Программа
«*Современное WEB-программирование*»

Входной ассемент

Лекция 4.

Модуль 4. «Основы технологий web-программирования»

(*завершение*)

Основы сетевых технологий и операционных систем

Цель:

сформировать базовое понимание принципов работы ОС, виртуализации, сетей и контейнеризации.

Ключевые темы лекции:

- как устроены операционные системы;
- основы сетевых взаимодействий;
- технологии виртуализации;
- контейнеризация и её преимущества.

Почему это важно:

- понимание ОС — фундамент для разработки и администрирования;
- сетевые знания нужны для построения распределённых систем;
- виртуализация и контейнеры — стандарт современной инфраструктуры.

Введение в операционные системы и виртуализацию

Операционная система - основа вычислений:

- Посредник между аппаратурой и приложениями
- Управление ресурсами: процессор, память, устройства ввода-вывода
- Обеспечение безопасности и многозадачности

Основные функции ОС:

- Управление процессами
- Управление памятью
- Файловая система
- Сетевые возможности
- Безопасность и доступ

Виртуализация - создание абстрактного слоя:

- Эмуляция аппаратных ресурсов
- Изоляция сред выполнения
- Эффективное использование ресурсов

Виртуализация - концепции и технологии

Уровни виртуализации:

- **Аппаратная виртуализация** - гипервизор (VMware, Hyper-V)
- **Паравиртуализация** - модифицированная ОС (Xen)
- **Виртуализация на уровне ОС** - контейнеры (Docker)

Типы гипервизоров:

Тип 1 (bare-metal):
Аппаратура → Hypervisor
↓
Гостевые ОС

Тип 2 (hosted):
Аппаратура → ОС → Hypervisor
↓
Гостевые ОС

Преимущества виртуализации:

- Консолидация серверов
- Изоляция приложений
- Быстрое развертывание
- Отказоустойчивость

Абстракции операционных систем

Ключевые абстракции:

- **Процесс** - выполняющаяся программа
- **Поток (thread)** - легковесный процесс
- **Файл** - абстракция для хранения данных
- **Сокет** - сетевая конечная точка
- **Память** - виртуальное адресное пространство

Управление процессами:

```
# Linux команды для управления процессами
ps aux                  # список процессов
top                     # мониторинг в реальном времени
kill -9 PID             # принудительное завершение
nice -n 10 command      # запуск с измененным приоритетом
```

Состояния процесса:

Создание → Готовность → Выполнение → Ожидание → Завершение

Управление памятью и файловые системы

Виртуальная память:

- Страничная организация памяти
- Кэширование и подкачка (swapping)
- Защита памяти между процессами

Файловые системы:

- **FAT/NTFS** - Windows
- **ext4/XFS** - Linux
- **APFS** - macOS
- **ZFS/Btrfs** - продвинутые с snapshot’ами

Иерархия файловой системы Linux:

/

├── bin (бинарные файлы)

├── etc (конфигурации)

├── home (пользователи)

├── var (изменяемые данные)

├── tmp (временные файлы)

└── proc (виртуальная файловая система процессов)

Основы сетевых технологий - модель OSI

Модель OSI (7 уровней):

7. Прикладной (HTTP, FTP, DNS)
6. Представления (шифрование, сжатие)
5. Сеансовый (управление сессиями)
4. Транспортный (TCP, UDP)
3. Сетевой (IP, маршрутизация)
2. Канальный (Ethernet, MAC)
1. Физический (кабели, сигналы)

Модель TCP/IP (4 уровня):

- Прикладной (Application)
- Транспортный (Transport)
- Сетевой (Internet)
- Канальный (Network Access)

Сетевые протоколы и адресация

Ключевые протоколы:

- **IP** - межсетевое взаимодействие
- **TCP** - надежная передача с установкой соединения
- **UDP** - быстрая передача без установки соединения
- **HTTP/HTTPS** - веб-коммуникация
- **DNS** - преобразование имен в IP-адреса

IP-адресация и подсети:

Пример IPv4 адресации

IP: 192.168.1.100

Маска: 255.255.255.0

CIDR: 192.168.1.0/24

Основные команды сети

ping 8.8.8.8 # проверка connectivity

traceroute google.com # трассировка маршрута

netstat -tulpn # открытые порты

Контейнеризация - революция в развертывании

Что такое контейнеры:

- Изолированные среды выполнения приложений
- Совместное использование ядра ОС
- Легковесная альтернатива виртуальным машинам

Архитектура контейнеров:

Виртуальная машина:

Аппаратура → Hypervisor

↓

Гостевая ОС

↓

Приложения

Контейнер:

Аппаратура → ОС → Docker Engine

↓

Контейнеры

Преимущества контейнеров:

- Быстрый запуск (секунды vs минуты)
- Меньший overhead
- Переносимость между средами
- Микросервисная архитектура

Docker и оркестрация контейнеров

Docker основы:

```
# Dockerfile пример
FROM ubuntu:20.04
RUN apt-get update && apt-get install -y python3
COPY . /app
WORKDIR /app
CMD ["python3", "app.py"]
```

Основные команды Docker:

```
docker build -t myapp .          # сборка образа
docker run -d -p 80:80 myapp     # запуск контейнера
docker ps                        # список контейнеров
docker logs conta
iner_id        # логи контейнера
```

Оркестрация:

- **Kubernetes** - промышленный стандарт
- **Docker Swarm** - встроенное решение Docker
- Управление масштабированием, обновлениями, отказоустойчивостью

Сетевые сервисы и безопасность

Критические сетевые сервисы:

- **DHCP** - автоматическая выдача IP-адресов
- **DNS** - разрешение доменных имен
- **NAT** - преобразование сетевых адресов
- **Firewall** - межсетевой экран

Безопасность ОС:

- Пользователи и группы
- Разрешения файловой системы
- SELinux/AppArmor
- Обновления и патчи

Мониторинг и диагностика:

```
# Мониторинг системы
htop                # монитор процессов
iotop               # монитор ввода-вывода
nethogs             # монитор сетевого трафика
ss -tulpn           # современная замена netstat
```

DevOps и современные практики

DevOps культура:

- Автоматизация процессов
- Непрерывная интеграция и доставка (CI/CD)
- Инфраструктура как код (IaC)

Инструментарий:

```
# Пример docker-compose.yml
version: '3'
services:
  web:
    build: .
    ports:
      - "80:80"
  db:
    image: postgres:13
    environment:
      POSTGRES_PASSWORD: example
```

Тренды:

- Serverless архитектура
- Service mesh (Istio, Linkerd)
- GitOps подход
- Облачные нативные технологии

Введение в технологии веб-программирования

Цель:

познакомить с основами веб-программирования на примере React,

научить настраивать окружение и строить простые приложения.

Что разберём:

- как подготовить среду для веб-разработки;
- основы работы с компонентами и данными в React;
- маршрутизацию (React Router): вложенные маршруты, перенаправления, страницу 404;
- создание простого интерактивного приложения.

Почему React?

- популярный фреймворк для интерфейсов;
- компонентный подход;
- большая экосистема и сообщество.

Введение в веб-программирование и установка окружения

Что такое веб-программирование:

- Frontend - клиентская часть (браузер)
- Backend - серверная часть
- Fullstack - полный цикл разработки

Современный стек технологий:

- **React** - библиотека для создания UI
- **Node.js** - серверная платформа
- **Базы данных** - PostgreSQL, MongoDB
- **Инструменты сборки** - Webpack, Vite

Установка окружения:

```
# Установка Node.js и npm
node --version
npm --version

# Создание React приложения
npx create-react-app my-app
cd my-app
npm start
```

Основы React - компоненты и JSX

Компонентный подход:

- Переиспользуемые блоки интерфейса
- Изоляция логики и стилей
- Иерархическая структура

Функциональные компоненты:

```
// Простой компонент
function Welcome(props) {
    return <h1>Привет, {props.name}!</h1>;
}

// Стрелочная функция
const Welcome = (props) => {
    return <h1>Привет, {props.name}!</h1>;
};

// Использование
<Welcome name="Анна" />
```

JSX синтаксис:

- HTML-подобный синтаксис в JavaScript
- Обязательный один корневой элемент
- className вместо class

Списки и рендеринг коллекций

Рендеринг массивов данных:

```
function UserList() {
    const users = [
        { id: 1, name: 'Анна', age: 25 },
        { id: 2, name: 'Иван', age: 30 },
        { id: 3, name: 'Мария', age: 28 }
    ];

    return (
        <ul>
            {users.map(user => (
                <li key={user.id}>
                    {user.name} - {user.age} лет
                </li>
            ))}
        </ul>
    );
}
```

Ключевые аспекты:

- **key** - уникальный идентификатор для оптимизации
- **map()** - преобразование данных в элементы
- Условный рендеринг с помощью && и тернарного оператора

State и обработка событий

Хук useState:

```
import { useState } from 'react';

function Counter() {
    const [count, setCount] = useState(0);

    return (
        <div>
            <p>Счетчик: {count}</p>
            <button onClick={() => setCount(count + 1)}>
                Увеличить
            </button>
        </div>
    );
}
```

Обработка событий:

```
function Form() {
    const [inputValue, setInputValue] = useState('');

    const handleSubmit = (event) => {
        event.preventDefault();
        console.log('Отправлено:', inputValue);
    };

    return (
        <form onSubmit={handleSubmit}>
            <input
                value={inputValue}
                onChange={(e) => setInputValue(e.target.value)}
            />
            <button type="submit">Отправить</button>
        </form>
    );
}
```

Создание простого приложения - Todo List

Полный пример приложения:

```
function TodoApp() {
    const [todos, setTodos] = useState([]);
    const [inputValue, setInputValue] = useState('');

    const addTodo = () => {
        if (inputValue.trim()) {
            setTodos([...todos, {
                id: Date.now(),
                text: inputValue,
                completed: false
            }]);
            setInputValue('');
        }
    };

    const toggleTodo = (id) => {
        setTodos(todos.map(todo =>
            todo.id === id
                ? { ...todo, completed: !todo.completed }
                : todo
        ));
    };

    return (
        <div>
            <input
                value={inputValue}
                onChange={(e) => setInputValue(e.target.value)}
                onKeyPress={(e) => e.key === 'Enter' && addTodo()}
            />
            <button onClick={addTodo}>Добавить</button>

            <ul>
                {todos.map(todo => (
                    <li
                        key={todo.id}
                        style={{
                            textDecoration: todo.completed ? 'line-through' : 'none'
                        }}
                        onClick={() => toggleTodo(todo.id)}
                    >
                        {todo.text}
                    </li>
                ))}
            </ul>
        </div>
    );
}
```

Введение в React Router

Установка и настройка:

```
npm install react-router-dom
```

Базовая маршрутизация:

```
import { BrowserRouter, Routes, Route, Link } from 'react-router-dom';


function App() {
  return (
    <BrowserRouter>
      <nav>
        <Link to="/">Главная</Link>
        <Link to="/about">О нас</Link>
        <Link to="/contact">Контакты</Link>
      </nav>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
        <Route path="/contact" element={<Contact />} />
      </Routes>
    </BrowserRouter>
  );
}

function Home() { return <h1>Главная страница</h1>; }
function About() { return <h1>О нас</h1>; }
function Contact() { return <h1>Контакты</h1>; }
```

Вложенные маршруты и динамические параметры

Вложенная маршрутизация:

```
function App() {
    return (
        <Routes>
            <Route path="/users" element={<Users />}>
                <Route path=":userId" element={<UserProfile />} />
                <Route path=":userId/posts" element={<UserPosts />} />
            </Route>
        </Routes>
    );
}

function Users() {
    return (
        <div>
            <h1>Пользователи</h1>
            <Outlet /> {/* Место для вложенных маршрутов */}
        </div>
    );
}
```

Динамические параметры:

```
import { useParams } from 'react-router-dom';


function UserProfile() {
    const { userId } = useParams();


    return <h1>Профиль пользователя {userId}</h1>;
}
function UserPosts() {
    const { userId } = useParams();
    const [posts, setPosts] = useState([]);
    // Загрузка постов пользователя
    useEffect(() => {
        fetchPosts(userId).then(setPosts);
    }, [userId]);
    return (
        <div>
            <h2>Посты пользователя {userId}</h2>
            {posts.map(post => (
                <div key={post.id}>{post.title}</div>
            ))}
        </div>
    );
}
```

Перенаправления и навигация

Программная навигация:

```
import { useNavigate } from 'react-router-dom';

function LoginForm() {
    const navigate = useNavigate();
    const [credentials, setCredentials] = useState({});

    const handleLogin = async () => {
        try {
            await login(credentials);
            navigate('/dashboard'); // Перенаправление после успеха
        } catch (error) {
            console.error('Ошибка входа');
        }
    };

    return (
        <form onSubmit={handleLogin}>
            {/* поля формы */}
        </form>
    );
}
```

Компонент Navigate для перенаправлений:

```
import { Navigate } from 'react-router-dom';

function ProtectedRoute({ user, children }) {
    if (!user) {
        return <Navigate to="/login" replace />;
    }

    return children;
}

// Использование
<Route
    path="/dashboard"
    element={
        <ProtectedRoute user={currentUser}>
            <Dashboard />
        </ProtectedRoute>
    }
/>
```

Обработка 404 и ошибок

Страница 404:

```
function NotFound() {
    return (
        <div style={{ textAlign: 'center', padding: '50px' }}>
            <h1>404 - Страница не найдена</h1>
            <p>Извините, запрашиваемая страница не существует.</p>
            <Link to="/">Вернуться на главную</Link>
        </div>
    );
}

// В маршрутизации
<Routes>
    <Route path="/" element={<Home />} />
    <Route path="/about" element={<About />} />
    <Route path="*" element={<NotFound />} />
</Routes>
```

Обработка ошибок в компонентах:

```
class ErrorBoundary extends Component {
    constructor(props) {
        super(props);
        this.state = { hasError: false };
    }
    static getDerivedStateFromError(error) {
        return { hasError: true };
    }
    componentDidCatch(error, errorInfo) {
        console.error('Ошибка:', error, errorInfo);
    }
    render() {
        if (this.state.hasError) {
            return <h1>Что-то пошло не так.</h1>;
        }
        return this.props.children;
    }
}

// Использование
<ErrorBoundary>
    <MyComponent />
</ErrorBoundary>
```

Итоги и лучшие практики

Ключевые концепции курса:

- Компонентный подход в React
- Управление состоянием с useState
- Рендеринг списков и коллекций
- Маршрутизация с React Router
- Обработка ошибок и 404

- **Дальнейшее развитие:**
- State management (Redux, Zustand)
- Серверные компоненты
- Оптимизация производительности
- Тестирование (Jest, React Testing Library)

Лучшие практики:

```
// Правильное использование ключей
{todos.map(todo => (
  <TodoItem key={todo.id} todo={todo} />
))}
// Разделение ответственности
function UserContainer() {
  const [users, setUsers] = useState([]);
    useEffect(() => {
    fetchUsers().then(setUsers);
  }, []);
  return <UserList users={users} />;
}
function UserList({ users }) {
  return (
    <ul>
      {users.map(user => (
        <UserItem key={user.id} user={user} />
      ))}
    </ul>
  );
}
```