Распределенные вычислительные системы

Перенос данных и кода

Вопросы

- Какие уровни клиент-серверной архитектуры вы знаете?
- Какие недостатки двухзвенной клиент-серверной архитектуры вы можете назвать?
- Как клиентское приложение узнает об интерфейсе удаленного метода?
- Назовите по шагам процесс вызова удаленной процедуры.

Обмен данными

Маршализация

- Для передачи параметров по сети используется **маршализация** (marshalling) - процесс преобразования параметров для передачи их между процессами при удаленном вызове
- Маршализация
  - **по ссылке** – экземпляр удаленного объекта находится на сервере и не покидает его, а для доступа используются посредники
  - **по значению** – удаленный объект сериализируется и его копия передается в другой процес

Форматы сериализации данных

- Текстовые (платформо-независимые):
  - XML
  - JSON
  - YAML
- Бинарные
  - Protocol Buffers (Google)
  - Byte stream (очень неэффективные):
    - java.io.Serializable interface
    - .NET Serializable attribute

XML vs JSON

XML

```
<person>
  <firstName>Иван</firstName>
  <lastName>Иванов</lastName>
  <address>
    <streetAddress>Московское ш., 101,
кв.101</streetAddress>
    <city>Ленинград</city>
    <postalCode>101101</postalCode>
  </address>
  <phoneNumbers>
    <phoneNumber>812 123-1234</phoneNumber>
    <phoneNumber>916 123-4567</phoneNumber>
  </phoneNumbers>
</person>
```

JSON

```
{
   "firstName": "Иван",
   "lastName": "Иванов",
   "address":  {
      "streetAddress": "Московское ш., 101,
кв.101",
      "city": "Ленинград",
      "postalCode": 101101
   },
   "phoneNumbers":  [
      "812 123-1234",
      "916 123-4567"
   ]
}
```

PROTOBUF

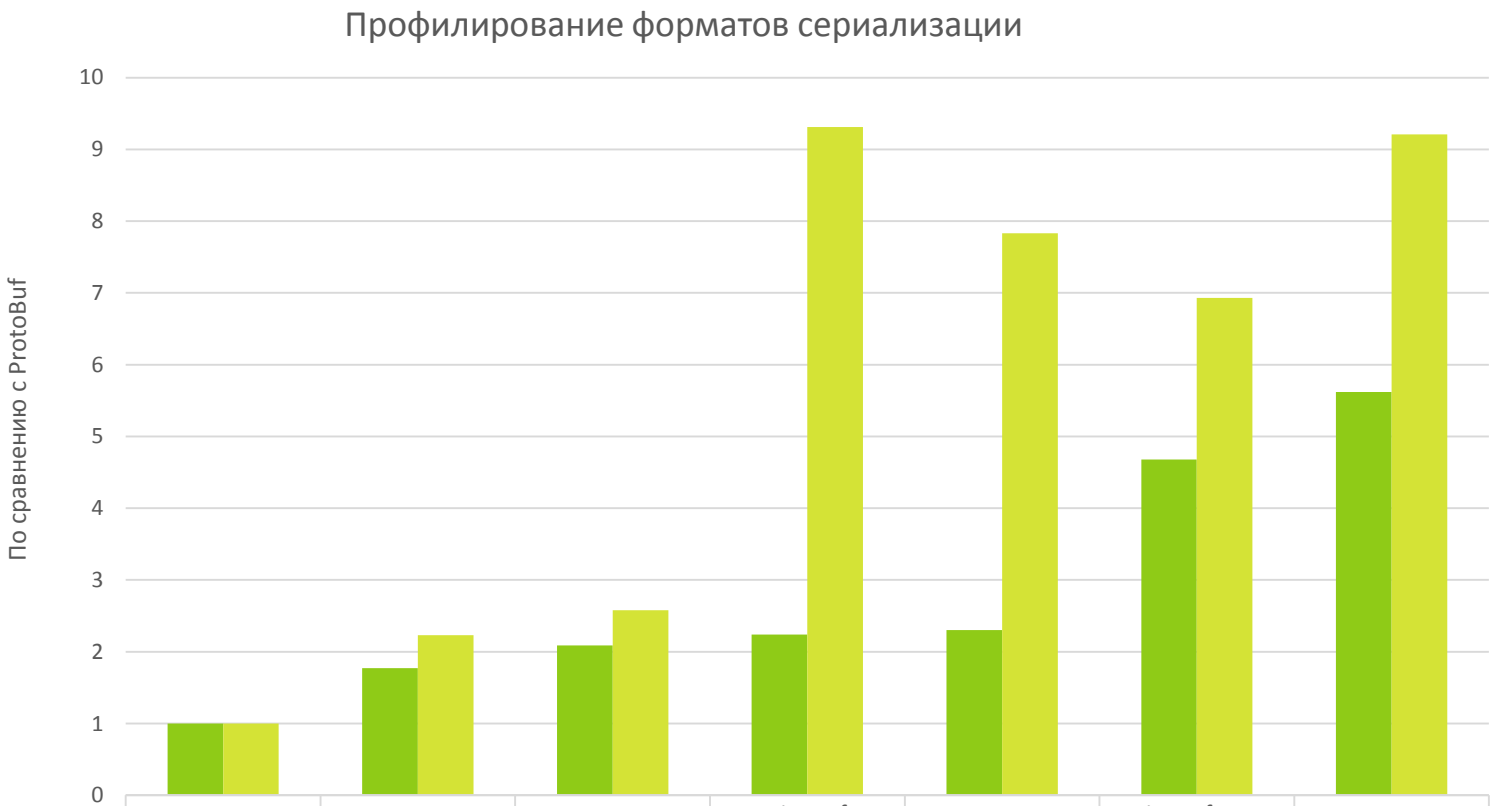
```
message Car {
    required string model = 1;

    enum BodyType {
        sedan = 0;
        hatchback = 1;
        SUV = 2;
    }

    required BodyType type = 2 [default = sedan];
    optional string color = 3;
    required int32 year = 4;

    message Owner {
        required string name = 1;
        required string lastName = 2;
        required int64 driverLicense = 3;
    }

    repeated Owner previousOwner = 5;
}
```

.proto - файл

Форматы обмена данными

- JSON
  - человеко-читаемый / редактируемый
  - может быть разобран без предварительного знания схемы
  - отличная поддержка браузеров (JavaScript native class description)
  - компактнее, чем XML
- XML
  - человеко-читаемый / редактируемый
  - может быть разобран без предварительного знания схемы
  - стандарт для многих протоколов (SOAP и т.п.)
  - хорошая инструментальная поддержка (XSD, XSLT, SAX, DOM и т.д.)
  - не компактный, большой объем «лишних» описаний
- Protobuf (Google)
  - очень компактный (плотная упаковка данных)
  - очень быстрая обработка
  - встроенная поддержка версий протокола (при изменении протокола, клиенты могут работать со старой версией, пока не обновятся)
  - без знания схемы очень тяжело разобрать (бинарный формат данных, внутренне не однозначен, требуется схема для разбора)
  - не предназначен для чтения/редактирования человеком

Форматы сериализации



	ProtoBuf.net	ServiceStack TypeSerializer	ServiceStack JsonSerializer	Microsoft JsonDataContract Serializer	NewtonSoft.Json	Microsoft XML DataContractSeria lizer	Microsoft BinaryFormatter
Больше (раз)	1	1,77	2,09	2,24	2,3	4,68	5,62
Медленнее (раз)	1	2,23	2,58	9,31	7,83	6,93	9,21

Basically protocol buffers (protobuf-net) is around **7x** quicker than the fastest Base class library Serializer in .NET (XML DataContractSerializer). Its also smaller than the competition as it is also **2.2x** smaller than Microsoft's most compact serialization format (JsonDataContractSerializer).

http://www.servicestack.net/benchmarks/NorthwindDatabaseRowsSerialization.100000-times.2010-08-17.html

Когда использовать какие форматы?

- XML
  - Если система предоставляет публичный API в виде XML Веб-сервиса (изучим их позже)
  - Работаете с «классической» системой, в которой уже используется XML в качестве стандарта обмена данными
  - Требуются стандартные инструменты верификации и трансформации (например, в HTML) (XSD, XSLT, SAX, DOM и т.д.)
- JSON
  - Если система предоставляет публичный API в виде REST-сервиса (изучим их позже)
  - Если клиенты, в большинстве своем, реализуются на JavaScript
  - Более компактный чем XML (в 2-2.5 раза) и самый простой для чтения/редактирования – соответственно, везде, где вы хотели бы использовать XML, подумайте, может быть стоит использовать JSON.
- Protobuf
  - Если для вас прежде всего важен объем и скорость обработки данных
  - Если важна возможность простого обновления протокола
  - Если реализуется внутренний (не публичный) протокол обмена

Методы переноса кода

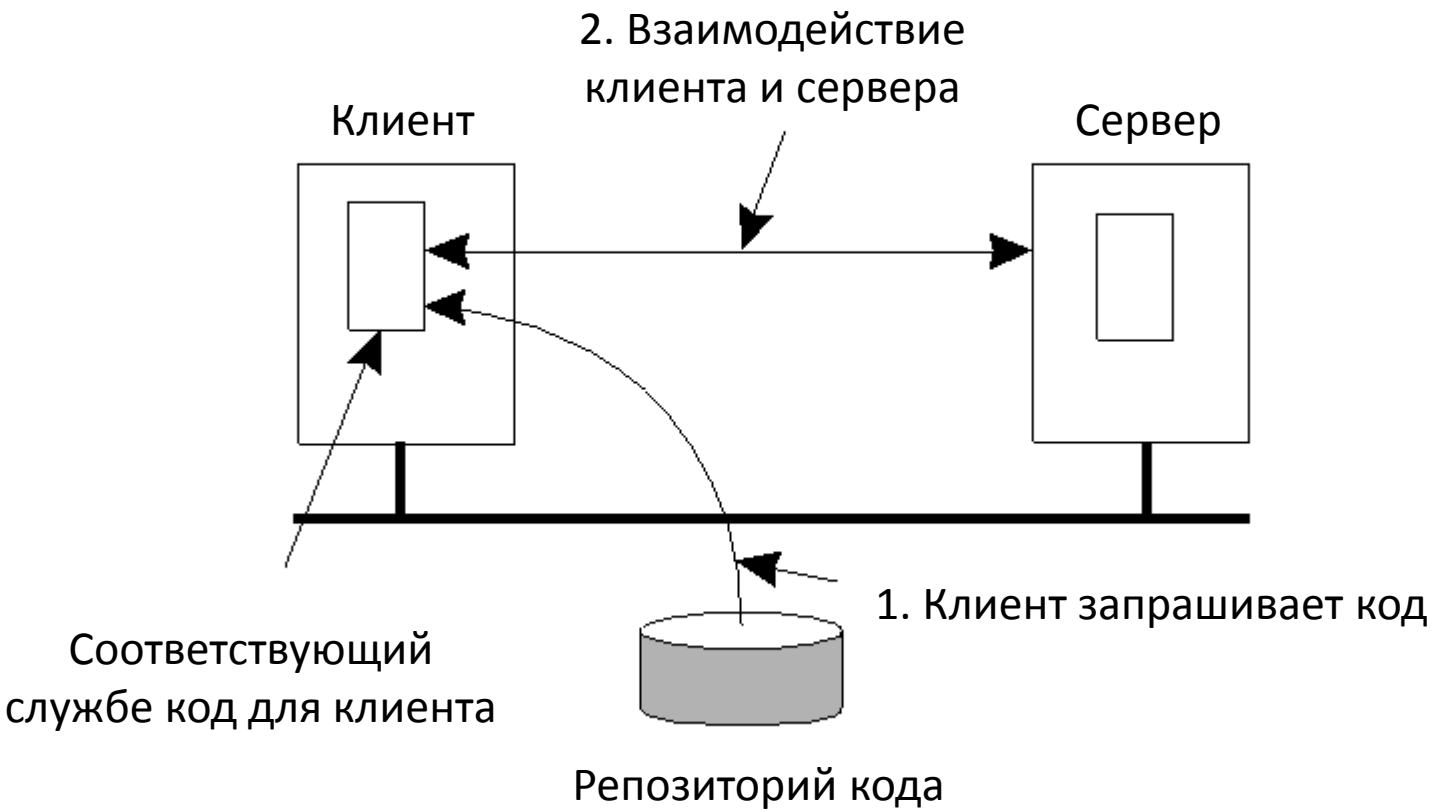
Принципы децентрализации

- Как известно, один из принципов компьютерной техники – единство представления данных и кода.
- В связи с этим возникает возможность переноса не только данных приложения, но и исполняемого кода на удаленную машину.

Основные преимущества переноса кода

- Балансировка нагрузки вычислительных модулей распределенной ВС и повышение производительности.
- Уменьшение трафика в коммуникационной подсистеме распределенной ВС.
- Обеспечение гибкости распределенных ВС и возможности динамического конфигурирования.
- Отсутствие необходимости иметь заранее сконфигурированное прикладное ПО в узлах системы.

Гибкость


2. Взаимодействие
клиента и сервера
Клиент
Сервер
1. Клиент запрашивает код
Соответствующий
службе код для клиента
Репозиторий кода

Процесс

Перенос кода в РВС происходит в форме миграции процессов.

В процессе можно выделить следующие сегменты:

1. **Сегмент кода:** исполняемый код программы
2. **Сегмент ресурсов:** содержит ссылки на внешние ресурсы, используемые процессом
3. **Сегмент исполнения:** хранение состояния процесса (стек, счетчик программы и т.п.)

Слабая мобильность

- **Слабая мобильность** (weak mobility) – перенос только *сегмента кода* (возможно с некоторыми данными для инициализации).
  - Перенесенная программа всегда запускается с исходного состояния.
- Пример – Java-апплеты.
- Достоинства: простота реализации

Сильная мобильность

- **Сильная мобильность** (strong mobility) – перенос сегмента кода и сегмента исполнения.
  - Исполнение процесса можно начать с того места, на котором он был остановлен на предыдущей машине.
- Возможен как перенос процесса (с уничтожением на источнике), так и его клонирование

Методы инициации переноса кода

Перенос кода может быть инициирован:

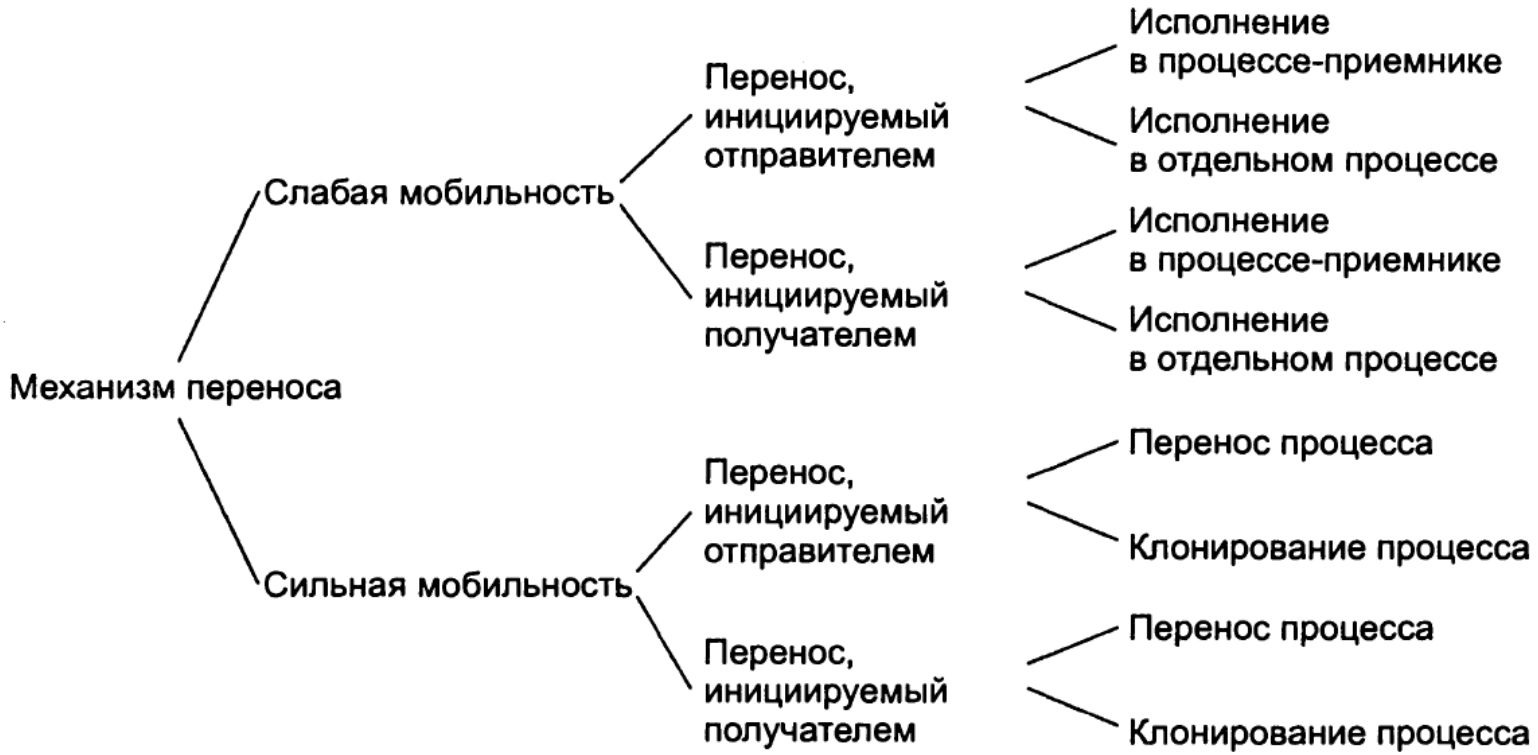
- Отправителем: передача поисковых программ на сервер баз данных, перенос и запуск троянских программ (серьезные проблемы в безопасности);
- Получателем: Java-апплеты (проблемы в безопасности тоже весомые, но не так очевидны как в первом случае).

Методы запуска перенесенного кода

Переносимый код может выполняться в процессе-приемнике или же в новом, отдельном процессе:

- Исполнение в процессе-приемнике (Java-апплет; требует отдельной защиты процесса-приемника от возможного вредоносного кода)
- Исполнение в отдельном процессе

Модели переноса кода



Перенос кода и локальные ресурсы

Связь процесса с локальными ресурсами

В отличие от сегмента кода и сегмента исполнения **сегмент ресурсов** невозможно также легко перенести на удаленную машину.

Выделяют 3 типа привязки ресурса к локальному процессу:

- **По идентификатору –** процесс требует в точности тот ресурс, на который ссылается (URL-адрес для ссылки на конкретный документ);
- **По значению –** процессу необходимо только значение ресурса (стандартные библиотеки Java)
- **По типу –** при обращении процесса к локальным устройствам (экран, принтер и т.п.)

Привязка ресурса к конкретному узлу

По способу привязки к конкретному узлу ВС ресурсы делятся на:

- **неприсоединенные,** т.е. легко переносимые с одного узла ВС на другой (например, файлы данных);
- **связанные**, перенос которых сопряжен с большими трудностями (например, базы данных или веб-сайты целиком);
- **фиксированные,** которые изначально привязаны к конкретной машине и не могут быть перенесены на другую (например, локальные устройства).

Возможные варианты переноса ресурсов

Выделяют 4 варианта решения задачи переноса локальных ресурсов:

1. **GR** (global reference)- организация ссылки;
2. **MV** (move) - перенос ресурса;
3. **CP** (copy) - копирование значения ресурса;
4. **RB** (rebind) - выполнение новой привязки к локальному ресурсу.

Решение задачи переноса ресурсов

	Неприсоединенный	Связанный	Фиксированный
По идентиф. По значению По типу	MV (или GR) CP (или MV, GR) RB (или GR, CP)	GR (или MV) GR (или CP) RB (или GR, CP)	GR GR RB (или GR)

- **По идентификатору:** лучше всего перенести его вместе с кодом. Однако, если он используется совместно с другими процессами – то лучше организовать глобальную ссылку.

Решение задачи переноса ресурсов

	Неприсоединенный	Связанный	Фиксированный
По идентиф. По значению По типу	MV (или GR) CP (или MV, GR) RB (или GR, CP)	GR (или MV) GR (или CP) RB (или GR, CP)	GR GR RB (или GR)

- **По значению:** обычно это библиотеки времени использования. Они допускают копирование. Если же копирование недопустимо (большой объем), возможно организовать глобальную ссылку.

Решение задачи переноса ресурсов

	Неприсоединенный	Связанный	Фиксированный
По идентиф. По значению По типу	MV (или GR) CP (или MV, GR) RB (или GR, CP)	GR (или MV) GR (или CP) RB (или GR, CP)	GR GR RB (или GR)

- **По типу:** независимо от способа привязки, лучший способ – это привязка процесса к аналогичным ресурсам на локальной машине. Если это невозможно – то желательно постараться сделать глобальную ссылку на них.

Перенос кода в гетерогенных средах

Проблема переноса кода в гетерогенных средах

- В гомогенных системах перенос кода очевиден
- В гетерогенных можно решить проблему переноса кода (слабая мобильность) создав по одному варианту исполняемого кода на каждый тип ВС.
  - Но как предугадать появление новой платформы?
  - Как организовать сильную мобильность?

Концепция виртуальных машин

Виртуальная машина –

- это программная или аппаратная система, эмулирующая аппаратное обеспечение некоторой платформы и исполняющая программы для этой платформы (целевая платформа) на другой платформе (хост-платформа)
- или виртуализирующая некоторую платформу и создающая на ней среды, изолирующие друг от друга программы и даже операционные системы (песочница);

Исполняемый код ВМ

В зависимости от типа, ВМ может исполнять:

- некоторый машинно-независимый код:
  - *байт-код Java* выполняется *виртуальной машиной Java;*
  - *ActionScript* выполняется *виртуальной машиной FlashPlayer;*
  - *IntermediateLanguage* выполняется *.NET Common Language Runtime;*
- или машинный код реального процессора.

Среда Java (JVM)

Идея Java зародилась в 1980 (язык Oak («дуб») для программирования бытовых электронных устройств).

Назван в честь марки кофе Java, любимого некоторыми программистами.

Основные компоненты среды Java:

- язык программирования Java;
- платформа Java.

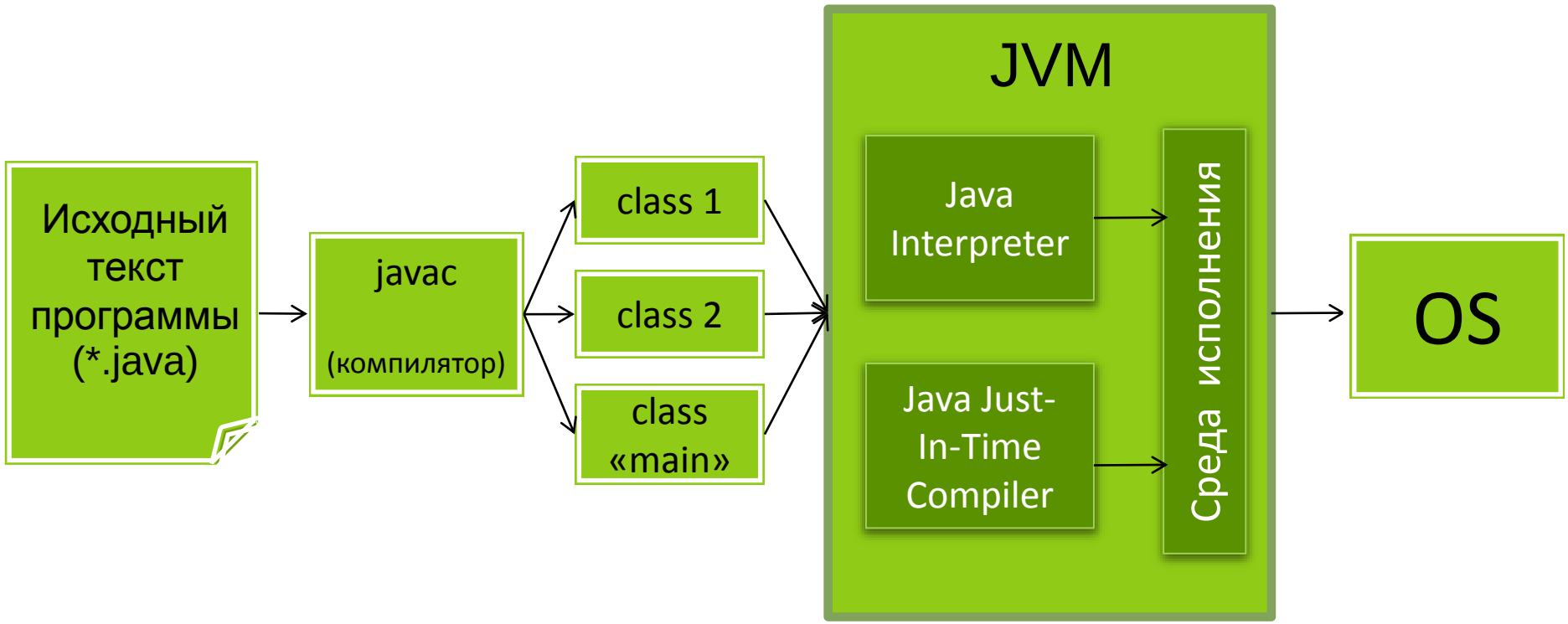
Типы приложений на платформе Java

- **приложения** – программы в обычном смысле, выполняемые, однако, в среде платформы Java;
- **апплеты** – программы, выполняемые в среде Web-броузера, поддерживающего платформу Java;
- **сервлеты** – Java-программы, серверные компоненты распределенных приложений;
- **программы, выполняющиеся в средах продуктов** промежуточного программного обеспечения (Oracle и т.п.)

Виртуальная машина Java

- **Переносимость в Java** достигается за счет того, что Java-программа компилируется в так называемый **байт-код.**
- **Байт-код –** Java - команды некоторой абстрактной машины, называемой **виртуальной машиной Java** (**JVM**)
- Байт-код не зависит от базовой операционной системы и оборудования ВС.

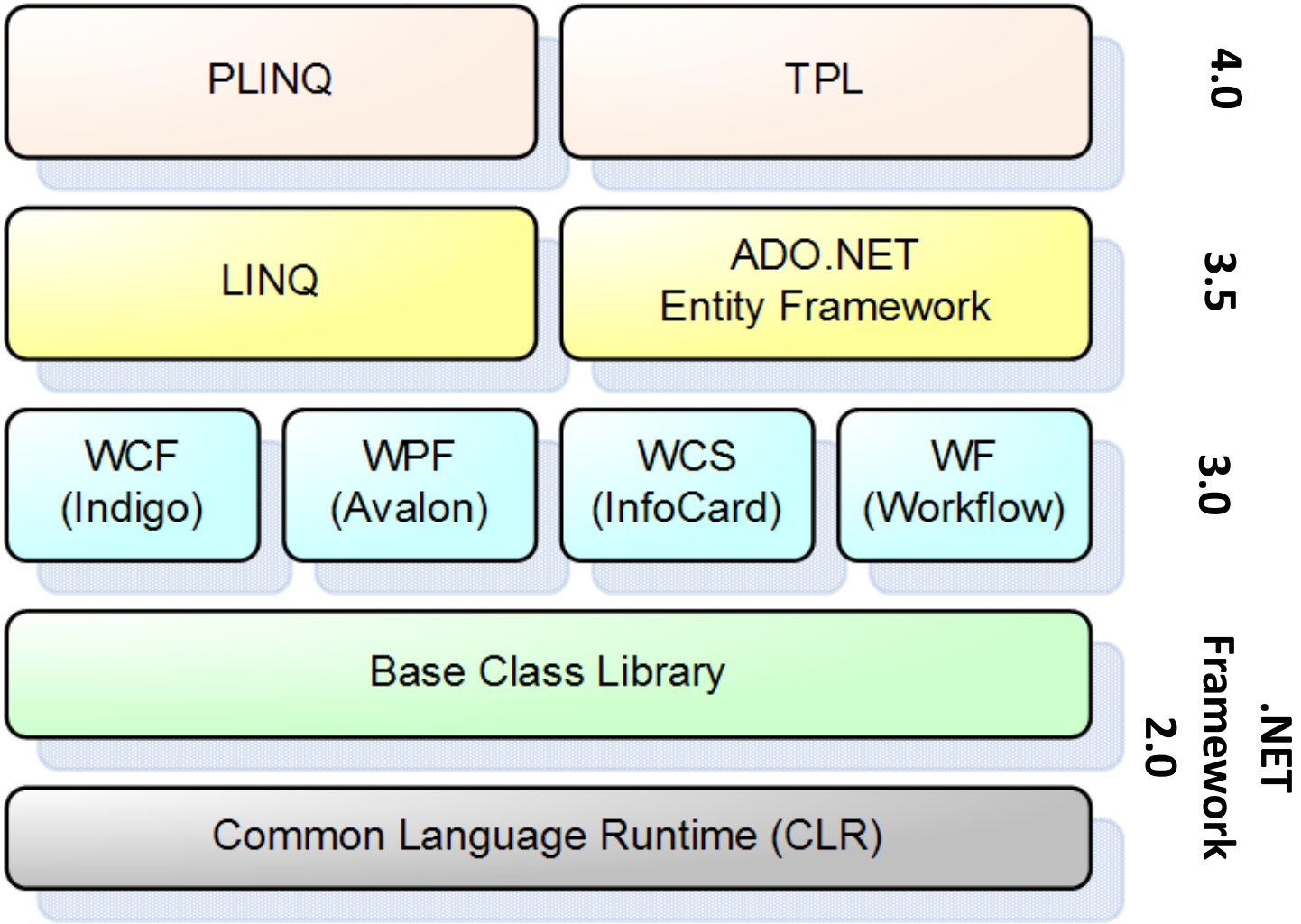
Исполнение приложений на платформе Java



Инфраструктура .NET Framework

- .NET Framework – это программная инфраструктура от компании Microsoft, обеспечивающая разработку как настольных, так и web-приложений.
- Основная идея .NET – совместимость программных систем, написанных на разных языках (на основе трансляции всех программ в байт-код **CIL - Common Intermediate Language**).

Стек технологий .NET Framework



Кросплатформенность .NET Framework

- .NET изначально не поддерживал кроссплатформенность.
- Mono — проект по созданию полноценного воплощения системы .NET на базе свободного программного обеспечения.
- Объем реализации:
  - компилятор C#;
  - среда исполнения .NET
  - отладчик;
  - блок библиотек (вплоть до .NET 3.5 кроме WPF и WF)

Итого

- Для переноса данных используются методы сериализации (протоколы XML, JSON, Protobuf)
- Мы сталкиваемся с переносом кода практически ежедневно.
- Перенос кода производится в виде миграции процессов.
- Выделяют 3 типа привязки ресурса к локальному процессу: по идентификатору, по значению, по типу.
- Выделяют 4 варианта решения задачи переноса локальных ресурсов: организация ссылки; перенос ресурса; копирование значения ресурса; выполнение новой привязки к локальному ресурсу.
- В современном мире мобильность кода чаще всего реализуется на базе виртуальных машин (Java, .NET)