@push.rocks/smartimap

stream messages over imap

- readme.md for @push.rocks/smartimap
- changelog.md for @push.rocks/smartimap

readme.md for @push.rocks/smartimap

A Node.js library for event-driven streaming and parsing of IMAP email messages.

Install

To install `@push.rocks/smartimap`, use npm or yarn:

```
npm install @push.rocks/smartimap
```

or

```
yarn add @push.rocks/smartimap
```

Usage

`@push.rocks/smartimap` is designed to facilitate the connection to an IMAP server, filter email messages and handle them in a streaming, event-driven manner. Below is a comprehensive guide that covers setting up the environment, features of the module, and usage scenarios.

Setting Up Your Environment

Before starting with `@push.rocks/smartimap`, ensure you have Node.js installed. You may also need environment variables for the IMAP server settings, which can be conveniently managed using a `.env` file.

Importing the Required Modules

Begin by importing the `SmartImap` class and any other necessary modules:

```
import { SmartImap, SmartImapConfig } from '@push.rocks/smartimap';
```

Configuration Object

Define a configuration object `SmartImapConfig` for the IMAP connection. This configuration object will include details such as the host, port, and authentication details:

```
const imapConfig: SmartImapConfig = {
  host: 'imap.example.com',
  port: 993, // Secure port for IMAP
  secure: true, // Connect securely using SSL/TLS
  auth: {
    user: 'user@example.com',
    pass: 'password123',
  },
  mailbox: 'INBOX', // Default mailbox to access
  filter: { seen: false }, // Default filter for unseen messages
};
```

Creating an Instance

Create an instance of the `SmartImap` class using the configuration object:

```
const smartImap = new SmartImap(imapConfig);
```

Connecting to the IMAP Server

Use the `connect` method to establish a connection with the IMAP server:

```
smartImap.connect().catch(console.error);
```

Handling Events

`@push.rocks/smartimap` is event-driven and emits various events such as `message`, `error`, `connected`, and `disconnected`. These events can be handled to perform appropriate actions.

Handling the `message` Event

The `message` event is emitted when a new message is fetched and parsed. Here's an example of how to handle this event:

```
smartImap.on('message', (parsedMessage) => {
  console.log('New message received:');
  console.log('From:', parsedMessage.from?.text);
  console.log('Subject:', parsedMessage.subject);
  console.log('Text:', parsedMessage.text);
});
```

Handling the `error` Event

The `error` event is emitted when an error occurs:

```
smartImap.on('error', (error) => {
  console.error('Error occurred:', error);
});
```

Handling the `connected` Event

The `connected` event is emitted when the connection to the IMAP server is successfully established:

```
smartImap.on('connected', () => {
  console.log('Successfully connected to the IMAP server');
});
```

Handling the `disconnected` Event

The `disconnected` event is emitted when the connection to the IMAP server is closed:

```
smartImap.on('disconnected', () => {
  console.log('Disconnected from the IMAP server');
});
```

Fetching and Parsing Messages

Upon connecting to the IMAP server, the `SmartImap` instance will automatically start fetching and parsing messages based on the provided filter criteria. The parsed messages are then emitted through the `message` event. Below is a detailed example:

```
smartImap.on('message', (parsedMessage) => {
  console.log('From:', parsedMessage.from?.text);
  console.log('Subject:', parsedMessage.subject);
  console.log('Text:', parsedMessage.text);
});
```

The `parsedMessage` object contains various properties such as `from`, `subject`, `text`, etc., derived using the `mailparser` module internally.

Modifying the Message Filter

The message filter criteria can be dynamically altered using the `setFilter` method:

```
smartImap.setFilter({ seen: false, from: 'specific_sender@example.com' });
```

Disconnecting from the Server

To safely disconnect from the IMAP server, use the `disconnect` method:

```
smartImap.disconnect().catch(console.error);
```

Full Example

Below is a comprehensive example that showcases creating an instance, connecting to the server, listening for messages, and handling errors and other events:

```
import { SmartImap, SmartImapConfig } from '@push.rocks/smartimap';

const imapConfig: SmartImapConfig = {
  host: 'imap.example.com',
  port: 993,
  secure: true,
  auth: {
    user: process.env.IMAP_USER || 'user@example.com',
    pass: process.env.IMAP_PASSWORD || 'password123',
  },
  mailbox: 'INBOX',
  filter: { seen: false },
```

```
};

const smartImap = new SmartImap(imapConfig);

smartImap.connect().catch(console.error);

smartImap.on('message', (parsedMessage) => {
  console.log('----New Message----');
  console.log('From:', parsedMessage.from?.text);
  console.log('Subject:', parsedMessage.subject);
  console.log('Text:', parsedMessage.text);
});

smartImap.on('error', (error) => {
  console.error('Error:', error);
});

smartImap.on('connected', () => {
  console.log('Connected to the IMAP server');
});

smartImap.on('disconnected', () => {
  console.log('Disconnected from the IMAP server');
});

// Changing filter to show only unseen messages from a specific sender
smartImap.setFilter({ seen: false, from: 'specific_sender@example.com' });

// Disconnect from the server safely
setTimeout(() => {
  smartImap.disconnect().catch(console.error);
}, 30000); // Timeout after 30 seconds
```

Advanced Usage

In this section, we delve into more advanced scenarios and functionalities.

Searching for Specific Messages

You may sometimes need to fetch messages based on more specific search criteria. `@push.rocks/smartimap` lets you customize the search queries by modifying the filter. Below is an example:

```
import { SmartImap, SmartImapConfig } from '@push.rocks/smartimap';

const imapConfig: SmartImapConfig = {
  host: 'imap.example.com',
  port: 993,
  secure: true,
  auth: {
    user: process.env.IMAP_USER || 'user@example.com',
    pass: process.env.IMAP_PASSWORD || 'password123',
  },
  mailbox: 'INBOX',
  filter: { seen: true },
};

const smartImap = new SmartImap(imapConfig);

smartImap.connect().catch(console.error);

smartImap.on('message', (parsedMessage) => {
  console.log('Filtered Message:', parsedMessage.subject);
});

smartImap.on('connected', async () => {
  console.log('Connected');

  // Adjust filter as needed
  await smartImap.setFilter({ subject: 'Invoice' });

  // Manually fetching new messages (maybe based on different filters)
  await smartImap['fetchNewMessages']();
});

smartImap.on('error', (error) => {
  console.error('Error:', error);
});
```

Streamlined Processing with Event Emitters

For more complex workflows, you can use event emitters to segregate processing tasks. Below is an example of a MessageProcessor class using event emitters:

```
import { SmartImap, SmartImapConfig } from '@push.rocks/smartimap';
import EventEmitter from 'events';

class MessageProcessor extends EventEmitter {
  constructor(private imapConfig: SmartImapConfig) {
    super();
    const smartImap = new SmartImap(imapConfig);

    smartImap.on('message', (parsedMessage) => {
      this.emit('processMessage', parsedMessage);
    });

    smartImap.on('error', (error) => {
      console.error('IMAP Error:', error);
    });

    smartImap.connect().catch(console.error);
  }

  startProcessing() {
    this.on('processMessage', this.processMessage.bind(this));
  }

  private processMessage(parsedMessage: any) {
    console.log('Processing message from:', parsedMessage.from?.text);

    // Additional processing logic here...

    this.emit('messageProcessed', parsedMessage);
  }
}

const messageProcessor = new MessageProcessor(imapConfig);
messageProcessor.startProcessing();
```

```
messageProcessor.on('messageProcessed', (message) => {
  console.log('Message processed:', message.subject);
});
```

Additional Features and Usage

Handling Attachments

The `mailparser` library used by `@push.rocks/smartimap` allows you to process email attachments. Here's how you can handle attachments:

```
smartImap.on('message', (parsedMessage) => {
  if (parsedMessage.attachments && parsedMessage.attachments.length > 0) {
    parsedMessage.attachments.forEach((attachment) => {
      console.log('Attachment:', attachment.filename);
      // Process the attachment as needed
    });
  }
});
```

Advanced Filtering

You can utilize complex filter criteria as allowed by the IMAP protocol. For example, filtering by date range:

```
const filter = [
  ['SINCE', new Date('2023-10-01')],
  ['BEFORE', new Date('2023-10-10')],
];

smartImap.setFilter({ ...filter });
```

Error Handling Strategies

Proper error handling is crucial for robust applications. Here's an approach to comprehensive error handling:

```
smartImap.on('error', (error) => {
  if (error.message.includes('Connection closed')) {
    // Attempt to reconnect
```

```
    smartImap.connect().catch(console.error);
  } else {
    console.error('Unexpected error:', error);
  }
});
```

Full Documentation and API Reference

The above examples demonstrate how to use `@push.rocks/smartimap` comprehensively. For detailed API documentation, refer to the project's documentation generated with tools like TypeDoc, ensuring you cover all methods and properties available.

Conclusion

With `@push.rocks/smartimap`, handling IMAP emails in a Node.js environment becomes efficient and straightforward. Utilize the detailed examples and advanced features to build tailored solutions for your email processing needs.

Enjoy coding with `@push.rocks/smartimap`!

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.

changelog.md for @push.rocks/smartimap

2024-11-26 - 1.2.0 - feat(core)

Introduce ImapClient and ImapServer classes for enhanced IMAP support

- Implemented ImapClient class for managing IMAP connections and message retrieval.
- Implemented ImapServer class to simulate an IMAP server for testing.
- Added new tests for ImapClient and ImapServer to ensure reliability.
- Updated dependencies in package.json to latest versions.

2024-09-19 - 1.1.0 - feat(core)

Enhance package with detailed documentation and updated npm metadata

- Comprehensive documentation added to README.md for installation, usage, and advanced features
- Updated package.json and npmextra.json descriptions and keywords
- Added detailed examples for handling IMAP emails, events, and attachments in README.md
- Improved error handling and message processing logic in SmartImap class

2024-09-19 - 1.0.1 - Initial Release

First release of the project
  - Initial commit and setup