# Платформа

1. Однако для полноценной разработки на стороне сервера или создания десктопных приложений скриптам необходим доступ к ресурсам операционной системы: файловой системе, сетям, процессам и многому другому.

Эти требования выходят далеко за рамки возможностей чистого JavaScript. Для их реализации нужна специальная программная среда — платформа. Именно платформа предоставляет языку программирования:

Доступ к API ОС: Файловая система, сетевые сокеты, переменные среды, аргументы командной строки.

Инструменты выполнения: Современные движки (V8, SpiderMonkey) с JIT-компиляцией для высокой производительности.

Систему управления пакетами: Например, npm (Node Package Manager) — огромная экосистема библиотек и инструментов.

Цикл событий (Event Loop): Модель асинхронного неблокирующего ввода-вывода, которая является ключевой для эффективных сетевых операций.

## Современные практики разработки платформ

### Кроссплатформенность (Слой абстракции над ОС)

1. Современные платформы используют дополнительный слой абстракции для API к функционалу ОС, чтобы код мог компилироваться для любых ОС

Единый API поверх разнородных системных вызовов (Абстракция ОС).

Суть практики: Платформа предоставляет разработчику единый, унифицированный интерфейс (API) для выполнения операций (работа с файлами, сетью, процессами), а сама заботится о его трансляции в конкретные системные вызовы целевой операционной системы (Windows, Linux, macOS).

Как это работает:

Разработчик пишет код на стандартном API платформы (например, File.read в Java или fs.readFile в Node.js).

Платформа содержит реализации этого API для каждой поддерживаемой ОС. Например, код для открытия файла на Windows использует WinAPI (CreateFile), а на Linux — системные вызовы POSIX (open).

Во время выполнения платформа (ее рантайм) определяет, на какой ОС она работает, и использует соответствующую реализацию.

Преимущества:

Кроссплатформенность: Код пишется один раз и запускается везде (Write Once, Run Anywhere — знаменитый слоган Java).

Упрощение разработки: Программисту не нужно изучать API всех операционных систем.

Безопасность: Платформа может "причесать" и обезопасить низкоуровневые вызовы, добавив проверки прав доступа, обработку ошибок и т.д.

Примеры:

JVM: Классы в пакетах java.io, java.nio абстрагируют файловый ввод-вывод.

.NET (CLR): Пространства имен System.IO, System.Net работают одинаково в Windows и на Unix-подобных системах (благодаря .NET Core/.NET 5+).

Node.js: Модули fs, os, path в своей реализации используют либо собственные биндинги на C++, либо библиотеку libuv, которая и является этим слоем абстракции.

### Intermidiate Language (IL)

1. Использование промежуточного представления (байт-кода) и JIT-компиляция

Суть практики: Исходный код компилируется не в машинный код конкретного процессора, а в универсальный промежуточный язык (байт-код). Этот байт-код затем выполняется виртуальной машиной (JVM, CLR), которая в реальном времени (Just-In-Time) компилирует его в оптимизированный машинный код.

Преимущества:

Переносимость: Байт-код одинаков для всех архитектур. Виртуальная машина — это единственный компонент, который нужно портировать на новую платформу.

Оптимизация во время выполнения (JIT): Виртуальная машина может анализировать поведение программы "на лету" и применять агрессивные оптимизации, которые недоступны для статического компилятора (например, инлайнинг виртуальных методов, удаление неиспользуемого кода на основе статистики).

Безопасность: Байт-код проходит верификацию перед выполнением, что предотвращает множество ошибок и уязвимостей (например, повреждение памяти).

Примеры:

JVM: Java-байт-код. JIT-компиляторы HotSpot (C1, C2).

CLR: Common Intermediate Language (CIL). JIT-компилятор (RyuJIT в современном .NET).

Node.js: Не использует байт-код в том же смысле. Код JavaScript компилируется движком V8 напрямую в машинный код, но сам V8 внутри использует собственное промежуточное представление и сложный многоуровневый JIT (Ignition, TurboFan).

### JIT компиляция

### Неблокирующие операции

1. Акцент на асинхронность и неблокирующие операции

Суть практики: Современные платформы оптимизированы для работы с большим количеством одновременных операций ввода-вывода (сеть, файлы), не блокируя при этом основные потоки выполнения. Это основа для высокопроизводительных и масштабируемых приложений.

Как это реализуется:

Event Loop (Цикл событий): Механизм, который непрерывно проверяет очередь событий и делегирует их обработку (Node.js, библиотека libuv).

Асинхронные API: Платформа предоставляет асинхронные версии всех долгих операций (например, fs.readFileSync vs fs.promises.readFile в Node.js, async/await в C#).

Модель акторов / Green threads / Виртуальные потоки: Чтобы избежать дорогостоящего создания нативных потоков ОС, платформы используют свои легковесные механизмы параллелизма (например, Project Loom в Java introduces virtual threads).

Примеры:

Node.js: Построена вокруг неблокирующего I/O и цикла событий как фундаментальная парадигма.

.NET: Модель async/await является первоклассной гражданкой языка и платформы.

Java: С появлением CompletableFuture и реактивных библиотек (Project Reactor), а теперь и виртуальных потоков (Loom).

### Менеджеры пакетов

1. Модульность и экосистема пакетов

Суть практики: Платформа поощряет разбиение приложения на небольшие, повторно используемые библиотеки (пакеты, модули), управляемые централизованно и с помощью инструментов зависимостей.

Преимущества:

Повторное использование кода: Не нужно изобретать велосипед.

Управление зависимостями: Четкий контроль версий и разрешение конфликтов.

Разделение ответственности: Каждый пакет решает одну задачу.

Примеры:

Node.js: npm (Node Package Manager) — крупнейшая экосистема библиотек в мире.

JVM: Maven и Gradle для Java, Kotlin, Scala. Пакеты хранятся в репозиториях (например, Maven Central).

.NET: NuGet — централизованная система управления пакетами.

# Различные платформы

1. В настоящее время реализовано множество платформ для языков программирования – каждая из которых и её язык нужны для своих целей.

Они отличаются по языковой экосистеме, архитектуре, сценариям использования и разными пододами для решения своих задач.

Ниже рассмотрены несколько основных платформ.

## Node.js

<https://nodejs.org/docs/latest/api/>

1. Node.js (среда выполнения JavaScript)
2. использует JavaScript для асинхронной обработки в одном потоке, что делает его быстрым для веб-приложений.

Что это:

Среда выполнения JavaScript, позволяющая запускать JavaScript вне браузера.

Язык:

JavaScript.

Цель:

Разработка высокопроизводительных, масштабируемых сетевых приложений (веб-серверы, API).

Ключевые особенности:

Асинхронная обработка событий, однопоточность с циклом событий, что делает его быстрым для операций ввода-вывода.

1. Подробнее Node.js рассмотрен в языки (база)/2 Front / 1 Node.js платформа для выполнения JS кода

## JVM

1. JVM (Java Virtual Machine) для Java и других JVM-языков
2. ориентированы на компилируемые, многопоточные и объектно-ориентированные языки
3. Что это:

Виртуальная машина, исполняющая байт-код, сгенерированный из Java-кода и других языков, таких как Kotlin.

Языки:

Java, Kotlin, Scala и другие языки, транслирующиеся в байт-код JVM.

Цель:

Обеспечить кроссплатформенность (один раз написано — работает везде), высокую производительность и масштабируемость.

Ключевые особенности:

Многопоточность, строгая типизация, ориентированность на большие, надежные корпоративные приложения.

## CLR

1. CLR (Common Language Runtime) для .NET-языков (вроде C#)
2. ориентированы на компилируемые, многопоточные и объектно-ориентированные языки
3. Что это:

Среда выполнения для платформы .NET, аналогичная JVM, но для .NET-языков.

Языки:

C#, VB.NET, F# и другие.

Цель:

Обеспечить выполнение управляемого кода, взаимодействие между разными языками .NET и доступ к системным ресурсам.

Ключевые особенности:

Многопоточность, сборка мусора, управление безопасностью.