custom_code: i32;
}
#Создадим интерфейс и пометим, что он может выбросить эти исключения
our_iface = interface +c {
   instance(): our_iface;
   do_throw(): throws
      custom_djinni_exception,
      other_custom_djinni_exception;
}
```

saby

Выброс исключений iOS

- В ObjC нет типизированных исключений, мы передаём информацию через NSError::domain
- На Swift генерируется обработчик, который типизирует исключения обратно

```
public func `doThrow`() throws
{
  var error : NSError? = nil
  let result = self.objc.__doThrow(error: &error)
  guard error == nil else
  {
    switch error.domain
    {
      case "djinni.package.customDjinniException":
        throw CustomDjinniException(error: error)
      case "djinni.package.customDjinniException":
        throw CustomDjinniException(error: error)
      default:
        throw SbisException(error: error)
    }
  }
  return result
}
```

saby

Перехват исключений iOS

- Все сгенерированные исключения наследуются от CustomNSError в случае необходимости работы с ними как с NSError

```
do {
  try doThrow()
} catch is CustomDjinniException {
   // Нам достаточно проверить тип исключения
} catch let exc as OtherCustomDjinniException{
   // Мы можем получить поле исключения из exc
   let code = exc.custom_code
}
catch let exc as NSError {
   // Перехват исключений из сторонних библиотек
}
```

saby

Наследование

- “Нечестное” наследование: в дочерний класс копируются все методы и поля родительского
- Реального наследования в сгенерированном коде нет, кроме исключений.

```
# Родительский класс
proud_parent = interface +c {
   be_proud();
 }

# Дочерний класс
humble_child = interface +c: proud_parent {
   be_humble();
}
```

saby

Шаблоны

- Реализация шаблона с помощью наследования
- Возможна void-специализация шаблонных методов, в таком случае void-параметр удаляется из сигнатуры метода

```
# Результат выборки списочного метода list
list_result< T > = record {
   # Список моделей полученных по фильтру
   result: list< T >;
   # Признак того, что в базе есть ещё модели по данному фильтру
   have_more: bool;
}

# Интерфейс источника данных
data_source<T, F > = interface +c {
  # @brief Выборка записей по фильтру f без запроса в облако
  refresh(f: F): list_result< T >;
 }

# Реализация интерфейса с реальными типами модели и фильтра
our_data_source = interface +c: data_source< model, filter >
{
}
```

saby

Кэширование JVM

- Из C++ потоков невозможно получить экземпляр JVM с нашими классами, поэтому мы кэшируем экземпляр при инициализации из мобильного кода.
- В мостах под Android при первом создании прокси-объекта в C++ мы кэшируем в нём все данные о соответствующем классе и методах из JVM

saby

Кэширование указателей

- Указатель на интерфейс одного языка может удерживаться в другом языке, для этого объекты мостов хранят сильные указатели на “настоящий” объект
- Один реальный объект – один мост. За это отвечает кэш с парой слабых указателей.

	C++	ObjC	JNI
Сильный указатель	std::unique_ptr	__strong NSObject	jobject
Слабый указатель	std::weak_ptr	__weak NSObject	java.lang.ref.WeakReference

saby

Мосты и корутины

- Корутинизация C++ потоков позволяет повысить эффективность работы приложения.
- Подмена стека приводит к беде в нативных вызовах
  - JNI проверяет переполнение стека относительно потока и убивает приложение
  - В ObjC @autoreleasepool использует thread-local хранилище, при корутинизации потока данные в thread-local хранилище перепутываются и это приводит к падению
- Решение: выносим мост в отдельный поток и синхронно ожидаем его исполнения в вызывающем C++ коде

saby

Направления развития

- Прямой мост в Swift
- Мосты в NodeJS
- Честное наследование
- Честные шаблоны

saby

Спасибо за внимание

Глеб Игумнов, Тензор

вед. программист мобильной платформы