Интерфейсы в программировании

Алексей Островский

Физико-технический учебно-научный центр НАН Украины

2 апреля 2015 г.

Интерфейсы в программировании

Определение

Интерфейс — связь между двумя отдельными компонентами программной системы, предназначенная для обмена информацией, а также спецификация этой связи.

Виды интерфейсов:

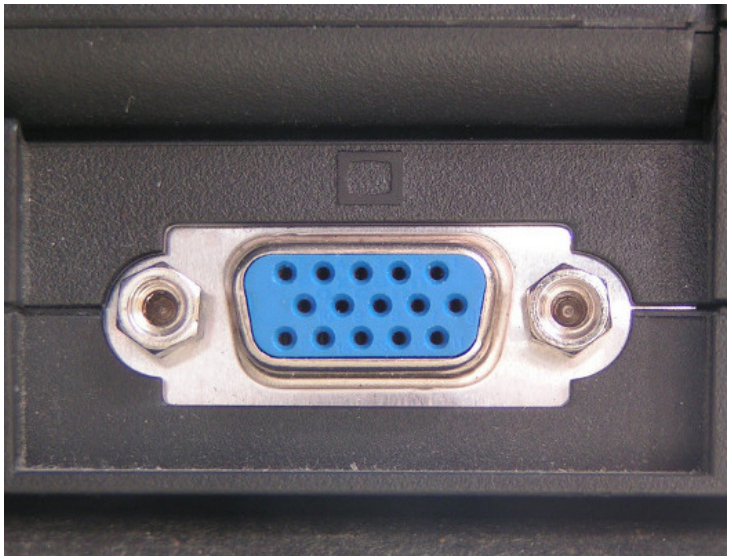
- аппаратные (в т. ч. для периферийных устройств);
- программные (англ. *software interfaces*);
- пользовательские.

NB. Software interface $\supset$ (application) programming interface.

Направленность передачи данных:

- однонаправленные интерфейсы (напр., работа с мышью, микрофоном и другими устройствами);
- двунаправленные интерфейсы (напр., большинство программных).

Аппаратные интерфейсы



Примеры аппаратных интерфейсов: SVGA-вход для подключения видеоустройства и универсальная последовательная шина (USB).

Аппаратные интерфейсы

Виды: шины, устройства хранения данных, радиоволны, ...

Спецификация:

- характеристики механических, электрических или логических сигналов, пересылаемых через интерфейс;
- последовательность передачи сигналов.

Цель:

- унификация связи с базовыми и периферийными устройствами;
- большие возможности при конструировании и сборке компьютерных систем.

Программные интерфейсы

Уровень операционной системы:

- связь программ с оборудованием (напр., с помощью драйверов);
- связь программ с операционной системой или с другими программами средствами ОС.

Уровень приложения:

- связь компонентов системы, написанных на разных ЯП, с помощью вспомогательной среды (middleware);
- связь компонентов распределенных приложений.

Уровень компонента:

связь между базовыми элементами программы (напр., между объектами в ООП; между функциями в функциональном программировании).

Уровень детализации интерфейса

Двоичные интерфейсы (ABI) — спецификации, связанные с *машинным* кодом программной системы.

Примеры:

- размер в байтах типов данных;
- особенности вызовов функций / методов;
- формат двоичных исполняемых файлов.

Программные интерфейсы (API) — спецификации, выражающиеся через *исходный* код программы.

Пример:

- сигнатуры задекларированных функций / классов;
- протоколы обмена данными.

ABI

Определение

Двоичный интерфейс приложений (англ. *application binary interface, ABI*) — программный интерфейс между двумя модулями (чаще всего, ОС или библиотекой и прикладной программой), заданный на уровне машинного кода.

Примеры спецификаций:

- вызов функций (порядок аргументов, размещение аргументов в стеке / регистрах);
- формат передачи данных (порядок передачи байтов, выравнивание);
- системные вызовы;
- формат исполняемых файлов, библиотек и т. п.;
- обработка исключительных ситуаций.

ABI

Способы согласования ABI:

- (в большинстве случаев) автоматически при компиляции / интерпретации программы;
- при помощи специальных средств ЯП (напр., спецификаторы `safecall`, `stdcall` и т. п. для функций в C/C++);
- явно при использовании низкоуровневых средств взаимодействия между программами на разных ЯП или при работе с оборудованием (напр., чтение / запись данных в память видеоускорителя).

Примеры ABI

- Спецификация **операционных систем**: Linux (Linux Standard Base), Windows, Mac OS;
- спецификация **виртуальных машин** и **сред выполнения**, напр., Java Virtual Machine:
  - общая для всех имплементаций (напр., формат .class-файлов JVM);
  - особенности имплементации в различных средах (напр., размеры указателей JVM и принципы сборки мусора).
- спецификация **языков программирования** (напр., размер, двоичное представление и выравнивание типов данных);
- EABI (embedded-application binary interface) — ABI для **встроенных** в микропроцессор **программ**.

API

Определение

Программный интерфейс приложений (англ. *application programming interface, API*) — спецификация утилит, протоколов и инструментов для интеграции компонентов приложений.

Примеры спецификаций:

- характеристики компонентов повторного использования (операции, входные и выходные данные);
- точки входа для расширения функциональности существующих приложений (плагины);
- программные «обертки» для доступа к оборудованию и источникам данных (напр., СУБД).

API

Формы API:

- Библиотека, содержащая функции / объекты, структуры данных, константы и переменные.

  Примеры: STL для C++; Java API.

- Протокол для передачи данных.

  Примеры: протоколы SOAP и REST для реализации веб-сервисов.

Формы спецификации:

- международный стандарт (напр., POSIX);
- документация производителя, независимая от ЯП + привязки к ЯП (напр., Windows API, OpenGL);
- встроенная / подключаемая библиотека на определенном ЯП (STL, Java APIs).

Сравнение API и ABI

API и ABI в C:

```c
#include<stdio.h>

// Определение функции printf в stdio.h — часть API
// и одинаково для всех режимов компиляции.
// int printf(const char* format, ...);

// Определение размеров типов данных — часть ABI.

int main() {
    printf("sizeof(int) == %zd\n", sizeof(int));
    printf("sizeof(long) == %zd\n", sizeof(long));
    printf("sizeof(long double) == %zd\n", sizeof(long double));
    printf("sizeof(void*) == %zd\n", sizeof(void*));
    return 0;
}
```

Сравнение API и ABI

- GCC с опцией `-m32` (32-битный код для архитектуры i386):

```
sizeof(int) == 4
sizeof(long) == 4
sizeof(long double) == 12
sizeof(void*) == 4
```

- GCC с опцией `-mx32` (32-битный код для архитектуры x86_64):

```
sizeof(int) == 4
sizeof(long) == 4
sizeof(long double) == 16
sizeof(void*) == 4
```

- GCC с опцией `-m64` (64-битный код для архитектуры x86_64):

```
sizeof(int) == 4
sizeof(long) == 8
sizeof(long double) == 16
sizeof(void*) == 8
```

Использование интерфейсов

Варианты использования:

- Интеграция компонентов, реализованных с помощью *одного* ЯП.

  Реализация:

  - спецификация типов данных;
  - контрактное программирование (напр., с помощью интерфейсов ЯП или утиной типизации);
  - инверсия управления.

- Интеграция компонентов, реализованных в *разных* ЯП.

  Реализация:

  - использование общей исполняемой среды (виртуальной машины);
  - вызов внешних функций;
  - использование посредников (middleware).

Пример: использование интерфейсов ЯП

Процедурный подход

Интерфейсы ЯП определяются как спецификация структур данных и функций для работы с ними.

Пример (C):

```c
// Файл заголовков math.h определяет интерфейсы математических функций.s
#include <math.h>

// Функция sqrt определена в math.h декларацией
// double sqrt(double X);

int main() {
    double s = sqrt(9.0);
    // другой код
}
```

Пример: использование интерфейсов ЯП

Объектно-ориентированный подход

Интерфейсы собраны в методах (конструкторах, свойствах и т. п.) объектов и классов, содержащих в себе данные и зачастую скрывающих их.

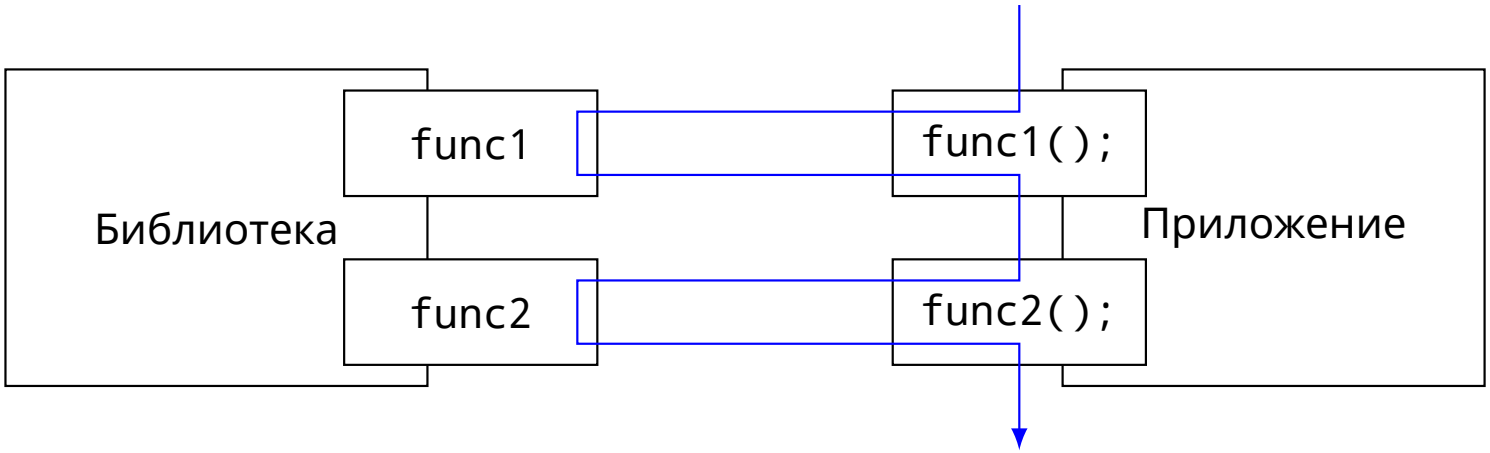
Пример (Java)

```
1  /*
2   * Класс java.math.BigInteger определяет интерфейсы методов
3   * для работы с произвольными целыми числами.
4   */
5  import java.math.BigInteger;
6
7  // в функции main
8  BigInteger num = new BigInteger("12345678987654321");
9  num = num.multiply(num);
```

Инверсия управления

Определение

Инверсия управления (англ. *inversion of control*) — принцип проектирования приложений, позволяющий передавать управление коду приложения во время выполнения сторонних библиотек.

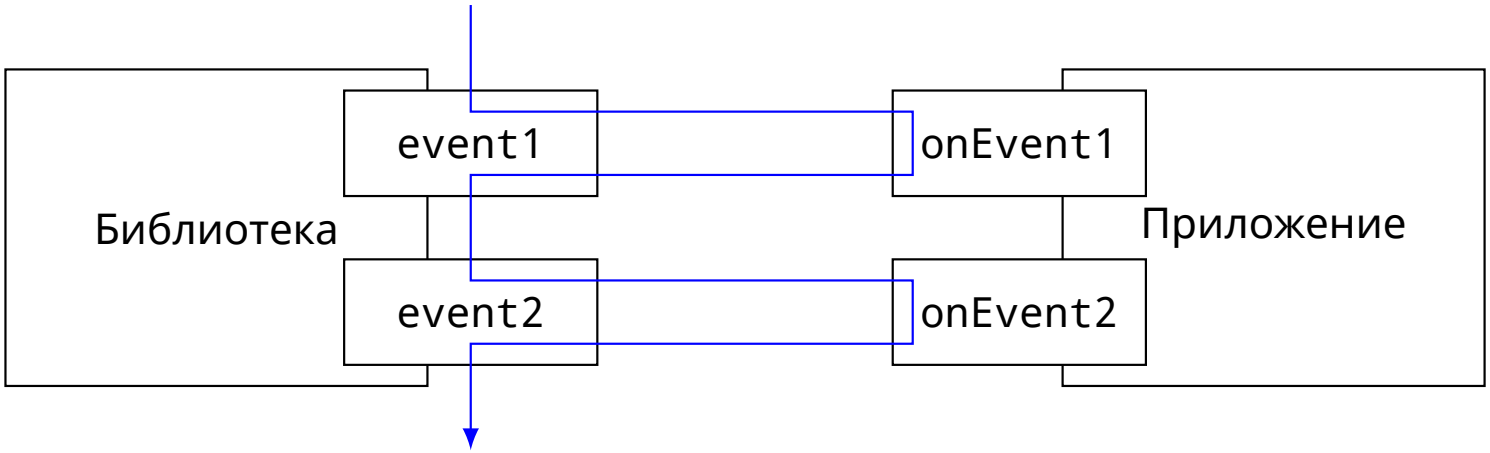


Традиционный подход к работе с библиотеками. Библиотека определяет интерфейсы реализованных функций, которые вызываются приложением. Синим обозначен поток выполнения.

Инверсия управления

Определение

Инверсия управления (англ. *inversion of control*) — принцип проектирования приложений, позволяющий передавать управление коду приложения во время выполнения сторонних библиотек.



Инверсия управления. Основной код выполняется библиотекой; передача управления приложению происходит с помощью точек расширения.

Применение инверсии управления

- **Обработка событий**, в частности при конструировании пользовательского интерфейса;
- **расширение функциональности** приложений (плагины);
- **упрощение построения** приложений на основе готовых каркасов (например, JavaBeans; аннотации в JUnit);
- **наследование** в ООП, когда контроль выполнения задан в родительском классе (точки входа — абстрактные виртуальные методы).

Пример: инверсия управления в DOM

Инверсия контроля — основной метод создания интерактивных HTML-документов с помощью JavaScript.

```
1  var docBody = document.getElementsByTagName('body')[0];
2  // Добавление обработчика события с помощью
3  // метода addEventListener.
4  docBody.addEventListener('load', function() {
5      function onBtnClick() {
6          // код, выполняемый при нажатии кнопки
7          setTimeout(function() {
8              // код, выполняемый с задержкой в две секунды
9              alert('Notification');
10         }, 2000);
11     }
12
13     // код, выполняемый при загрузке HTML-страницы
14     document.getElementById('some-button')
15         .addEventListener('click', onBtnClick);
16 });
```

Взаимодействие разноязыковых программ

Методы взаимодействия (по возрастанию усилий для реализации):

- вызов функций / методов в пределах одной среды выполнения (JVM, CLR);
- использование интерфейса внешних функций;
- использование распределенных систем (CORBA, RMI)[1]

Проблемы (следуют из различия ABI для разных ЯП):

- трансляция типов данных;
- различающиеся среды выполнения (напр., принципы сбора мусора);
- работа с разделяемой памятью.

[1] рассматриваются в отдельной лекции.

Виртуализация

Определение

Виртуальная машина (англ. *virtual machine*) — программная среда для эмуляции вычислительной системы с заданной конфигурацией и интерфейсом.

Цели:

- унификация доступных для программ интерфейсов в различных окружениях, улучшение переносимости;
- повышение отказоустойчивости и надежности;
- упрощение взаимодействия между программами в пределах ВМ;
- тестирование поведения ПО при заданной архитектуре или конфигурации оборудования.

Спецификация ВМ

- **Модель вычислений:** стековая, регистровая;
- **управление памятью:** автоматическое, вручную, смешанное, принцип работы сборщика мусора;
- **метод выполнения кода:** интерпретация, JIT, компиляция); принципы комбинирования различных методов;
- **взаимодействие модулей:** трансляция типов данных для различных ЯП, совместимость объектов в различных языках, работа с нативными библиотеками;
- **безопасность:** возможность ограничений на выполняемые инструкции.

Примеры ВМ

- **Java Virtual Machine**

  ЯП: Java, Groovy, Scala, Clojure, Jython, JRuby, ...

- **Dalvik; Android Runtime (ART)**

  ЯП: поддерживаемые JVM (вход — байткод для JVM).

  Примечание: Dalvik / ART и JVM почти совместимы на уровне API, но используют разные ABI (регистровая машина в Dalvik / ART против стековой в JVM).

- **Common Language Runtime; Mono**

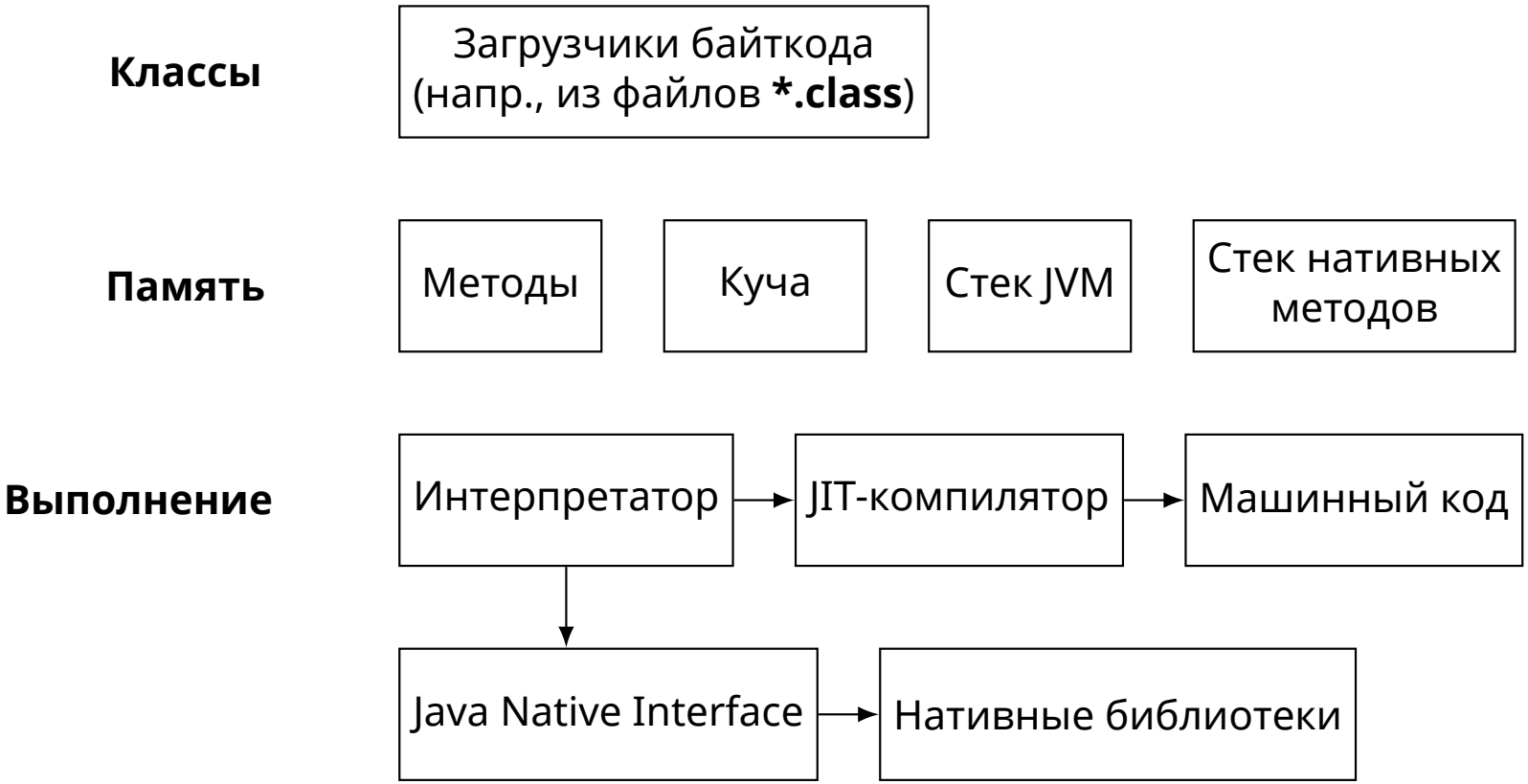
  ЯП: C#, F#, Visual Basic .NET, IronPython, ...

- **LLVM** (Low Level Virtual Machine)

  ЯП: C, C++, D, Objective-C, Ada, Fortran.

  Примечание: часто используется в качестве компилятора.

Пример ВМ: Java Virtual Machine



Пример: взаимодействие программ в JVM

- Языки, созданные **с расчетом на JVM** (Groovy, Scala) поддерживают импорт произвольных классов Java (из стандартной библиотеки и сторонних).
- Версии **адаптированных** для JVM языков (Jython, JRuby) поддерживают классы Java с определенными предостережениями из-за различия спецификаций.

  Пример. Конфликты между Java и Jython:
  - кодировки строк;
  - области видимости;
  - названия переменных и методов класса, совпадающие между собой или с ключевыми словами Python.
- Использование языков JVM **из Java** возможно за счет увеличения сложности структуры программ (напр., применение фабрик объектов или Java Scripting API для Jython).

Интерфейс внешних функций

Определение

Интерфейс внешних функций (англ. *foreign function interface*) — механизм поддержки вызова программой, написанной на одном ЯП, функций, реализованных на других ЯП.

Цели:

- повышение производительности;
- вызов специфичных для платформы функций.

Типичная архитектура интерфейса внешних функций

Примеры FFI

- Директивы `extern "C"` в C++ для вызова C-функций.
- Java Native Interface, Java Native Access.
- Модуль `ctypes` в Python для вызова функций C:

```
1  import ctypes
2
3  # Загрузка базовой библиотеки C
4  libc = ctypes.CDLL('libc.so.6')
5  # Вызов функции C time
6  # Эквивалентный вызов в C: t = time(NULL);
7  t = libc.time(None)
8  print t
```

Выводы

1. Интерфейсы — необходимая часть системы, составленной из разнородных компонентов. Интерфейс определяет способ обмена данными между компонентами программной системы.
2. Выделяют два типа программных интерфейсов: определяющие характеристики данных на уровне машинного кода (ABI) и на уровне языков программирования (API).
3. Интерфейсы применяются для связи компонентов в пределах одного ЯП (за счет спецификации типов данных, протоколов работы с ними, инверсии управления и т. д.) или для связи разноязычных компонентов.
4. Основные способы взаимодействия ЯП: общая среда выполнения (виртуальная машина); интерфейсы внешних функций; протоколы передачи данных с помощью посредников.

Материалы

Лавріщева К. М.

Програмна інженерія (підручник).

К., 2008. — 319 с.

Bloch, Joshua

Руководство по написанию API.

http://lcsd05.cs.tamu.edu/slides/keynote.pdf

Fowler, Martin

Inversion of Control

http://martinfowler.com/bliki/InversionOfControl.html

Спасибо за внимание!