@push.rocks/isotransport

a bi-directional, multiplatform, best-effort transport

readme.md for @push.rocks/isotransport

a bi-directional, multiplatform, best-effort transport

Install

To install `@push.rocks/isotransport`, use the following command in your project directory:

```
npm install @push.rocks/isotransport --save
```

This will fetch and install the `isotransport` package and add it as a dependency to your project's `package.json` file.

Usage

The `@push.rocks/isotransport` module is designed as a versatile transport layer that abstracts away the complexity of bi-directional communication across different platforms. Its goal is to provide a "best effort" transport, meaning it aims to optimize communication reliability and efficiency without guaranteeing message delivery under all circumstances.

In the following sections, various aspects of using `@push.rocks/isotransport` are covered, including setting up a simple server-client connection, handling messages, and configuring the transport layer for different environments.

Basic Setup

First, ensure that you are using TypeScript and have it configured in your project. `@push.rocks/isotransport` is developed with TypeScript in mind, offering type definitions out of the box for enhanced development experience.

Importing

Start by importing the necessary components from `@push.rocks/isotransport` in your TypeScript file:

```
import { IsotransportServer, IsotransportClient } from '@push.rocks/isotransport';
```

Setting Up a Server

To set up a server instance that listens for incoming connections, use the `IsotransportServer` class:

```
const transportServer = new IsotransportServer({
  port: 8080, // Specify the port on which the server should listen
});

// Start listening for connections
transportServer.listen().then(() => {
  console.log('Server is listening for incoming connections...');
});

// Handling client connections
transportServer.on('connection', (client) => {
  console.log('Client connected:', client.id);

  client.on('message', (message) => {
    console.log('Message from client:', message);
  });

  client.send('Welcome to isotransport server!');
});
```

Setting Up a Client

Setting up a client that connects to an isotransport server is straightforward with the `IsotransportClient` class:

```
const transportClient = new IsotransportClient({
  url: 'ws://localhost:8080', // URL of the isotransport server
});
```

```
// Connecting to the server
transportClient.connect().then(() => {
  console.log('Connected to the server.');
});

// Sending a message to the server
transportClient.send('Hello, server!');

// Receiving messages from the server
transportClient.on('message', (message) => {
  console.log('Message from server:', message);
});
```

Multiplatform Communication

`@push.rocks/isotransport` shines in scenarios where communication needs to happen across different platforms, for example, between a Node.js server and a web client. The design of isotransport abstracts the underlying transport mechanisms (like WebSockets for web clients and TCP sockets for Node.js), offering a unified API for sending and receiving messages.

Advanced Configuration

Isotransport provides hooks and configuration options for tweaking its behavior to fit specific use cases. For instance, you can configure retry strategies for message delivery, set custom serializers for message encoding/decoding, or integrate with custom logging solutions to monitor communication flows.

Due to the scope of this guide, these advanced topics are not covered in detail here. However, they are well-documented in the isotransport API documentation, offering comprehensive insights into enhancing the capabilities of your transport layer.

Conclusion

`@push.rocks/isotransport` is a powerful tool for creating reliable, efficient, and scalable communication layers in your application. By abstracting the complexities of bi-directional multiplatform communication, it allows developers to focus on building the core features of their applications. Whether you're developing a real-time chat application, a distributed microservices architecture, or any system that requires robust communication, isotransport provides the foundational elements needed to bring your project to life.

License and Legal Information

This repository contains open-source code that is licensed under the MIT License. A copy of the MIT License can be found in the license file within this repository.

Please note: The MIT License does not grant permission to use the trade names, trademarks, service marks, or product names of the project, except as required for reasonable and customary use in describing the origin of the work and reproducing the content of the NOTICE file.

Trademarks

This project is owned and maintained by Task Venture Capital GmbH. The names and logos associated with Task Venture Capital GmbH and any related products or services are trademarks of Task Venture Capital GmbH and are not included within the scope of the MIT license granted herein. Use of these trademarks must comply with Task Venture Capital GmbH's Trademark Guidelines, and any usage must be approved in writing by Task Venture Capital GmbH.

Company Information

Task Venture Capital GmbH
Registered at District court Bremen HRB 35230 HB, Germany

For any legal inquiries or if you require further information, please contact us via email at hello@task.vc.

By using this repository, you acknowledge that you have read this section, agree to comply with its terms, and understand that the licensing of the code does not imply endorsement by Task Venture Capital GmbH of any derivative works.