Введение в сетевые протоколы

ОСНОВЫ WEB ТЕХНОЛОГИЙ

Get Started

Введение в сетевые протоколы

Сетевые протоколы — это наборы правил, которые управляют обменом данными между компьютерами в сети. Они определяют формат, последовательность и иногда ошибки при передаче данных.

Основные сетевые протоколы:

- **IP (Internet Protocol)**: Определяет адресацию и маршрутизацию пакетов данных, чтобы они достигли нужного места.
- **TCP (Transmission Control Protocol)**: Обеспечивает надежную, упорядоченную и безошибочную доставку данных от отправителя к получателю.
- **UDP (User Datagram Protocol)**: Предоставляет простые и быстрые, но без гарантий доставки, передачи данных.
- **HTTP/HTTPS**: Протоколы для передачи веб-документов (мы поговорим о них подробнее далее).
- **FTP (File Transfer Protocol)**: Протокол для передачи файлов.
- **SMTP (Simple Mail Transfer Protocol)**: Протокол для отправки электронной почты.

Как работает HTTP/HTTPS

HTTP (HyperText Transfer Protocol) — это протокол прикладного уровня, предназначенный для передачи гипертекстовых документов, таких как HTML.

Основные характеристики:

1. **Безсостояние:** Каждый запрос считается отдельной транзакцией.
2. **Соединение может быть не постоянным:** По умолчанию соединение закрывается после каждого запроса.
3. **Методы:** HTTP включает в себя ряд методов, таких как **GET** (запрос данных), **POST** (отправка данных), **PUT** (обновление данных) и **DELETE** (удаление данных).

HTTPS (HTTP Secure) — это расширение HTTP, предоставляющее безопасное соединение с помощью шифрования. Это достигается с использованием SSL (Secure Socket Layer) или TLS (Transport Layer Security).

Основное отличие между HTTP и HTTPS — это безопасность. HTTPS шифрует информацию, которую пользователь отправляет и получает, предотвращая перехват данных третьими лицами.

HTTP

- **Заголовки:** Заголовки HTTP передают информацию о браузере пользователя, странице, на которой находится пользователь, и самом сервере.

Например:

  - User-Agent: информация о браузере пользователя.
  - Accept: типы данных, которые браузер может принимать.

- **Куки:** Куки — это данные, которые сервер может отправить браузеру, и которые будут храниться и возвращаться с каждым последующим запросом к этому серверу.
- **Сессии:** Сессии позволяют хранить информацию о пользователе между запросами.

HTTPS

- **Цифровые сертификаты:** Сервер использует цифровой сертификат для подтверждения своей личности. Этот сертификат должен быть подписан доверенным центром сертификации (ЦС).
- **Шифрование:** С помощью процесса "рукопожатия" клиент (браузер) и сервер согласовывают, какой алгоритм шифрования они будут использовать, и обмениваются ключами шифрования.

Пример запроса HTTP

```vbnet
GET /index.html HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0
Accept: text/html
```

Пример ответа HTTP

```php
HTTP/1.1 200 OK
Date: Mon, 23 May 2022 22:38:34 GMT
Server: Apache/2.4.1 (Unix)
Last-Modified: Sun, 22 May 2022 19:15:56 GMT
Content-Length: 4321
Content-Type: text/html

<!DOCTYPE html>
<html>
...
</html>
```

3. Основы работы с RESTful API

REST (Representational State Transfer) — это архитектурный стиль проектирования сетевых приложений. RESTful API — это API, который следует принципам REST.

Основные характеристики REST:

1. **Безсостояние:** Каждый запрос с сервера выполняется независимо, без знания предыдущих запросов.
2. **Клиент-сервер:** Существует разделение между клиентом и сервером.
3. **Кэширование:** Ответы от сервера могут кэшироваться на стороне клиента.
4. **Единообразие интерфейса:** Все запросы выполняются единообразно.
5. **Слой системы:** Клиент не может прямо взаимодействовать с сервером, а только через интерфейс.

Работа с RESTful API обычно включает в себя отправку HTTP-запросов к серверу и получение ответов. Например, вы можете отправить GET-запрос к API, чтобы получить информацию о пользователе, или POST-запрос, чтобы создать нового пользователя.

Концепции RESTful API:

- **Ресурсы**: В сердце любого RESTful API лежат ресурсы. Это объекты или данные, которыми вы хотите поделиться. Например, это могут быть пользователи, посты, комментарии.
- **Методы**: Основные методы HTTP, используемые в REST, это GET (получение данных), POST (создание новых данных), PUT (обновление данных) и DELETE (удаление данных).
- **Статусы ответов**: RESTful API использует стандартные HTTP-статусы для указания результата запроса. Например:
  - **200 OK**: успешный запрос.
  - **404 Not Found**: ресурс не найден.
  - **500 Internal Server Error**: ошибка на сервере.
- **JSON и XML**: RESTful API часто используют JSON или XML для отправки и получения данных.

Пример запроса RESTful API для получения информации о пользователе с ID 123:

vbnet

```
GET /api/v1/users/123 HTTP/1.1
Host: www.example.com
Authorization: Bearer YOUR_API_TOKEN
Accept: application/json
```

Ответ

makefile

```
HTTP/1.1 200 OK
Date: Mon, 23 May 2022 22:38:34 GMT
Content-Length: 150
Content-Type: application/json

{
    "id": 123,
    "name": "John Doe",
    "email": "john@example.com",
    "joined_at": "2022-01-15T12:34:56Z"
}
```

Пример запроса RESTful API для создания нового пользователя:

bash

```
POST /api/v1/users HTTP/1.1
Host: www.example.com
Authorization: Bearer YOUR_API_TOKEN
Content-Type: application/json

{
    "name": "Jane Smith",
    "email": "jane@example.com"
}
```

Ответ

makefile

```
HTTP/1.1 201 Created
Date: Mon, 23 May 2022 22:40:00 GMT
Content-Length: 160
Content-Type: application/json

{
    "id": 124,
    "name": "Jane Smith",
    "email": "jane@example.com",
    "joined_at": "2022-05-23T22:40:00Z"
}
```

Лучшие практики

Версионирование: Чтобы обеспечивать обратную совместимость, рекомендуется версионировать ваш API. Например: /v1/users или /v2/posts.
Пагинация: Если у вас много данных, вы можете использовать пагинацию для их разбивки на меньшие порции.
Аутентификация и авторизация: Многие API требуют, чтобы пользователи аутентифицировались, перед тем как получить доступ к ресурсам. Это часто делается с использованием токенов.

Практическое задание: Работа с RESTful API

Цель:
Получить практический опыт взаимодействия с **RESTful API**, используя **HTTP** запросы для создания, чтения, обновления и удаления ресурсов.

Задача:
Разработать простой клиентский интерфейс (например, на **HTML** и **JavaScript**), который будет взаимодействовать с открытым **RESTful API**.

Шаги:

1. Выберите API: Вы можете выбрать любой публичный API для этого задания. Например, JSONPlaceholder предоставляет простой API для тестирования и прототипирования.
2. **Создание интерфейса:**
   - На вашей HTML-странице создайте простую форму, чтобы пользователи могли добавлять новые ресурсы (например, посты).
   - Отобразите список существующих ресурсов (например, всех текущих постов).
   - Рядом с каждым ресурсом добавьте кнопки для обновления и удаления.
3. **Взаимодействие с API:**
   - Когда пользователь отправляет форму для создания нового ресурса, используйте метод POST для отправки данных на сервер.
   - Чтобы отобразить существующие ресурсы, используйте метод GET.
   - Для обновления ресурса используйте метод PUT или PATCH.
   - Для удаления ресурса используйте метод DELETE.
4. **Обработка ответов:**
   - После каждого запроса к серверу отобразите соответствующее уведомление или ошибку на вашей странице.
   - Убедитесь, что ваш интерфейс корректно обновляет данные на странице после каждого успешного запроса.

Дополнительно:

- Добавьте возможность фильтрации или сортировки ресурсов.
- Реализуйте пагинацию, если API поддерживает такую возможность.

После завершения задания у вас будет простое веб-приложение, которое может взаимодействовать с внешним API для управления ресурсами. Это отличная практика для понимания, как работают RESTful API и как их можно использовать в реальных проектах.

Подготовьте пошаговый отчет скриншотами для защиты!