readme.md for @push.rocks/smartcls

continuation-local-storage using native AsyncLocalStorage

Install

To install `@push.rocks/smartcls`, use the following command with npm:

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

or if you prefer using Yarn:

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

Usage

`@push.rocks/smartcls` simplifies the usage of native AsyncLocalStorage in Node.js, allowing for a more convenient approach to work with continuation-local storage. This can be especially handy in scenarios where context passing through asynchronous calls is essential, such as in web servers or complex application flows that involve async operations.

Basic Setup and Running Context

To start using `@push.rocks/smartcls`, you first need to import it and create an instance of `SmartCls`.

```
import { SmartCls } from '@push.rocks/smartcls';


const mySmartCls = new SmartCls();
```

Once you have an instance, you can utilize the `run` method to establish a new context. Within this context, you can set and get values that are scoped to the lifetime of the context.

```
mySmartCls.run(() => {
  mySmartCls.set('key', 'value');

  // later in the async flow
  console.log(mySmartCls.get('key')); // Outputs: 'value'
});
```

Asynchronous Example

The power of `SmartCls` becomes evident when working with asynchronous operations. Values set in a specific context are accessible throughout the async flow initiated within that context.

```
import { SmartCls } from '@push.rocks/smartcls';

const mySmartCls = new SmartCls();

mySmartCls.run(async () => {
  mySmartCls.set('asyncKey', 'asyncValue');

  await someAsyncFunction();
  console.log(mySmartCls.get('asyncKey')); // Outputs: 'asyncValue', even after async
operations
});
```

Ensure all asynchronous operations initiated in the context are awaited or properly handled to maintain the context's integrity.

Nested Contexts

`@push.rocks/smartcls` supports nested contexts, allowing you to create sub-contexts within a main context. This can be useful for overriding values or isolating sections of your async flow.

```
import { SmartCls } from '@push.rocks/smartcls';

const mySmartCls = new SmartCls();

mySmartCls.run(() => {
  mySmartCls.set('level', 'top');
```

```
  mySmartCls.run(() => {
    mySmartCls.set('level', 'nested');
    console.log(mySmartCls.get('level')); // Outputs: 'nested'
  });

  console.log(mySmartCls.get('level')); // Outputs: 'top', reverting back to the parent
context
});
```

Working with Express or Similar Frameworks

In a web server scenario using Express or a similar framework, you can integrate `SmartCls` to track request-specific data across asynchronous operations without explicitly passing the request object.

```
import express from 'express';
import { SmartCls } from '@push.rocks/smartcls';

const app = express();
const mySmartCls = new SmartCls();

app.use((req, res, next) => {
  mySmartCls.run(() => {
    mySmartCls.set('requestId', req.header('X-Request-Id') || Math.random().toString());
    next();
  });
});

app.get('/', async (req, res) => {
  // Simulate an async operation
  await new Promise(resolve => setTimeout(resolve, 100));

  // Access the requestId set at the beginning of the request
  res.send(`Request ID: ${mySmartCls.get('requestId')}`);
});

app.listen(3000, () => {
  console.log('Server running on port 3000');
```

```
});
```

This example demonstrates how to maintain a unique `requestId` for logging or tracking purposes across asynchronous operations within a single request-response cycle.

Important Considerations

- Ensure that all async operations in a context are awaited to prevent context leakage.
- Be mindful of memory usage when storing large objects in the context, as each context retains its values until completion.
- `SmartCls` leverages Node.js's native `AsyncLocalStorage`, so it's dependent on the Node.js version supporting this feature.

Conclusion

`@push.rocks/smartcls` offers a straightforward and efficient approach to managing continuation-local storage in Node.js applications, simplifying context management across asynchronous operations.

License and Legal Information

Trademarks

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.

Revision #3
Created 2026-03-28 11:10:29 UTC by foss.global Team
Updated 2026-03-28 12:17:15 UTC by foss.global Team