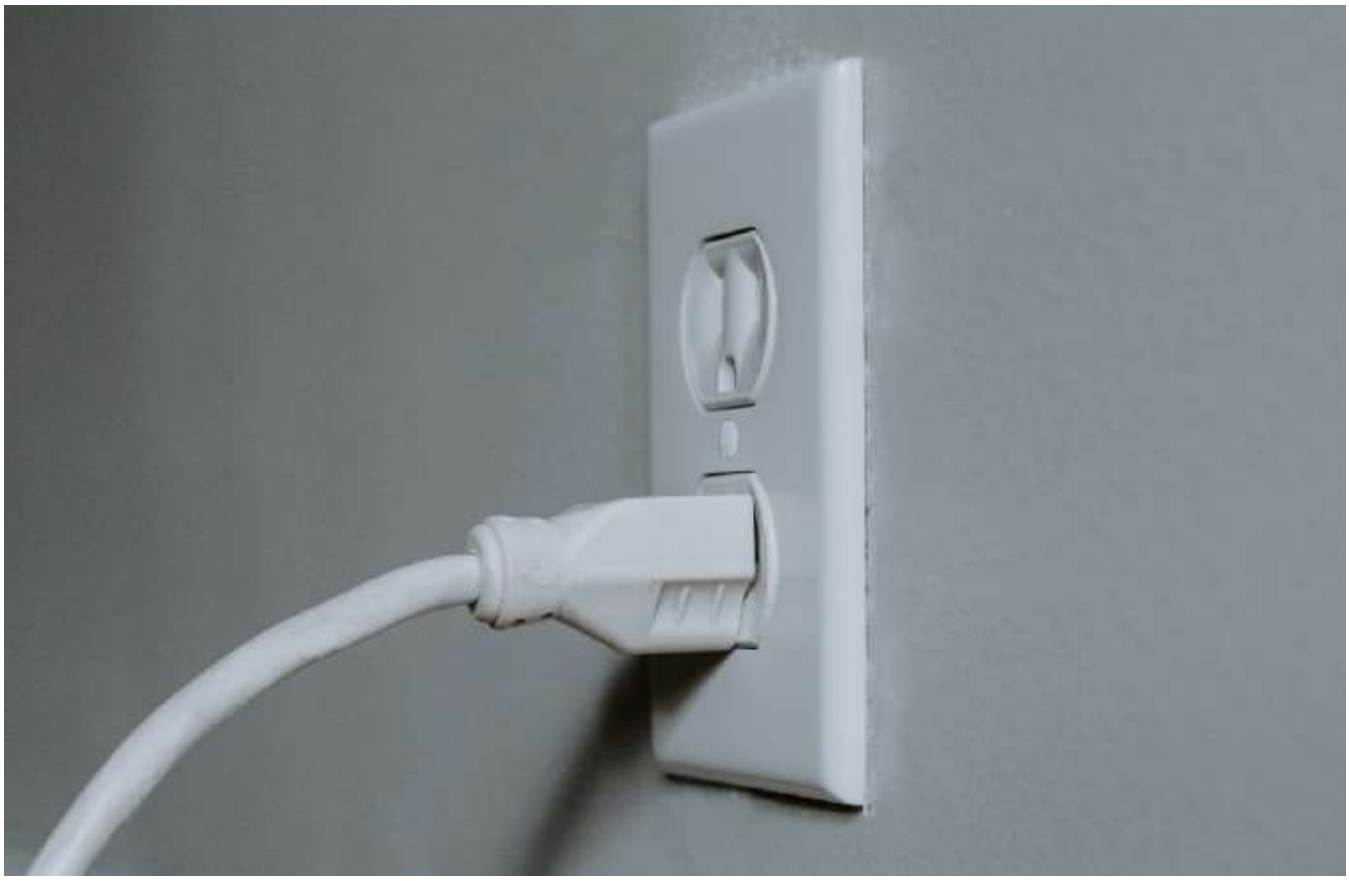
# Working with Sockets in Dart

Suragch
Jan 16 · 3 min read ★

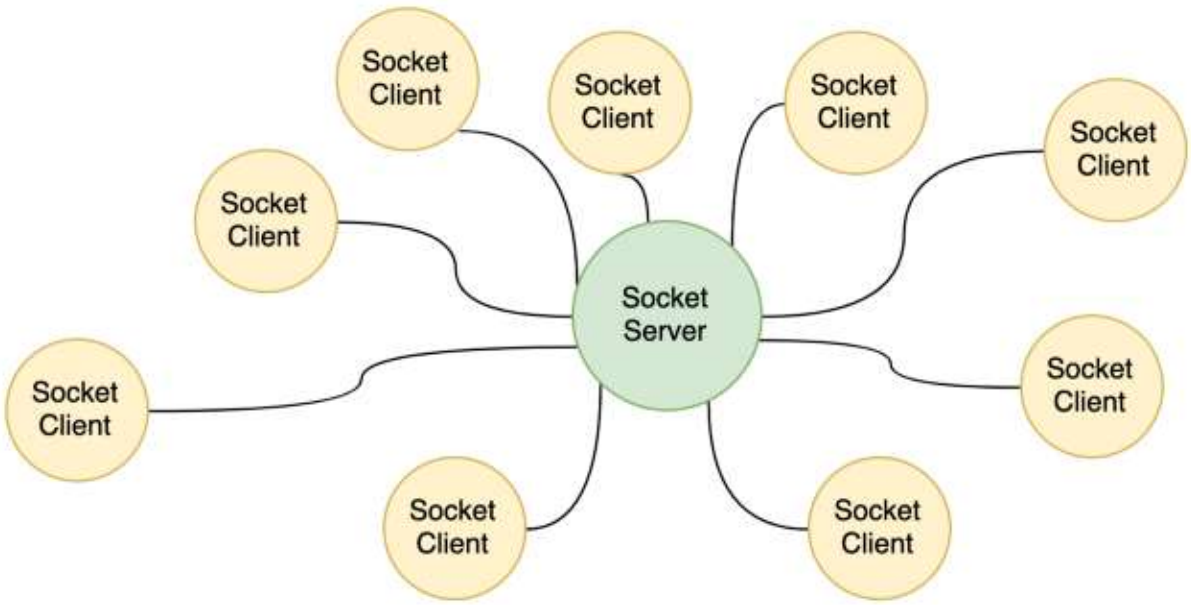


Ok, not that kind of socket.

Sockets are a means of two-way communication over a network. This is similar to HTTP communication, but instead of making a single request and getting a single response, you leave the channel open for further communication. This has applications in chat apps, database communication, and other real-time messaging applications.

A socket server can listen for and receive connections from multiple socket clients. The server can then broadcast messages to all of the connected clients or take a message

from one client and forward it on to another.




In Dart you create a socket client using the `Socket` class, and a socket server using the `SocketServer` class. The next two sections will show you exactly how to do that. Rather than having the server broadcast or relay messages to lots of clients, though, you'll just create a single client and server.

## Socket server

Let's make a knock knock joke server. It'll answer knock knock jokes from clients who connect.

Create a new Dart project called **socket_server**:

```
dart create socket_server
```

Then replace **bin/socket_server.dart** with the following code:

```
1  import 'dart:io';
2  import 'dart:typed_data';
3
4
5  void main() async {
6    // bind the socket server to an address and port
```

```dart
  final server = await ServerSocket.bind(InternetAddress.anyIPv4, 4567);

  // listen for clent connections to the server
  server.listen((client) {
    handleConnection(client);
  });
}

void handleConnection(Socket client) {
  print('Connection from'
      ' ${client.remoteAddress.address}:${client.remotePort}');

  // listen for events from the client
  client.listen(

    // handle data from the client
    (Uint8List data) async {
      await Future.delayed(Duration(seconds: 1));
      final message = String.fromCharCodes(data);
      if (message == 'Knock, knock.') {
        client.write('Who is there?');
      } else if (message.length < 10) {
        client.write('$message who?');
      } else {
        client.write('Very funny.');
        client.close();
      }
    },

    // handle errors
    onError: (error) {
      print(error);
      client.close();
    },

    // handle the client closing the connection
    onDone: () {
      print('Client left');
      client.close();
    },
  );
}
```

socket_server.dart hosted with ♡ by GitHub view raw

You start by binding the `SocketServer` to an IP address and port. After that you listen for connections from a client. When there is a connection, you get a reference to the

client `Socket`, which you can use to listen for data coming from that client. The data is of type `Uint8List` (see Working with bytes in Dart if you don't know what that is), so you need to convert it back to a string before you can work with it. Finally, you send messages back to the client by calling `client.write`.

## Socket client

The server is done, so let's make a client to connect to the server and tell it a funny knock knock joke.

Create a new Dart project and call it **socket_client**:

```
dart create socket_client
```

Then replace **bin/socket_client.dart** with the following code:

```dart
import 'dart:io';
import 'dart:typed_data';

void main() async {

  // connect to the socket server
  final socket = await Socket.connect('localhost', 4567);
  print('Connected to: ${socket.remoteAddress.address}:${socket.remotePort}');

  // listen for responses from the server
  socket.listen(

    // handle data from the server
    (Uint8List data) {
      final serverResponse = String.fromCharCodes(data);
      print('Server: $serverResponse');
    },

    // handle errors
    onError: (error) {
      print(error);
      socket.destroy();
    },

    // handle server ending connection
    onDone: () {
      print('Server left.');
```

```
      socket.destroy();
    },
  );

  // send some messages to the server
  await sendMessage(socket, 'Knock, knock.');
  await sendMessage(socket, 'Banana');
  await sendMessage(socket, 'Banana');
  await sendMessage(socket, 'Banana');
  await sendMessage(socket, 'Banana');
  await sendMessage(socket, 'Banana');
  await sendMessage(socket, 'Orange');
  await sendMessage(socket, "Orange you glad I didn't say banana again?");
}

Future<void> sendMessage(Socket socket, String message) async {
  print('Client: $message');
  socket.write(message);
  await Future.delayed(Duration(seconds: 2));
}
```

socket_client.dart hosted with ♡ by GitHub view raw

The socket client initiates a connection with the socket server at the address and port where the server is listening. After that the client listens for data coming in from the server. As before, this data is in bytes, so if you want a string, you have to convert it to one.

Besides listening for data coming in from the server, you can also send data to the server by calling `socket.write`.

## Testing it out

Start up the server. You can either run it on the command line from the **socket_server** project root like this:

```
dart bin/socket_server.dart
```

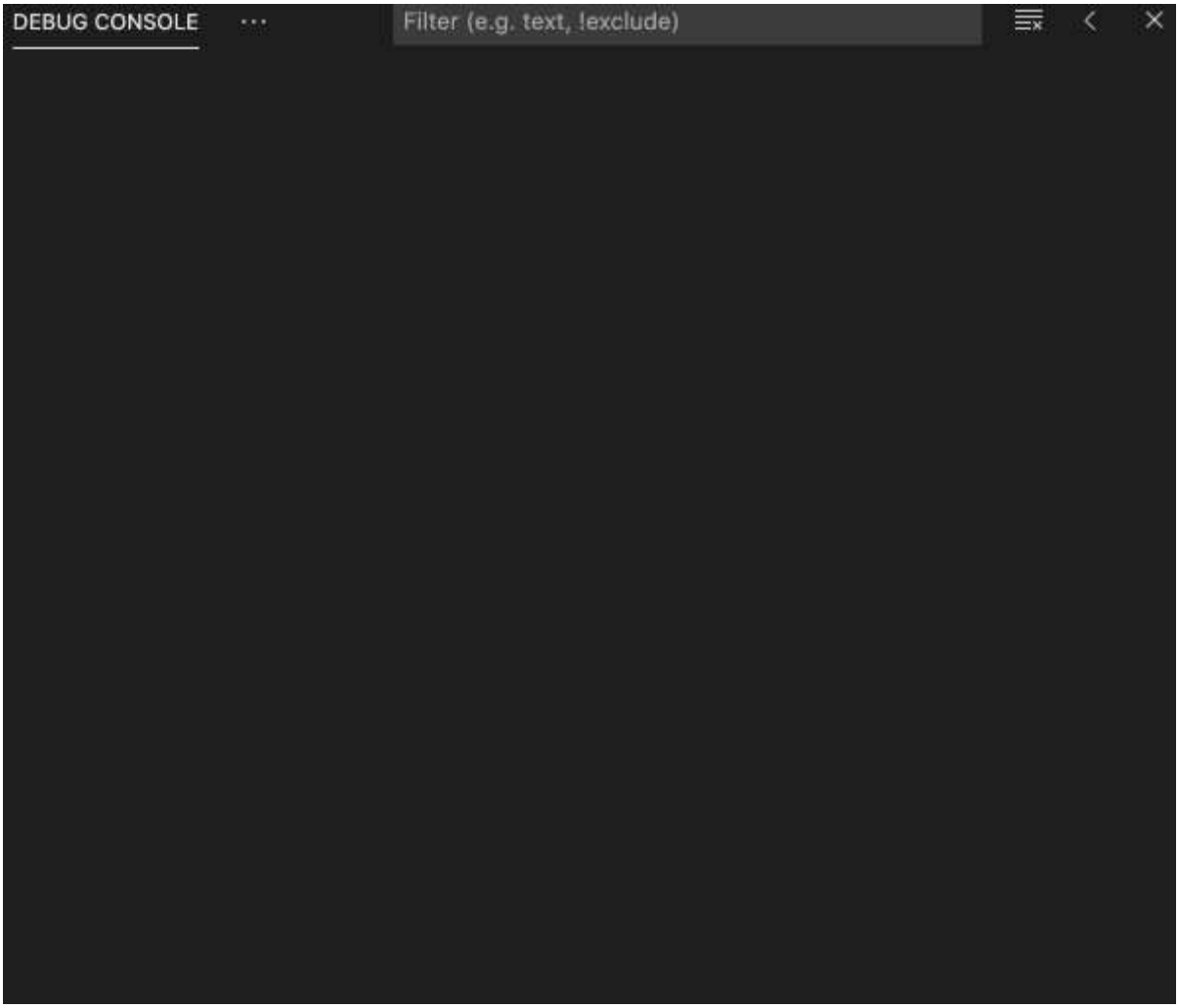
Or you can run it directly in VS Code (or whatever IDE you're using).

With the server running, start the socket client. If you want to run it from the command line, go to the **socket_client** project root and type the following command in a terminal:

```
dart bin/socket_client.dart
```

Or you can just open a new window in VS Code and run it from there. That's what I did in the image below.

With both programs running, you should see the following result:




## Other notes

- The socket is just a stream. If you want to send bytes rather than string data, you can call `socket.add` instead of `socket.write`.
- The `Future.delayed` was in there to make the content appear in a readable manner. Firing lots of message at the server at once also seems to put it all in the same data chunk in the stream.
- I learned most of this from Network Socket Programming with Dart 1.0. You should definitely read that. Although old, the code still mostly works. It also has a

lot more details than I've written here, including how to make a chat room server. The same author also wrote an article called WebSocket programming with Dart 1.1.

Sockets are very interesting and they have a lot of potential. I plan to use them more in the future, so you may see them in upcoming articles. Follow me on Twitter for updates.

Flutter Dart Programming Socket Programming Servers