要使用 FastAPI 和 Retrofit (Kotlin) 实现长连接并传输 `ByteArray`，你可以利用 WebSocket 来处理长连接，而 FastAPI 和 Retrofit 各自负责服务端和客户端的实现。下面是一个完整的例子。

1. FastAPI 服务端

FastAPI 支持 WebSocket，可以用来处理二进制数据（`ByteArray`）的传输。下面是一个简单的 FastAPI 服务端实现，它接收来自客户端的 `ByteArray`，并将其原样返回。

安装依赖

```
pip install fastapi uvicorn
```

服务端代码

```
from fastapi import FastAPI, WebSocket, WebSocketDisconnect
from typing import List

app = FastAPI()

class ConnectionManager:
    def __init__(self):
        self.active_connections: List[WebSocket] = []

    async def connect(self, websocket: WebSocket):
        await websocket.accept()
        self.active_connections.append(websocket)

    def disconnect(self, websocket: WebSocket):
        self.active_connections.remove(websocket)

    async def send_personal_message(self, data: bytes, websocket:
WebSocket):
        await websocket.send_bytes(data)

    async def broadcast(self, data: bytes):
        for connection in self.active_connections:
            await connection.send_bytes(data)

manager = ConnectionManager()

@app.websocket("/ws")
async def websocket_endpoint(websocket: WebSocket):
    await manager.connect(websocket)
    try:
```

```
        while True:
            data = await websocket.receive_bytes()
            print(f"Received {len(data)} bytes")
            await manager.send_personal_message(data, websocket)  # Echo
back the data
    except WebSocketDisconnect:
        manager.disconnect(websocket)
```

2. Retrofit 客户端 (Kotlin)

在客户端，可以使用 Retrofit 的 WebSocket 支持来建立长连接并传输 `ByteArray` 数据。

依赖配置

首先，在 `build.gradle.kts` 中添加 Retrofit 和 OkHttp 的依赖：

```
dependencies {
    implementation("com.squareup.okhttp3:okhttp:4.9.1")
    implementation("com.squareup.retrofit2:retrofit:2.9.0")
    implementation("com.squareup.retrofit2:converter-gson:2.9.0")
    implementation("org.jetbrains.kotlinx:kotlinx-coroutines-core:1.5.1")
}
```

客户端代码

下面是一个使用 OkHttp 和 Retrofit 实现 WebSocket 客户端的示例，该客户端将 `ByteArray` 数据发送到 FastAPI 服务端，并接收响应。

```
import okhttp3.OkHttpClient
import okhttp3.Request
import okhttp3.WebSocket
import okhttp3.WebSocketListener
import okio.ByteString

class WebSocketClient : WebSocketListener() {

    private val client = OkHttpClient()

    fun start() {
        val request = Request.Builder()
            .url("ws://localhost:8000/ws")
            .build()

        val webSocket = client.newWebSocket(request, this)
```

```
        // 发送 ByteArray 数据
        val data = "Hello, FastAPI!".toByteArray()
        webSocket.send(ByteString.of(*data))
    }

    override fun onMessage(webSocket: WebSocket, bytes: ByteString) {
        println("Received: ${bytes.size()} bytes")
    }

    override fun onOpen(webSocket: WebSocket, response: okhttp3.Response) {
        println("WebSocket Connected")
    }

    override fun onFailure(webSocket: WebSocket, t: Throwable, response:
okhttp3.Response?) {
        t.printStackTrace()
    }

    override fun onClosing(webSocket: WebSocket, code: Int, reason: String)
{
        webSocket.close(1000, null)
        println("WebSocket Closing: $code / $reason")
    }

    override fun onClosed(webSocket: WebSocket, code: Int, reason: String)
{
        println("WebSocket Closed: $code / $reason")
    }
}

fun main() {
    val webSocketClient = WebSocketClient()
    webSocketClient.start()

    // Keep the JVM alive to allow the WebSocket to operate
    Thread.sleep(10000)
}
```

3. 运行服务端和客户端

1. **运行 FastAPI 服务端**：

```
uvicorn your_script:app --host 0.0.0.0 --port 8000
```

2. **运行 Kotlin 客户端**：

使用你的 IDE 或直接通过 `main()` 函数运行客户端代码。它会发送 `ByteArray` 数据到 FastAPI，并在控制台中打印出接收到的数据大小。

总结

这个设置展示了如何使用 FastAPI 处理 WebSocket 长连接，并使用 Retrofit/OkHttp 在 Kotlin 客户端中传输 `ByteArray` 数据。这种方式适合需要实时通信和二进制数据传输的应用场景，如实时音视频传输、文件传输等。

要在 Python 中使用 `http.server` 模块实现 HTTP 请求转发到一个 WebSocket 连接，通常会用到多种技术结合，例如使用 `http.server` 处理 HTTP 请求，然后将请求内容转发给一个 WebSocket 服务器或客户端。

不过，Python 标准库中的 `http.server` 只支持基本的 HTTP 功能，并不直接支持 WebSocket。所以你通常需要使用 `websockets` 或 `websocket-client` 等库来实现 WebSocket 通信。

方案概述

1. **HTTP Server**: 使用 `http.server` 模块接收 HTTP 请求。
2. **WebSocket Client**: 使用 `websockets` 或 `websocket-client` 库连接到 WebSocket 服务器，并将数据转发过去。

示例实现

下面是一个简单的示例，展示如何使用 `http.server` 接收 HTTP 请求，并将请求数据转发到 WebSocket 服务器。

安装依赖

```
pip install websocket-client
```

Python 代码实现

```python
from http.server import BaseHTTPRequestHandler, HTTPServer
import threading
import websocket

# 连接 WebSocket 服务端的 URL
WEBSOCKET_URL = "ws://localhost:8000/ws"

# WebSocket 客户端
class WebSocketClient:
    def __init__(self, url):
        self.ws = websocket.WebSocketApp(url,
                                         on_message=self.on_message,
                                         on_error=self.on_error,
                                         on_close=self.on_close)
        self.thread = threading.Thread(target=self.ws.run_forever)
        self.thread.daemon = True
        self.thread.start()

    def on_message(self, message):
        print(f"Received from WebSocket: {message}")

    def on_error(self, error):
        print(f"WebSocket Error: {error}")

    def on_close(self):
        print("WebSocket Closed")

    def send(self, message):
        self.ws.send(message)

# 初始化 WebSocket 客户端
ws_client = WebSocketClient(WEBSOCKET_URL)

# HTTP 服务器处理程序
class RequestHandler(BaseHTTPRequestHandler):
    def do_POST(self):
        content_length = int(self.headers['Content-Length'])
        post_data = self.rfile.read(content_length)

        # 打印接收到的 HTTP 请求内容
        print(f"Received HTTP POST data: {post_data}")

        # 将数据转发到 WebSocket
        ws_client.send(post_data)

        # 返回响应给客户端
        self.send_response(200)
```

```
        self.end_headers()
        self.wfile.write(b'HTTP request forwarded to WebSocket')

# 启动 HTTP 服务器
def run(server_class=HTTPServer, handler_class=RequestHandler, port=8080):
    server_address = ('', port)
    httpd = server_class(server_address, handler_class)
    print(f"Starting HTTP server on port {port}")
    httpd.serve_forever()

if __name__ == "__main__":
    run()
```

运行步骤

1. **启动 HTTP 服务器**：运行上面的代码，它会启动一个 HTTP 服务器，监听端口 `8080`。
2. **HTTP 请求转发到 WebSocket**：当 HTTP 服务器接收到 POST 请求时，会将请求数据转发到预先定义的 WebSocket 服务器。
3. **WebSocket 服务器处理**：你需要确保 WebSocket 服务器正在运行，并能够处理来自客户端的消息。

代码说明

- **WebSocketClient 类**: 使用 `websocket-client` 库创建 WebSocket 客户端，并连接到 WebSocket 服务器。
- **RequestHandler 类**: 继承自 `BaseHTTPRequestHandler`，用于处理 HTTP 请求。在 `do_POST` 方法中，接收 POST 请求内容并将其发送到 WebSocket 服务器。
- **run 函数**: 启动 HTTP 服务器并监听指定端口。

扩展

如果你需要实现更复杂的逻辑，例如将 WebSocket 的响应返回给 HTTP 客户端，或者支持更多类型的 HTTP 请求 (如 GET)，可以在 `RequestHandler` 类中添加相应的逻辑。

这个实现是一个简单的示例，可以根据实际需要进行调整和扩展。