



Стандартная библиотека. Сеть



Игорь Лизунов, Android developer@T-bank

22 октября 2025

План семинара

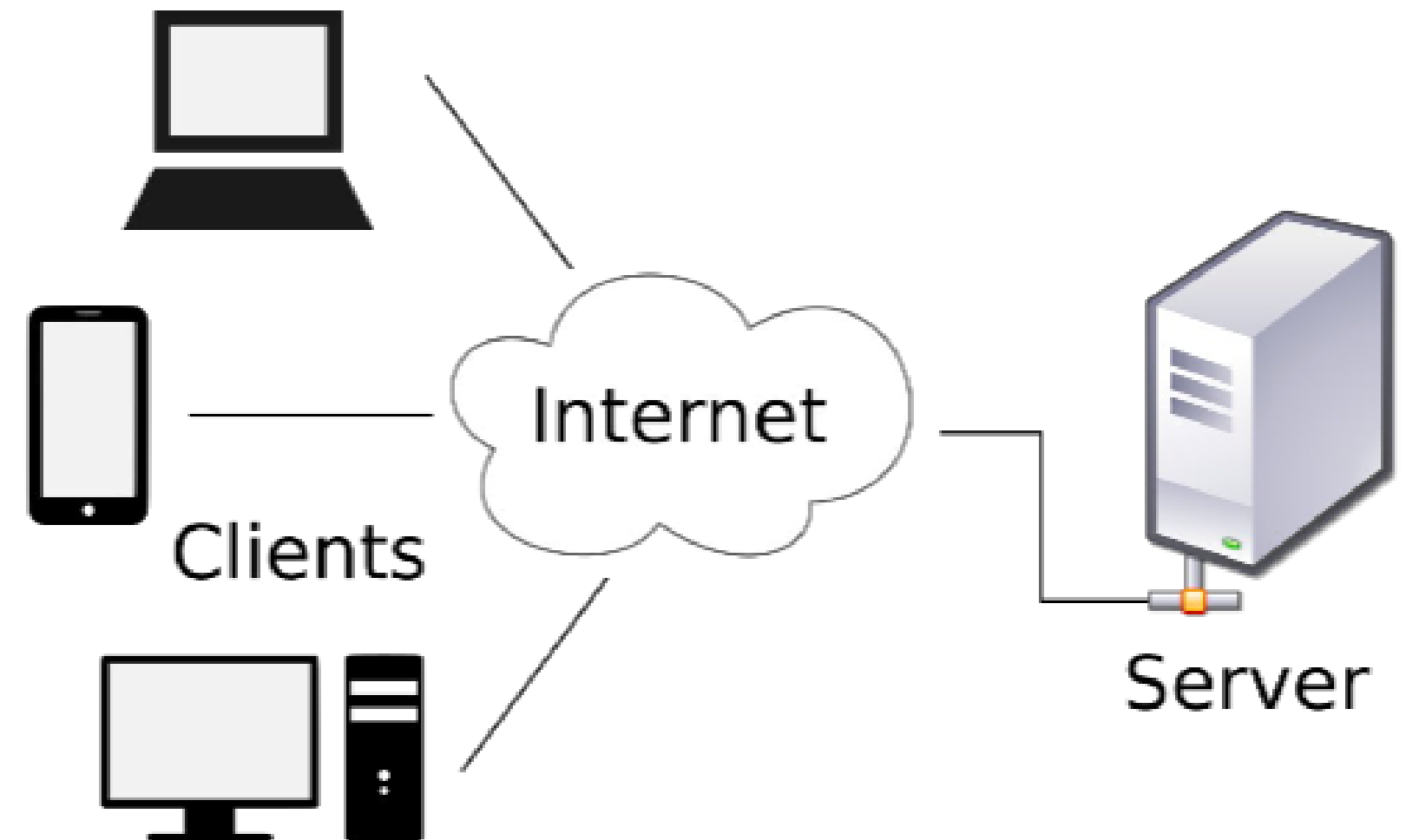
- ➔ Основы сетевого взаимодействия
- ➔ HTTP и REST API
- ➔ Библиотеки для сети
- ➔ JSON сериализация



Основы сетевого взаимодействия

Сетевое взаимодействие

Сетевое взаимодействие - обмен данными между удалёнными участниками по согласованным правилам.

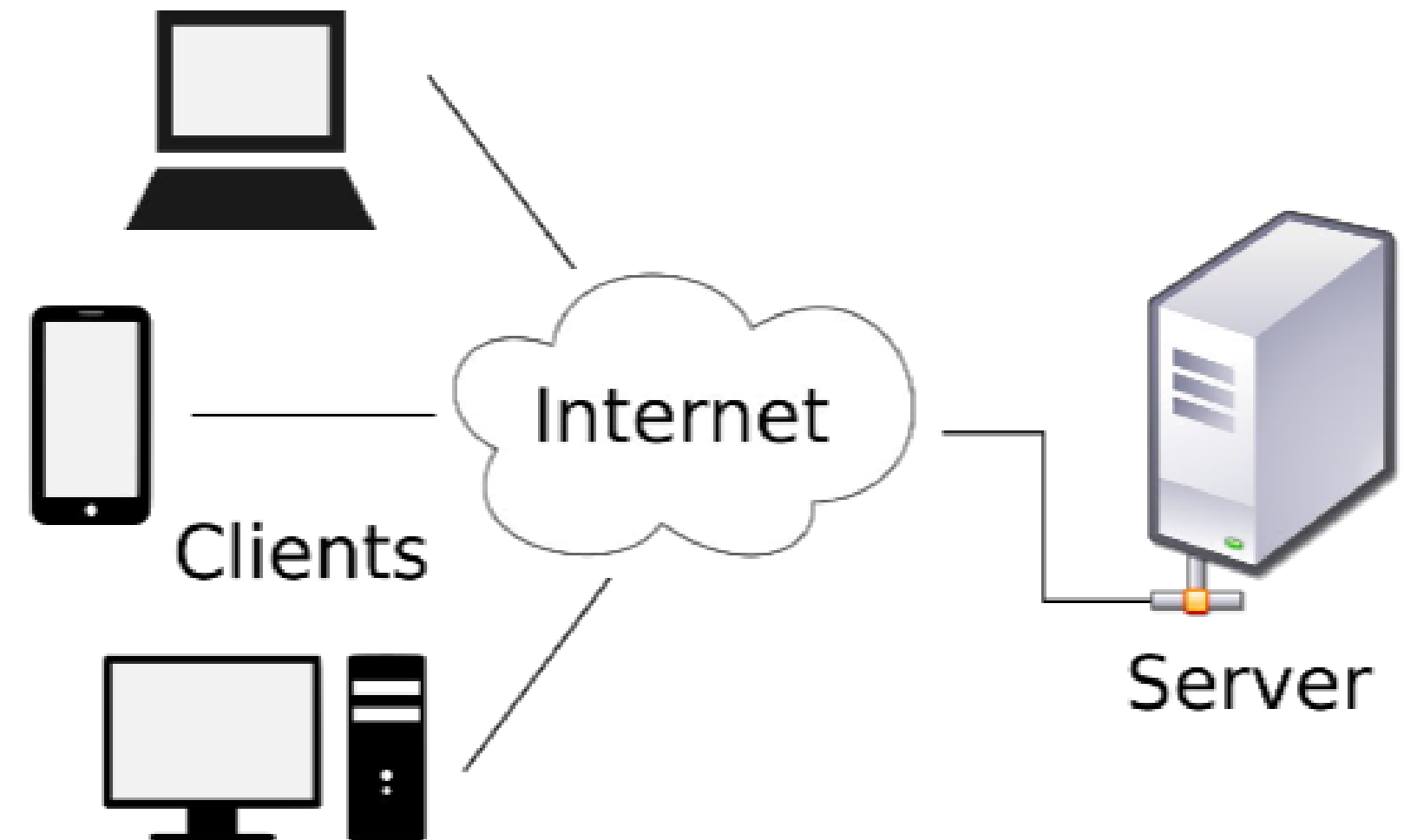


Сетевое взаимодействие

Сетевое взаимодействие - обмен данными между удалёнными участниками по согласованным правилам.

Модели обмена:

1. Запрос-ответ
2. Двухнаправленный канал



Сетевое взаимодействие

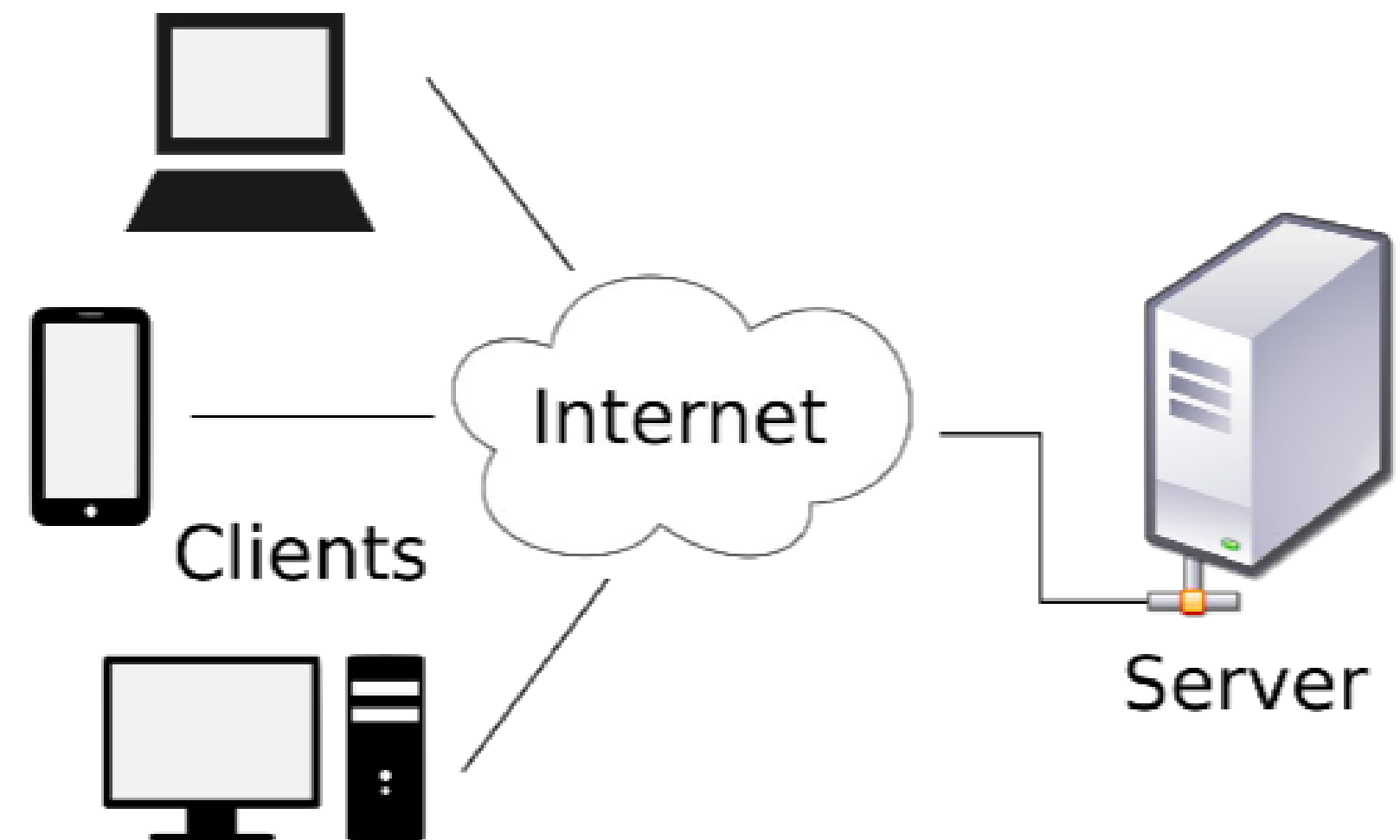
Сетевое взаимодействие - обмен данными между удалёнными участниками по согласованным правилам.

Модели обмена:

1. Запрос-ответ
2. Двухнаправленный канал

Роли:

1. Инициатор (клиент)
2. Принимающая сторона (сервер)
3. Равноправные узлы (peer-to-peer)



Сетевое взаимодействие

Сетевое взаимодействие - обмен данными между удалёнными участниками по согласованным правилам.

Модели обмена:

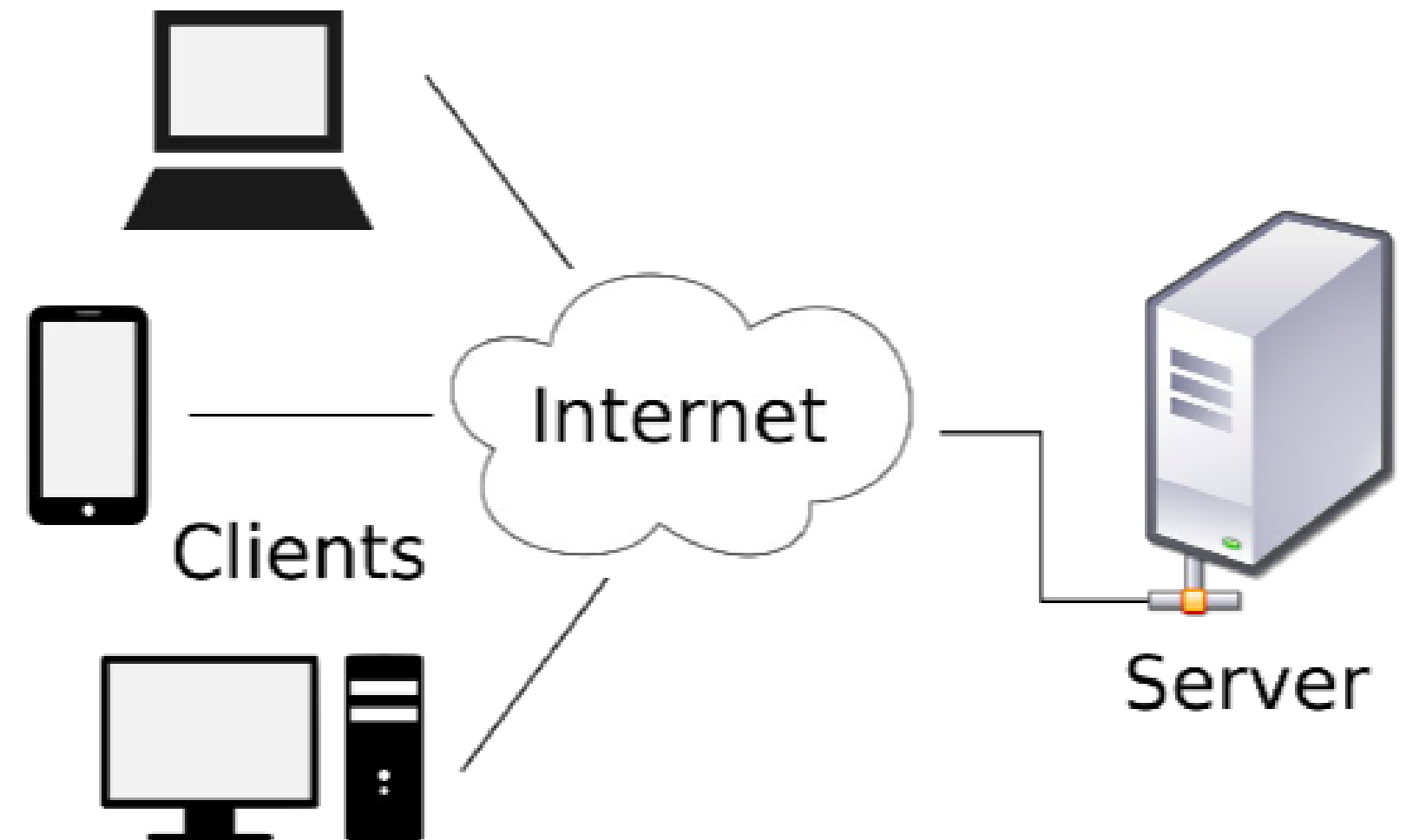
1. Запрос-ответ
2. Двухнаправленный канал

Роли:

1. Инициатор (клиент)
2. Принимающая сторона (сервер)
3. Равноправные узлы (peer-to-peer)

Типы клиентов:

1. Веб приложение
2. Мобильной приложение
3. Бекенд приложение
4. Любое устройство, подключенное к сети



Протоколы

Протокол - это набор правил и соглашений для передачи данных между устройствами в сети: как установить связь, как адресовать и форматировать сообщения, как обрабатывать порядок и ошибки.

Протоколы

Протокол - это набор правил и соглашений для передачи данных между устройствами в сети: как установить связь, как адресовать и форматировать сообщения, как обрабатывать порядок и ошибки.

Проблемы без протоколов:

1. Несовместимость - разные устройства могут использовать разные форматы данных

Протоколы

Протокол - это набор правил и соглашений для передачи данных между устройствами в сети: как установить связь, как адресовать и форматировать сообщения, как обрабатывать порядок и ошибки.

Проблемы без протоколов:

1. Несовместимость - разные устройства могут использовать разные форматы данных
2. Сложность интеграции - невозможность взаимодействия между разными системами

Протоколы

Протокол - это набор правил и соглашений для передачи данных между устройствами в сети: как установить связь, как адресовать и форматировать сообщения, как обрабатывать порядок и ошибки.

Проблемы без протоколов:

1. Несовместимость - разные устройства могут использовать разные форматы данных
2. Сложность интеграции - невозможность взаимодействия между разными системами
3. Отсутствие контроля ошибок - нет единого механизма проверки целостности данных

OSI

Модель OSI – это эталонная модель сетевого взаимодействия, разделяющая процесс передачи данных на 7 уровней.

Модель OSI				
Уровень	Название	Тип данных	Функции	Примеры
7	Прикладной		Доступ к сетевым службам	HTTP, FTP, POP3, SMTP, WebSocket
6	Представительский	Данные	Представление и шифрование данных	ASII, EBCDIC, JPEG, MIDI
5	Сеансовый		Управление сеансом связи	RPC, PAP, L2TP, gRPC
4	Транспортный	Сегменты Датаграммы	Прямая связь между конечными файлами и надежность	TCP, UDP, SCTP, Порты
3	Сетевой	Пакеты	Определение маршрута и логическая адресация	IPv4, IPv6, IPsec, AppleTalk, ICMP
2	Канальный	Биты/кадры	Физическая адресация	PPP, IEEE, 802.22, Ethernet, DSL, ARP, сетевая карта
1	Физический	Биты	Работа со средой передачи, сигналами и двоичными данными	USB, RJ (витая пара), коаксиальный, оптоволоконный), радиоканал

ТСР/IP

ТСР/IP — это практическая модель и набор протоколов, на которых работает Интернет.

Модель ТСР/IP				
Уровень	Название	Тип данных	Функции	Примеры
4	Прикладной	Данные	Доступ к сетевым службам, представление данных	HTTP, FTP, SMTP, DNS, Telnet, SSH
3	Транспортный	Сегменты Датаграммы	Надежная передача данных между приложениями	TCP, UDP, SCTP
2	Межсетевой	Пакеты	IP-адресация и маршрутизация пакетов	IPv4, IPv6, ICMP, ARP, IGMP
1	Сетевой интерфейс	Биты/кадры	Физическая передача данных по сети	Ethernet, Wi-Fi, PPP, DSL

OSI vs TCP/IP

Сравнение моделей OSI и TCP/IP		
Модель OSI	Модель TCP/IP	Соответствие и примечания
7. Прикладной 6. Представительский 5. Сеансовый	4. Прикладной	Три верхних уровня OSI (7, 6, 5) объединены в один прикладной уровень TCP/IP. Включает работу с приложениями, представление данных и управление сеансами.
4. Транспортный	3. Транспортный	Прямое соответствие 1:1. Оба уровня отвечают за надежную доставку данных между процессами (TCP, UDP).
3. Сетевой	2. Межсетевой	Прямое соответствие 1:1. Оба уровня отвечают за IP-адресацию, маршрутизацию и доставку пакетов между сетями.
2. Канальный 1. Физический	1. Сетевой интерфейс	Два нижних уровня OSI (2, 1) объединены в один уровень сетевого интерфейса. Включает физическую передачу и работу с сетевым оборудованием.

ТСР-соединение

ТСР-соединение - это сеанс транспортного протокола между двумя IP-адресами и портами, обеспечивающий надежную и упорядоченную доставку байтового потока.



UDP-соединение

UDP - это транспортный протокол без установления соединения, передающий независимые датаграммы с минимальными накладными расходами, без гарантий доставки, порядка и защиты от дубликатов.



TCP



UDP



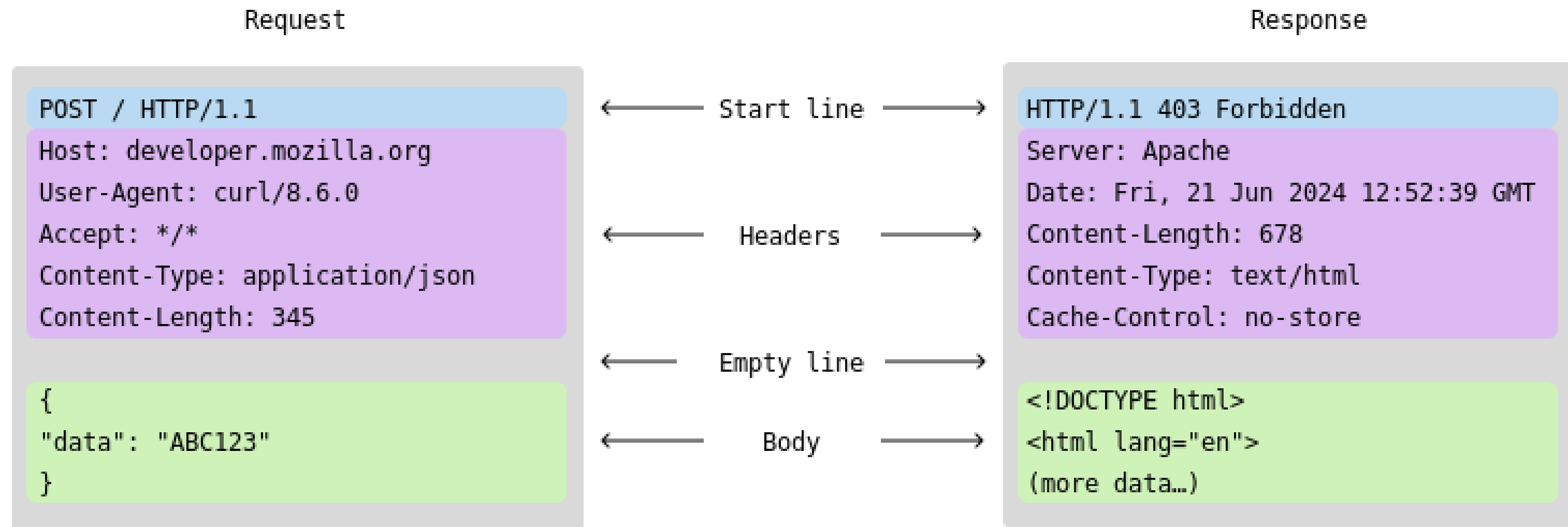
**Ваши
вопросы**



HTTP и REST API

HTTP

HTTP - это прикладной протокол: набор правил и соглашений, как клиент и сервер идентифицируют ресурсы (URI/URL), формируют и обрабатывают запросы/ответы (методы, заголовки, тело), интерпретируют коды состояния, согласуют формат и кэширование.



HTTP Status codes

HTTP status codes - это стандартизированные числовые коды в ответе сервера, описывающие результат обработки запроса, они управляют поведением клиентов.

Категории кодов состояния		
Категория	Диапазон	Назначение
1xx	Информационные	Запрос получен, процесс продолжается. Промежуточные ответы для информирования клиента о ходе обработки.
2xx	Успешные	Запрос успешно получен, понят и принят. Действие было получено, понято и принято сервером.
3xx	Перенаправления	Требуется дополнительное действие для завершения запроса. Клиент должен предпринять дополнительные действия.
4xx	Ошибки клиента	Запрос содержит ошибку или не может быть выполнен. Проблема на стороне клиента.
5xx	Ошибки сервера	Сервер не смог выполнить корректный запрос. Проблема на стороне сервера.

REST API

REST API - это стиль и набор ограничений для проектирования сервисов поверх HTTP.

Ключевые принципы REST constraints:

1. Клиент–сервер: разделение интерфейса и данных/логики.
2. Без состояния: каждый запрос самодостаточен.
3. Единообразный интерфейс: стандартные методы (GET/POST/PUT/PATCH/DELETE), согласование форматов (Accept/Content-Type).

Users

POST

/api/v1/users

Создать пользователя

▼

GET

/api/v1/users/{userId}

Получить пользователя по ID

▼

PUT

/api/v1/users/{userId}

Обновить информацию о пользователе

▼

DELETE

/api/v1/users/{userId}

Удалить пользователя по ID

▼

Schemas

User ▼ {

id

string
example: 123e4567-e89b-12d3-a456-426614174000
Уникальный идентификатор пользователя

name*

string
example: Иван Иванов
Имя пользователя

email*

string
example: ivan.ivanov@example.com
Email пользователя

blocked

boolean
example: false
Статус блокировки пользователя

}

JSON

JSON (JavaScript Object Notation) - формальный, текстовый, универсальный формат представления и обмена данными, описывающий деревообразные структуры из объектов и массивов. Формат независим от языков программирования и предназначен для межсистемной сериализации данных.

```
{  
  "string": "Текстовое значение",  
  "number": 42,  
  "decimal": 3.14,  
  "boolean": true,  
  "null": null,  
  "array": [1, 2, 3],  
  "object": {  
    "nested": "value"  
  }  
}
```

Summary

- Сетевое взаимодействие - это обмен данными по согласованным правилам.
- Модели: запрос-ответ и двусторонний канал.
- Роли: клиент, сервер, peer-to-peer.
- Протоколы обеспечивают совместимость и управление передачей данных.
- Модели OSI и TCP/IP определяют уровни сетевого взаимодействия.
- TCP - надёжное соединение с контролем доставки.
- UDP - простое соединение без гарантий.
- HTTP - прикладной протокол для обмена данными по сети.
- Методы, заголовки, тело и коды состояния определяют формат взаимодействия.
- REST - архитектурный стиль поверх HTTP.
- JSON - текстовый универсальный формат сериализации данных.

**Ваши
вопросы**



Библиотеки для сети

HttpURLConnection

HttpURLConnection - низкоуровневый, блокирующий HTTP-клиент из пакета `java.net`. Он удобен для базовых задач и ситуаций, когда важно обойтись без внешних зависимостей, при этом API предоставляет минимум удобств.

HttpURLConnection

HttpURLConnection - низкоуровневый, блокирующий HTTP-клиент из пакета java.net. Он удобен для базовых задач и ситуаций, когда важно обойтись без внешних зависимостей, при этом API предоставляет минимум удобств.

```
fun main() {  
    val conn = URL("https://httpbin.org/get").openConnection() as HttpURLConnection  
    return try {  
        conn.requestMethod = "GET"  
        conn.connectTimeout = 5_000  
        conn.readTimeout = 5_000  
        conn.setRequestProperty("Accept", "application/json")  
  
        val code = conn.responseCode  
        val stream = if (code in 200..299) conn.inputStream else conn.errorStream  
        val body = stream?.bufferedReader(Charsets.UTF_8)?.use { it.readText() } ?: ""  
        println("$code $body")  
    } finally {  
        conn.disconnect()  
    }  
}
```

HttpURLConnection

HttpURLConnection - низкоуровневый, блокирующий HTTP-клиент из пакета java.net. Он удобен для базовых задач и ситуаций, когда важно обойтись без внешних зависимостей, при этом API предоставляет минимум

Открытие
соединения по URL

```
fun main() {  
    val conn = URL("https://httpbin.org/get").openConnection() as HttpURLConnection  
    return try {  
        conn.requestMethod = "GET"  
        conn.connectTimeout = 5_000  
        conn.readTimeout = 5_000  
        conn.setRequestProperty("Accept", "application/json")  
  
        val code = conn.responseCode  
        val stream = if (code in 200..299) conn.inputStream else conn.errorStream  
        val body = stream?.bufferedReader(Charsets.UTF_8)?.use { it.readText() } ?: ""  
        println("$code $body")  
    } finally {  
        conn.disconnect()  
    }  
}
```

HttpURLConnection

HttpURLConnection - низкоуровневый, блокирующий HTTP-клиент из пакета java.net. Он удобен для базовых задач и ситуаций, когда важно обойтись без внешних зависимостей, при этом API предоставляет минимум удобств.

```
fun main() {  
    val conn = URL("https://httpbin.org/get").openConnection() as HttpURLConnection  
    return try {  
        conn.requestMethod = "GET"  
        conn.connectTimeout = 5_000  
        conn.readTimeout = 5_000  
        conn.setRequestProperty("Accept", "application/json")  
  
        val code = conn.responseCode  
        val stream = if (code in 200..299) conn.inputStream else conn.errorStream  
        val body = stream?.bufferedReader(Charsets.UTF_8)?.use { it.readText() } ?: ""  
        println("$code $body")  
    } finally {  
        conn.disconnect()  
    }  
}
```

Установка
параметров
соединения

HttpURLConnection

HttpURLConnection - низкоуровневый, блокирующий HTTP-клиент из пакета java.net. Он удобен для базовых задач и ситуаций, когда важно обойтись без внешних зависимостей, при этом API предоставляет минимум удобств.

```
fun main() {  
    val conn = URL("https://httpbin.org/get").openConnection() as HttpURLConnection  
    return try {  
        conn.requestMethod = "GET"  
        conn.connectTimeout = 5_000  
        conn.readTimeout = 5_000  
        conn.setRequestProperty("Accept", "application/json")  
  
        val code = conn.responseCode  
        val stream = if (code in 200..299) conn.inputStream else conn.errorStream  
        val body = stream?.bufferedReader(Charsets.UTF_8)?.use { it.readText() } ?: ""  
        println("$code $body")  
    } finally {  
        conn.disconnect()  
    }  
}
```

Выполнение запроса

HttpURLConnection

HttpURLConnection - низкоуровневый, блокирующий HTTP-клиент из пакета `java.net`. Он удобен для базовых задач и ситуаций, когда важно обойтись без внешних зависимостей, при этом API предоставляет минимум удобств.

```
fun main() {  
    val conn = URL("https://httpbin.org/get").openConnection() as HttpURLConnection  
    return try {  
        conn.requestMethod = "GET"  
        conn.connectTimeout = 5_000  
        conn.readTimeout = 5_000  
        conn.setRequestProperty("Accept", "application/json")  
  
        val code = conn.responseCode  
        val stream = if (code in 200..299) conn.inputStream else conn.errorStream  
        val body = stream?.bufferedReader(Charsets.UTF_8)?.use { it.readText() } ?: ""  
        println("$code $body")  
    } finally {  
        conn.disconnect()  
    }  
}
```

Чтение результата

HttpURLConnection

HttpURLConnection - низкоуровневый, блокирующий HTTP-клиент из пакета `java.net`. Он удобен для базовых задач и ситуаций, когда важно обойтись без внешних зависимостей, при этом API предоставляет минимум удобств.

```
fun main() {  
    val conn = URL("https://httpbin.org/get").openConnection() as HttpURLConnection  
    return try {  
        conn.requestMethod = "GET"  
        conn.connectTimeout = 5_000  
        conn.readTimeout = 5_000  
        conn.setRequestProperty("Accept", "application/json")  
  
        val code = conn.responseCode  
        val stream = if (code in 200..299) conn.inputStream else conn.errorStream  
        val body = stream?.bufferedReader()?.readText() ?: ""  
        println("$code $body")  
    } finally {  
        conn.disconnect()  
    }  
}
```

**Важно закрыть
соединение**

HttpClient

HttpClient – новый HTTP-клиент в JDK11+, поддерживает HTTP/1.1 и HTTP/2, синхронные и асинхронные, веб-сокеты, редиректы, прокси, аутентификацию, cookie.

HttpClient

HttpClient – новый HTTP-клиент в JDK11+, поддерживает HTTP/1.1 и HTTP/2, синхронные и асинхронные, веб-сокеты, редиректы, прокси, аутентификацию, cookie.

```
fun main() {  
    val client = HttpClient.newBuilder()  
        .followRedirects(HttpClient.Redirect.NORMAL)  
        .connectTimeout(Duration.ofSeconds(5))  
        .build()  
  
    val request = HttpRequest.newBuilder()  
        .uri(URI.create("https://httpbin.org/get"))  
        .header("Accept", "application/json")  
        .timeout(Duration.ofSeconds(5))  
        .GET()  
        .build()  
  
    val response = client.send(request, HttpResponse.BodyHandlers.ofString())  
    println("${response.statusCode} ${response.body()}")  
}
```

HttpClient

HttpClient – новый HTTP-клиент в JDK11+, поддерживает HTTP/1.1 и HTTP/2, синхронные и асинхронные, веб-сокеты, редиректы, прокси, аутентификацию, cookie.

```
fun main() {  
    val client = HttpClient.newBuilder()  
        .followRedirects(HttpClient.Redirect.NORMAL)  
        .connectTimeout(Duration.ofSeconds(5))  
        .build()  
  
    val request = HttpRequest.newBuilder()  
        .uri(URI.create("https://httpbin.org/get"))  
        .header("Accept", "application/json")  
        .timeout(Duration.ofSeconds(5))  
        .GET()  
        .build()  
  
    val response = client.send(request, HttpResponse.BodyHandlers.ofString())  
    println("${response.statusCode} ${response.body()}")  
}
```

Создание клиента

HttpClient

HttpClient – новый HTTP-клиент в JDK11+, поддерживает HTTP/1.1 и HTTP/2, синхронные и асинхронные, веб-сокеты, редиректы, прокси, аутентификацию, cookie.

```
fun main() {  
    val client = HttpClient.newBuilder()  
        .followRedirects(HttpClient.Redirect.NORMAL)  
        .connectTimeout(Duration.ofSeconds(5))  
        .build()  
  
    val request = HttpRequest.newBuilder()  
        .uri(URI.create("https://httpbin.org/get"))  
        .header("Accept", "application/json")  
        .timeout(Duration.ofSeconds(5))  
        .GET()  
        .build()  
  
    val response = client.send(request, HttpResponse.BodyHandlers.ofString())  
    println("${response.statusCode} ${response.body()}")  
}
```

Формирование
запроса

HttpClient

HttpClient – новый HTTP-клиент в JDK11+, поддерживает HTTP/1.1 и HTTP/2, синхронные и асинхронные, веб-сокеты, редиректы, прокси, аутентификацию, cookie.

```
fun main() {  
    val client = HttpClient.newBuilder()  
        .followRedirects(HttpClient.Redirect.NORMAL)  
        .connectTimeout(Duration.ofSeconds(5))  
        .build()  
  
    val request = HttpRequest.newBuilder()  
        .uri(URI.create("https://httpbin.org/get"))  
        .header("Accept", "application/json")  
        .timeout(Duration.ofSeconds(5))  
        .GET()  
        .build()  
  
    val response = client.send(request, HttpResponse.BodyHandlers.ofString())  
    println("${response.statusCode} ${response.body()}")  
}
```

Отправка запроса и
получение ответа

HttpServer

HttpServer - минималистичный встроенный HTTP-сервер. Подходит для простых сервисов, тестовых стендов и встраивания в приложения без внешних зависимостей.

HttpServer

HttpServer - минималистичный встроенный HTTP-сервер. Подходит для простых сервисов, тестовых стендов и встраивания в приложения без внешних зависимостей.

```
fun main() {  
    val server = HttpServer.create(InetSocketAddress(8080), 0)  
  
    server.createContext("/hello") { exchange ->  
        try {  
            val response = """"{"message":"Hello from server"}""""  
            val bytes = response.toByteArray(StandardCharsets.UTF_8)  
            exchange.responseHeaders.add("Content-Type", "application/json; charset=UTF-8")  
            exchange.sendResponseHeaders(200, bytes.size.toLong())  
            exchange.responseBody.use { it.write(bytes) }  
        } finally {  
            exchange.close()  
        }  
    }  
}  
  
server.executor = Executors.newFixedThreadPool(Runtime.getRuntime().availableProcessors())  
server.start()  
Runtime.getRuntime().addShutdownHook(Thread { server.stop(0) })  
}
```


HttpServer

HttpServer - минималистичный встроенный HTTP-сервер. Подходит для простых сервисов, тестовых стендов и встраивания в приложения без внешних зависимостей.

```
fun main() {  
    val server = HttpServer.create(InetSocketAddress(8080), 0)  
  
    server.createContext("/hello") { exchange ->  
        try {  
            val response = """"{"message":"Hello from server}""""  
            val bytes = response.toByteArray(StandardCharsets.UTF_8)  
            exchange.responseHeaders.add("Content-Type", "application/json; charset=UTF-8")  
            exchange.sendResponseHeaders(200, bytes.size.toLong())  
            exchange.responseBody.use { it.write(bytes) }  
        } finally {  
            exchange.close()  
        }  
    }  
}  
  
server.executor = Executors.newFixedThreadPool(Runtime.getRuntime().availableProcessors())  
server.start()  
Runtime.getRuntime().addShutdownHook(Thread { server.stop(0) })  
}
```

Создание пула
обработчиков и старт
сервера

HttpServer

HttpServer - минималистичный встроенный HTTP-сервер. Подходит для простых сервисов, тестовых стендов и встраивания в приложения без внешних зависимостей.

```
fun main() {  
    val server = HttpServer.create(InetSocketAddress(8080), 0)  
  
    server.createContext("/hello") { exchange ->  
        try {  
            val response = """"{"message":"Hello from server"}""""  
            val bytes = response.toByteArray(StandardCharsets.UTF_8)  
            exchange.responseHeaders.add("Content-Type", "application/json; charset=UTF-8")  
            exchange.sendResponseHeaders(200, bytes.size.toLong())  
            exchange.responseBody.use { it.write(bytes) }  
        } finally {  
            exchange.close()  
        }  
    }  
  
    server.executor = Executors.newFixedThreadPool(Runtime.getRuntime().availableProcessors())  
    server.start()  
    Runtime.getRuntime().addShutdownHook(Thread { server.stop(0) })  
}
```

Создание эндпоинта
/hello

HttpServer

HttpServer - минималистичный встроенный HTTP-сервер. Подходит для простых сервисов, тестовых стендов и встраивания в приложения без внешних зависимостей.

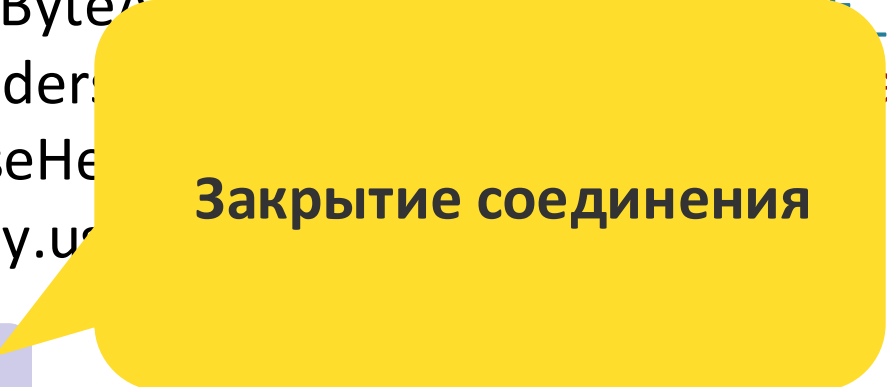
```
fun main() {  
    val server = HttpServer.create(InetSocketAddress(8080), 0)  
  
    server.createContext("/hello") { exchange ->  
        try {  
            val response = """"{"message":"Hello from server"}"""  
            val bytes = response.toByteArray(StandardCharsets.UTF_8)  
            exchange.responseHeaders.add("Content-Type", "application/json; charset=UTF-8")  
            exchange.sendResponseHeaders(200, bytes.size.toLong())  
            exchange.responseBody.use { it.write(bytes) }  
        } finally {  
            exchange.close()  
        }  
    }  
}  
  
server.executor = Executors.newFixedThreadPool(Runtime.getRuntime().availableProcessors())  
server.start()  
Runtime.getRuntime().addShutdownHook(Thread { server.stop(0) })  
}
```

Обработка запроса

HttpServer

HttpServer - минималистичный встроенный HTTP-сервер. Подходит для простых сервисов, тестовых стендов и встраивания в приложения без внешних зависимостей.

```
fun main() {  
    val server = HttpServer.create(InetSocketAddress(8080), 0)  
  
    server.createContext("/hello") { exchange ->  
        try {  
            val response = """"{"message":"Hello from server"}""""  
            val bytes = response.toByteArray(StandardCharsets.UTF_8)  
            exchange.responseHeaders.put("Content-Type", "application/json; charset=UTF-8")  
            exchange.sendResponseHeaders(200, bytes.length)  
            exchange.responseBody.write(bytes)  
        } finally {  
            exchange.close()  
        }  
    }  
  
    server.executor = Executors.newFixedThreadPool(Runtime.getRuntime().availableProcessors())  
    server.start()  
    Runtime.getRuntime().addShutdownHook(Thread { server.stop(0) })  
}
```



Summary

- `HttpURLConnection` - базовый HTTP-клиент без внешних зависимостей. Простое открытие соединения, установка параметров, чтение ответа.
- `HttpClient` - современный клиент с поддержкой HTTP/2, редиректов и асинхронности.
- `HttpServer` - минималистичный встроенный сервер для тестов и простых сервисов.

**Ваши
вопросы**



Практика



Http engine

HTTP-движок - это низкоуровневая реализация HTTP-клиента, на него опираются библиотеки и фреймворки верхнего уровня, предоставляя удобный API, а сам движок отвечает за установление TCP/TLS-соединений, поддержку протоколов.

OkHttp

OkHttp - высокопроизводительный HTTP-клиент от Square, стандарт в Android и популярная база для библиотек вроде Retrofit и Ktor Client. Он поддерживает HTTP/1.1 и HTTP/2, WebSocket, пул соединений и работу с сетевыми ошибками и редиректами, поддерживает интерсепторы, дисковый HTTP-кэш, CookieJar, события, расширенные возможности TLS и пиннинг сертификатов.

OkHttp

OkHttp - высокопроизводительный HTTP-клиент от Square, стандарт в Android и популярная база для библиотек вроде Retrofit и Ktor Client. Он поддерживает HTTP/1.1 и HTTP/2, WebSocket, пул соединений и работу с сетевыми ошибками и редиректами, поддерживает интерсепторы, дисковый HTTP-кэш, CookieJar, события, расширенные возможности TLS и пиннинг сертификатов.

```
fun main() {  
    val client = OkHttpClient()  
  
    val request = Request.Builder().url("https://httpbin.org/get").build()  
  
    client.newCall(request).execute().use { resp ->  
        if (!resp.isSuccessful) {  
            println("Ошибка: ${resp.code}")  
        } else {  
            println(resp.body?.string())  
        }  
    }  
}
```

OkHttp

OkHttp - высокопроизводительный HTTP-клиент от Square, стандарт в Android и популярная база для библиотек вроде Retrofit и Ktor Client. Он поддерживает HTTP/1.1 и HTTP/2, WebSocket, пул соединений и работу с сетевыми ошибками и редиректами, поддерживает интерсепторы, дисковый HTTP-кэш, CookieJar, события, расширенные возможности TLS и пиннинг сертификатов.

```
fun main() {  
    val client = OkHttpClient()  
  
    val request = Request.Builder().url("https://httpbin.org/get").build()  
  
    client.newCall(request).execute().use { resp ->  
        if (!resp.isSuccessful) {  
            println("Ошибка: ${resp.code}")  
        } else {  
            println(resp.body?.string())  
        }  
    }  
}
```

Создание клиента

OkHttp

OkHttp - высокопроизводительный HTTP-клиент от Square, стандарт в Android и популярная база для библиотек вроде Retrofit и Ktor Client. Он поддерживает HTTP/1.1 и HTTP/2, WebSocket, пул соединений и работу с сетевыми ошибками и редиректами, поддерживает интерсепторы, дисковый HTTP-кэш, CookieJar, события, расширенные возможности TLS и пиннинг сертификатов.

```
fun main() {  
    val client = OkHttpClient()  
  
    val request = Request.Builder().url("https://httpbin.org/get").build()  
  
    client.newCall(request).execute().use { resp ->  
        if (!resp.isSuccessful) {  
            println("Ошибка: ${resp.code}")  
        } else {  
            println(resp.body?.string())  
        }  
    }  
}
```

Формирование
запроса

OkHttp

OkHttp - высокопроизводительный HTTP-клиент от Square, стандарт в Android и популярная база для библиотек вроде Retrofit и Ktor Client. Он поддерживает HTTP/1.1 и HTTP/2, WebSocket, пул соединений и работу с сетевыми ошибками и редиректами, поддерживает интерсепторы, дисковый HTTP-кэш, CookieJar, события, расширенные возможности TLS и пиннинг сертификатов.

```
fun main() {  
    val client = OkHttpClient()  
  
    val request = Request.Builder().url("https://httpbin.org/get").build()  
  
    client.newCall(request).execute().use { resp ->  
        if (!resp.isSuccessful) {  
            println("Ошибка: ${resp.code}")  
        } else {  
            println(resp.body?.string())  
        }  
    }  
}
```

Отправка запроса и
получение ответа

OkHttp Interceptor

Interceptor - это перехватчик в OkHttp, который оборачивает выполнение HTTP-запроса и ответа. Он позволяет централизованно добавлять или изменять заголовки, выполнять простое логирование, применять аутентификацию или иные политики перед тем, как запрос отправится.

OkHttp Interceptor

Interceptor - это перехватчик в OkHttp, который оборачивает выполнение HTTP-запроса и ответа. Он позволяет централизованно добавлять или изменять заголовки, выполнять простое логирование, применять аутентификацию или иные политики перед тем, как запрос отправится.

```
class SimpleAuthInterceptor(val token: String?) : Interceptor {
    override fun intercept(chain: Interceptor.Chain): Response {
        val request = chain.request()
        val newRequest = if (!token.isNullOrEmpty()) {
            request.newBuilder()
                .header("Authorization", "Bearer $token")
                .build()
        } else {
            request
        }
        return chain.proceed(newRequest)
    }
}

fun main() {
    val client = OkHttpClient.Builder()
        .addInterceptor(SimpleAuthInterceptor(token = "Auth token"))
        .build()
}
```

OkHttp Interceptor

Interceptor - это перехватчик в OkHttp, который оборачивает выполнение HTTP-запроса. Он позволяет централизованно добавлять или изменять заголовки, выполнять простое логирование, проверять ответы или иные политики перед тем, как запрос отправится.

Класс интерсептора,
наследуется от
Intecceptor

```
class SimpleAuthInterceptor(val token: String?) : Interceptor {  
    override fun intercept(chain: Interceptor.Chain): Response {  
        val request = chain.request()  
        val newRequest = if (!token.isNullOrEmpty()) {  
            request.newBuilder()  
                .header("Authorization", "Bearer $token")  
                .build()  
        } else {  
            request  
        }  
        return chain.proceed(newRequest)  
    }  
}  
  
fun main() {  
    val client = OkHttpClient.Builder()  
        .addInterceptor(SimpleAuthInterceptor(token = "Auth token"))  
        .build()  
}
```


OkHttp Interceptor

Interceptor - это перехватчик в OkHttp, который оборачивает выполнение HTTP-запроса и ответа. Он позволяет централизованно добавлять или изменять заголовки, выполнять простое логирование, применять аутентификацию или иные политики перед тем, как запрос отправится.

```
class SimpleAuthInterceptor(val token: String?) : Interceptor {  
    override fun intercept(chain: Interceptor.Chain): Response {  
        val request = chain.request()  
        val newRequest = if (!token.isNullOrEmpty()) {  
            request.newBuilder()  
                .header("Authorization", "Bearer $token")  
                .build()  
        } else {  
            request  
        }  
        return chain.proceed(newRequest)  
    }  
}
```

Добавление
заголовка
авторизации

```
fun main() {  
    val client = OkHttpClient.Builder()  
        .addInterceptor(SimpleAuthInterceptor(token = "Auth token"))  
        .build()  
}
```

OkHttp Interceptor

Interceptor - это перехватчик в OkHttp, который оборачивает выполнение HTTP-запроса и ответа. Он позволяет централизованно добавлять или изменять заголовки, выполнять простое логирование, применять аутентификацию или иные политики перед тем, как запрос отправится.

```
class SimpleAuthInterceptor(val token: String?) : Interceptor {  
    override fun intercept(chain: Interceptor.Chain): Response {  
        val request = chain.request()  
        val newRequest = if (!token.isNullOrEmpty()) {  
            request.newBuilder()  
                .header("Authorization", "Bearer $token")  
                .build()  
        } else {  
            request  
        }  
        return chain.proceed(newRequest)  
    }  
}
```

```
fun main() {  
    val client = OkHttpClient.Builder()  
        .addInterceptor(SimpleAuthInterceptor(token = "Auth token"))  
        .build()  
}
```

Создание
обновленного
запроса

OkHttp Interceptor

Interceptor - это перехватчик в OkHttp, который оборачивает выполнение HTTP-запроса и ответа. Он позволяет централизованно добавлять или изменять заголовки, выполнять простое логирование, применять аутентификацию или иные политики перед тем, как запрос отправится.

```
class SimpleAuthInterceptor(val token: String?) : Interceptor {  
    override fun intercept(chain: Interceptor.Chain): Response {  
        val request = chain.request()  
        val newRequest = if (!token.isNullOrEmpty()) {  
            request.newBuilder()  
                .header("Authorization", "Bearer $token")  
                .build()  
        } else {  
            request  
        }  
        return chain.proceed(newRequest)  
    }  
}
```

```
fun main() {  
    val client = OkHttpClient.Builder()  
        .addInterceptor(SimpleAuthInterceptor(token = "Auth token"))  
        .build()  
}
```

Добавление в клиент

Ktor

Ktor - это Kotlin-first фреймворк от JetBrains для создания сетевых приложений. Он состоит из двух частей: Ktor Server и Ktor Client. Архитектура основана на плагинах, глубоко интегрирована с корутинами и поддерживает мультиплатформенность. За счёт легковесности и гибкости Ktor удобен для микросервисов, CLI-утилит, интеграций и мобильных клиентов.

KtorClnet

Ktor Client работает поверх разных «движков» (CIO, OkHttp, Java HttpClient), поддерживает HTTP/1.1 и HTTP/2, WebSocket, таймауты, ретраи, куки и аутентификацию.

KtorClnet

Ktor Client работает поверх разных движков (CIO, OkHttp, Java HttpClient), поддерживает HTTP/1.1 и HTTP/2, WebSocket, таймауты, ретраи, куки и аутентификацию.

```
fun main() = runBlocking {
    val client = HttpClient(OkHttp) {
        install(HttpTimeout) { requestTimeoutMillis = 5_000 }

        engine {
            config {
                retryOnConnectionFailure(true)
                connectTimeout(5, TimeUnit.SECONDS)
                readTimeout(5, TimeUnit.SECONDS)

                val logging = HttpLoggingInterceptor().apply {
                    level = HttpLoggingInterceptor.Level.BASIC
                }
                addInterceptor(logging)
            }
        }

        defaultRequest {
            header("Accept", "application/json")
        }
    }

    val getResp = client.get("https://httpbin.org/get")
    println("${getResp.status} ${getResp.bodyAsText()}")

    client.close()
}
```

KtorClnet

Ktor Client работает поверх разных движков (OkHttp, Apache HttpClient), поддерживает HTTP/1.1 и HTTP/2, WebSocket, таймауты, ретраи, куки и авторизацию.

Установка Http
движка

```
fun main() = runBlocking {
    val client = HttpClient(OkHttp) {
        install(HttpTimeout) { requestTimeoutMillis = 5_000 }

        engine {
            config {
                retryOnConnectionFailure(true)
                connectTimeout(5, TimeUnit.SECONDS)
                readTimeout(5, TimeUnit.SECONDS)

                val logging = HttpLoggingInterceptor().apply {
                    level = HttpLoggingInterceptor.Level.BASIC
                }
                addInterceptor(logging)
            }
        }

        defaultRequest {
            header("Accept", "application/json")
        }
    }

    val getResp = client.get("https://httpbin.org/get")
    println("${getResp.status} ${getResp.bodyAsText()}")

    client.close()
}
```

KtorClnet

Ktor Client работает поверх разных движков (CIO, OkHttp, Java I/O) и поддерживает HTTP/1.1 и HTTP/2, WebSocket, таймауты, ретраи, куки и аутентификацию.

```
fun main() = runBlocking {
    val client = HttpClient(OkHttp) {
        install(HttpTimeout) { requestTimeoutMillis = 5_000 }

        engine {
            config {
                retryOnConnectionFailure(true)
                connectTimeout(5, TimeUnit.SECONDS)
                readTimeout(5, TimeUnit.SECONDS)

                val logging = HttpLoggingInterceptor().apply {
                    level = HttpLoggingInterceptor.Level.BASIC
                }
                addInterceptor(logging)
            }
        }

        defaultRequest {
            header("Accept", "application/json")
        }
    }

    val getResp = client.get("https://httpbin.org/get")
    println("${getResp.status} ${getResp.bodyAsText()}")

    client.close()
}
```

Установка Ktor
плагинов

KtorClnet

Ktor Client работает поверх разных движков (CIO, OkHttp, Java HttpClient), поддерживает HTTP/1.1 и HTTP/2, WebSocket, таймауты, ретраи, куки и аутентификацию.

```
fun main() = runBlocking {  
    val client = HttpClient(OkHttp) {  
        install(HttpTimeout) { requestTimeoutMillis = 5_000 }  
  
        engine {  
            config {  
                retryOnConnectionFailure(true)  
                connectTimeout(5, TimeUnit.SECONDS)  
                readTimeout(5, TimeUnit.SECONDS)  
  
                val logging = HttpLoggingInterceptor().apply {  
                    level = HttpLoggingInterceptor.Level.BASIC  
                }  
                addInterceptor(logging)  
            }  
        }  
    }  
  
    defaultRequest {  
        header("Accept", "application/json")  
    }  
}  
  
val getResp = client.get("https://httpbin.org/get")  
println("${getResp.status} ${getResp.bodyAsText()}")  
  
client.close()  
}
```

Настройка OkHttp

KtorClnet

Ktor Client работает поверх разных движков (CIO, OkHttp, Java HttpClient), поддерживает HTTP/1.1 и HTTP/2, WebSocket, таймауты, ретраи, куки и аутентификацию.

```
fun main() = runBlocking {  
    val client = HttpClient(OkHttp) {  
        install(HttpTimeout) { requestTimeoutMillis = 5_000 }  
  
        engine {  
            config {  
                retryOnConnectionFailure(true)  
                connectTimeout(5, TimeUnit.SECONDS)  
                readTimeout(5, TimeUnit.SECONDS)  
  
                val logging = HttpLoggingInterceptor().apply {  
                    level = HttpLoggingInterceptor.Level.BASIC  
                }  
                addInterceptor(logging)  
            }  
        }  
  
        defaultRequest {  
            header("Accept", "application/json")  
        }  
    }  
  
    val getResp = client.get("https://httpbin.org/get")  
    println("${getResp.status} ${getResp.bodyAsText()}")  
  
    client.close()  
}
```

Отправка запроса,
получение ответа и
заккрытие соединения

KtorServer

Ktor Server даёт роутинг, фильтры/плагины, сериализацию и WebSocket.

KtorServer

Ktor Server даёт роутинг, фильтры/плагины, сериализацию и WebSocket.

```
fun main() {  
    embeddedServer(CIO, port = 8080, module = Application::module)  
        .start(wait = true)  
}  
  
fun Application.module() {  
    install(CallLogging)  
  
    routing {  
        get("/hello") {  
            call.respondText("{\\\"message\\\": \\\"Hello for server\\\"}", ContentType.Application.json)  
        }  
    }  
}
```

KtorServer

Ktor Server даёт роутинг, фильтры/плагины, сериализацию и WebSocket.

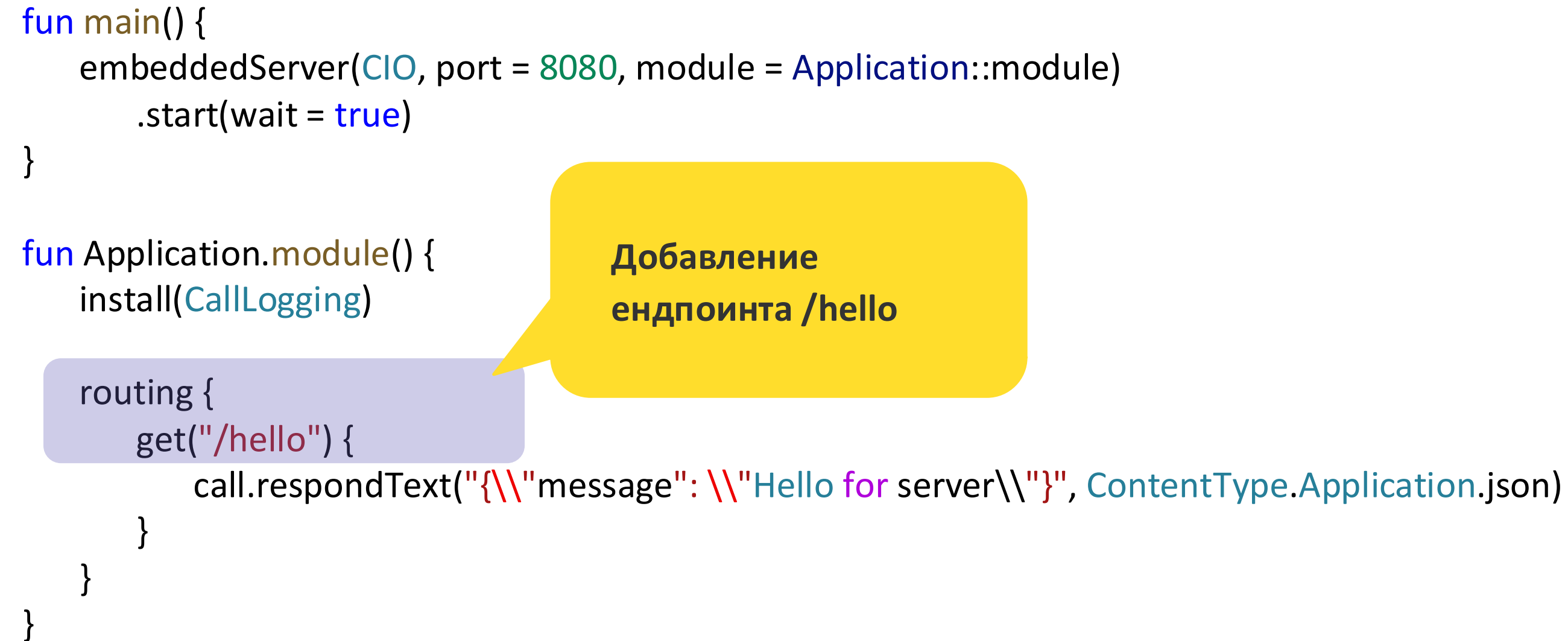
```
fun main() {  
    embeddedServer(CIO, port = 8080, module = Application::module)  
        .start(wait = true)  
}  
  
fun Application.module() {  
    install(CallLogging)  
  
    routing {  
        get("/hello") {  
            call.respondText("{\\\"message\\\": \\\"Hello for server\\\"}", ContentType.Application.json)  
        }  
    }  
}
```

Старт сервера

KtorServer

Ktor Server даёт роутинг, фильтры/плагины, сериализацию и WebSocket.

```
fun main() {  
    embeddedServer(CIO, port = 8080, module = Application::module)  
        .start(wait = true)  
}  
  
fun Application.module() {  
    install(CallLogging)  
  
    routing {  
        get("/hello") {  
            call.respondText("{\\\"message\\\": \\\"Hello for server\\\"}", ContentType.Application.json)  
        }  
    }  
}
```



Добавление
эндпоинта /hello

KtorServer

Ktor Server даёт роутинг, фильтры/плагины, сериализацию и WebSocket.

```
fun main() {  
    embeddedServer(CIO, port = 8080, module = Application::module)  
        .start(wait = true)  
}  
  
fun Application.module() {  
    install(CallLogging)  
  
    routing {  
        get("/hello") {  
            call.respondText("{\\\"message\\\": \\\"Hello for server\\\"}", ContentType.Application.json)  
        }  
    }  
}
```

Обработка запроса

Retrofit

Типобезопасный HTTP-клиент для Android и Java. Retrofit превращает ваш HTTP API в интерфейс Java (или Kotlin).

[Документация](https://square.github.io/retrofit/)

<https://square.github.io/retrofit/>

[Github](https://github.com/square/retrofit)

<https://github.com/square/retrofit>

Summary

- HTTP Engine - базовый слой, на котором строятся высокоуровневые библиотеки.
- OkHttp - быстрый и гибкий HTTP-клиент для JVM и Android. Поддержка HTTP/2, WebSocket, кэша, интерсепторов, TLS.
- Interceptor - перехватчик запросов/ответов (например, для авторизации).
- Ktor - Kotlin-first фреймворк для клиента и сервера.
 - Client: Поддержка разных движков (CIO, OkHttp, Java HttpClient). Гибкая конфигурация через плагины и интерсепторы.
 - Server: Простая настройка роутинга, плагинов, логирования и сериализации. Быстрый старт и определение эндпоинтов.

**Ваши
вопросы**



Практика



JSON сериализация

Kotlin.serialization

kotlin.serialization - мультиплатформенная библиотека сериализации от JetBrains для Kotlin. Генерирует код на этапе компиляции без рефлексии, поддерживает JSON и другие протоколы.

Kotlin.serialization

kotlinx.serialization - мультиплатформенная библиотека сериализации от JetBrains для Kotlin. Генерирует код на этапе компиляции без рефлексии, поддерживает JSON и другие протоколы.

@Serializable

```
data class User(  
    val id: Int,  
    val name: String,  
    val email: String? = null  
)
```

```
fun main() {  
    val json = Json {  
        prettyPrint = true  
        ignoreUnknownKeys = true  
    }
```

```
    val u = User(1, "Alice")  
    val asString: String = json.encodeToString(u)  
    println(asString)
```

```
    val input = """"{ "id": 2, "name": "Bob", "extra": 42 }"""  
    val parsed: User = json.decodeFromString(input)  
    println(parsed)  
}
```

Kotlin.serialization

kotlinx.serialization — это библиотека сериализации от JetBrains для Kotlin. Генерирует код на этапе компиляции. Поддерживает JSON и другие протоколы.

Аннотация для
сериализации

@Serializable

```
data class User(  
    val id: Int,  
    val name: String,  
    val email: String? = null  
)
```

```
fun main() {  
    val json = Json {  
        prettyPrint = true  
        ignoreUnknownKeys = true  
    }
```

```
    val u = User(1, "Alice")  
    val asString: String = json.encodeToString(u)  
    println(asString)
```

```
    val input = """"{ "id": 2, "name": "Bob", "extra": 42 }""""  
    val parsed: User = json.decodeFromString(input)  
    println(parsed)  
}
```

Kotlin.serialization

kotlinx.serialization - мультиплатформенная библиотека сериализации от JetBrains для Kotlin. Генерирует код на этапе компиляции без рефлексии, поддерживает JSON и другие протоколы.

@Serializable

```
data class User(  
    val id: Int,  
    val name: String,  
    val email: String? = null  
)
```

```
fun main() {  
    val json = Json {  
        prettyPrint = true  
        ignoreUnknownKeys = true  
    }
```

Создание и настройка
сериализатора

```
    val u = User(1, "Alice")  
    val asString: String = json.encodeToString(u)  
    println(asString)
```

```
    val input = """"{ "id": 2, "name": "Bob", "extra": 42 }""""  
    val parsed: User = json.decodeFromString(input)  
    println(parsed)  
}
```


Kotlin.serialization

kotlinx.serialization - мультиплатформенная библиотека сериализации от JetBrains для Kotlin. Генерирует код на этапе компиляции без рефлексии, поддерживает JSON и другие протоколы.

@Serializable

```
data class User(  
    val id: Int,  
    val name: String,  
    val email: String? = null  
)
```

```
fun main() {  
    val json = Json {  
        prettyPrint = true  
        ignoreUnknownKeys = true  
    }
```

```
    val u = User(1, "Alice")  
    val asString: String = json.encodeToString(u)  
    println(asString)
```

```
    val input = """"{ "id": 2, "name": "Bob", "extra": 42 }""""  
    val parsed: User = json.decodeFromString(input)  
    println(parsed)  
}
```

Сериализация
объекта

Kotlin.serialization

kotlinx.serialization - мультиплатформенная библиотека сериализации от JetBrains для Kotlin. Генерирует код на этапе компиляции без рефлексии, поддерживает JSON и другие протоколы.

@Serializable

```
data class User(  
    val id: Int,  
    val name: String,  
    val email: String? = null  
)
```

```
fun main() {  
    val json = Json {  
        prettyPrint = true  
        ignoreUnknownKeys = true  
    }
```

```
    val u = User(1, "Alice")  
    val asString: String = json.encodeToString(u)  
    println(asString)
```

```
    val input = """"{ "id": 2, "name": "Bob", "extra": 42 }""""  
    val parsed: User = json.decodeFromString(input)  
    println(parsed)  
}
```

Десериализация
объекта

Ktor + Kotlinx.serialization

Добавление сериализации Json в KtorClient при помощи плагина `kotlinx.serialization`

Ktor + Kotlinx.serialization

Добавление сериализации Json в KtorClient при помощи плагина kotlinx.serialization

```
@Serializable
data class Todo(
    val userId: Int,
    val id: Int,
    val title: String,
    val completed: Boolean
)

fun main() = runBlocking {
    val client = HttpClient(OkHttp) {
        install(ContentNegotiation) {
            json(Json {
                ignoreUnknownKeys = true
            })
        }
    }

    val todo: Todo = client.get("https://jsonplaceholder.typicode.com/todos/1").body()
    println(todo)

    client.close()
}
```

Ktor + Kotlinx.serialization

Добавление сериализации Json

плагина kotlinx.serialization

Модель данных

```
@Serializable
data class Todo(
    val userId: Int,
    val id: Int,
    val title: String,
    val completed: Boolean
)
```

```
fun main() = runBlocking {
    val client = HttpClient(OkHttp) {
        install(ContentNegotiation) {
            json(Json {
                ignoreUnknownKeys = true
            })
        }
    }

    val todo: Todo = client.get("https://jsonplaceholder.typicode.com/todos/1").body()
    println(todo)

    client.close()
}
```

Ktor + Kotlin.serialization

Добавление сериализации Json в KtorClient при помощи плагина kotlin.serialization

`@Serializable`

```
data class Todo(  
    val userId: Int,  
    val id: Int,  
    val title: String,  
    val completed: Boolean  
)
```

```
fun main() = runBlocking {  
    val client = HttpClient(OkHttp) {  
        install(ContentNegotiation) {  
            json(Json {  
                ignoreUnknownKeys = true  
            })  
        }  
    }  
}
```

Добавление плагина
для десериализации
Json

```
val todo: Todo = client.get("https://jsonplaceholder.typicode.com/todos/1").body()  
println(todo)  
  
client.close()  
}
```

Ktor + Kotlin.serialization

Добавление сериализации Json в KtorClient при помощи плагина kotlin.serialization

```
@Serializable
data class Todo(
    val userId: Int,
    val id: Int,
    val title: String,
    val completed: Boolean
)

fun main() = runBlocking {
    val client = HttpClient(OkHttp) {
        install(ContentNegotiation) {
            json(Json {
                ignoreUnknownKeys = true
            })
        }
    }

    val todo: Todo = client.get("https://jsonplaceholder.typicode.com/todos/1").body()
    println(todo)

    client.close()
}
```

Получение ответа
сразу в виде модели

Альтернативы?

1. Gson — библиотека от Google, использует рефлексия, проста в применении, хуже чем `kotlinx.serialization` по поддержке Kotlin-типов и по производительности
2. Moshi — библиотека от Square, работает через рефлексия или генерацию кода, дружелюбнее к Kotlin чем Gson, но ограничена JVM и без codegen медленнее

Summary

- `kotlinx.serialization` - библиотека от JetBrains для сериализации в Kotlin.
 - Генерация кода на этапе компиляции.
 - Поддержка JSON и других форматов.
 - Основные операции: аннотация `@Serializable`, настройка сериализатора, сериализация и десериализация объектов.
- Альтернативы: Gson, Moshi.

**Ваши
вопросы**



Практика





Спасибо!

