@serve.zone/corerender

a rendering service for serve.zone that preserves styles for web components

- readme.md for @serve.zone/corerender
- changelog.md for @serve.zone/corerender

readme.md for @serve.zone/corerender

A rendering service for serve.zone that preserves styles for web components.

Install

To install Corerender in your project, you can use npm. Make sure you have Node.js installed and then run the following command in your terminal:

```
npm install corerender
```

This will add `corerender` as a dependency to your project, allowing you to use its rendering services to preserve styles for web components efficiently.

Usage

Welcome to the comprehensive usage guide for `corerender`, a powerful rendering service designed to integrate seamlessly within your web applications, ensuring that styles for web components are preserved properly. The guide is structured to provide a thorough understanding of `corerender`'s capabilities, demonstrating its flexibility and efficiency through realistic scenarios.

Setting Up Your Environment

First things first, let’s get `corerender` up and running in your project. Ensure you've installed the package as detailed in the Install section. Since `corerender` is a TypeScript-friendly library, it is recommended to use TypeScript for development to leverage the full power of type safety and IntelliSense.

Basic Render Service Setup

```
import { Rendertron } from 'corerender';

const rendertronInstance = new Rendertron();

(async () => {
  console.log('Starting rendertron...');
  await rendertronInstance.start();
  console.log('Rendertron started successfully!');
})();
```

The code initializes an instance of `Rendertron` and starts the service asynchronously. `Rendertron` is the core class responsible for managing the rendering processes, including task scheduling and storing rendering results persistently in a database.

Understanding the Rendertron Architecture

The architecture of `Rendertron` is designed to support web component rendering through several integral components:

1. **Prerender Manager**: Manages the creation and retrieval of prerender results.
2. **Task Manager**: Handles scheduling tasks for prerendering operations and cleanup routines.
3. **Utility Service Server**: Provides the server interface that accepts/render requests and serves prerendered content efficiently.

Using the Prerender Manager

The `PrerenderManager` is responsible for generating and caching the rendering results of webpages. Here's how you can use the `PrerenderManager` to prerender a webpage:

```
import { PrerenderManager } from 'corerender/dist_ts/rendertron.classes.prerendermanager';

(async () => {
  const prerenderManager = new PrerenderManager();
  await prerenderManager.start();

  const urlToPrerender = 'https://example.com';
```

```
  const prerenderResult = await prerenderManager.getPrerenderResultForUrl(urlToPrerender);
  console.log(`Prerendered content for ${urlToPrerender}:`);
  console.log(prerenderResult);

  await prerenderManager.stop();
})();
```

The above script demonstrates accessing a webpage's prerendered content. It initializes the `PrerenderManager`, specifies a URL, and requests the rendering result, which is stored or retrieved from the database.

Scheduling Prerendering Tasks

The `TaskManager` class allows for efficiently scheduling tasks, such as regular prerendering of local domains and cleanup of outdated render results:

```
import { TaskManager } from 'corerender/dist_ts/rendertron.taskmanager';

const taskManager = new TaskManager(rendertronInstance);
taskManager.start();

// Example: Manual trigger of a specific task
taskManager.triggerTaskByName('prerenderLocalDomains');

taskManager.stop();
```

`TaskManager` works closely with the `Rendertron` service to ensure tasks are executed as per defined schedules (e.g., every 30 minutes or daily). It allows manual triggering for immediate execution outside the schedule.

Managing Render Results

The pre-rendered results are stored using `smartdata`'s `SmartDataDbDoc`. You may need advanced control over whether these are retrieved, created anew, or updated:

```
import { PrerenderResult } from 'corerender/dist_ts/rendertron.classes.prerenderresult';

(async () => {
  const url = 'https://example.com';
  let prerenderResult = await PrerenderResult.getPrerenderResultForUrl(prerenderManager, url);
```

```
  // Check if an updated result is necessary
  if (prerenderResultNeedsUpdate(prerenderResult)) {
    prerenderResult = await PrerenderResult.createPrerenderResultForUrl(prerenderManager,
url);
  }

  console.log(`Final Prerendered content for ${url}:`, prerenderResult.renderResultString);
})();
```

Integrating with External Systems

`Corerender` can be integrated into broader systems that programmatically manage URLs and rendering frequencies. For instance, parsing and prerendering sitemaps:

```
class IntegrationExample {
  private prerenderManager: PrerenderManager;

  constructor() {
    this.prerenderManager = new PrerenderManager();
  }

  async prerenderFromSitemap(sitemapUrl: string) {
    await this.prerenderManager.prerenderSitemap(sitemapUrl);
    console.log('Finished prerendering sitemap:', sitemapUrl);
  }
}

(async () => {
  const integrationExample = new IntegrationExample();
  await integrationExample.prerenderFromSitemap('https://example.com/sitemap.xml');
})();
```

Server-Side Rendering Directly with SmartSSR

`Rendertron` uses the highly efficient `smartssr` for SSR requests. You can easily direct incoming server requests to utilize this rendering pipeline:

```
import { typedserver } from 'corerender/dist_ts/rendertron.plugins';

const serviceServerInstance = new typedserver.utilityservers.UtilityServiceServer({
  serviceDomain: 'rendertron.example.com',
  serviceName: 'RendertronService',
  serviceVersion: '2.0.61', // Replace with dynamic version retrieval if needed
  addCustomRoutes: async (serverArg) => {
    serverArg.addRoute(
      '/render/*',
      new typedserver.servertools.Handler('GET', async (req, res) => {
        const requestedUrl = req.url.replace('/render/', '');
        const prerenderedContent = await
prerenderManager.getPrerenderResultForUrl(requestedUrl);
        res.write(prerenderedContent);
        res.end();
      })
    );
  },
});

(async () => {
  await serviceServerInstance.start();
  console.log('SSR Server Started');
})();
```

Customizing the Logger

`Rendertron` employs the `smartlog` package for logging activities across the service. To customize logging, instantiate a logger with custom configurations:

```
import { smartlog } from 'corerender/dist_ts/rendertron.plugins';

const customLogger = smartlog.Smartlog.create({ /* custom options */ });
customLogger.log('info', 'Custom logger integrated successfully.');
```

Closing Remarks

With these examples, you should have a robust understanding of how to implement `corerender` in your web application. It’s a powerful service that takes care of rendering optimizations, allowing developers to focus on building components and architecture, with clear workflows to handle tasks and results efficiently.

undefined

changelog.md for @serve.zone/corerender

2025-01-01 - 2.0.62 - fix(metadata)

Updated package and npmextra.json metadata fields for clarity

- Updated description and added keywords in package.json and npmextra.json
- Clarified project metadata to streamline categorization and searchability

Since the commit "initial" lacks enough detail or relevant changes, it can be omitted from the changelog. The version 2.0.61 does not have any significant entries to document.

2025-01-01 - 2.0.61 - No significant changes

Version 2.0.61 did not include any notable updates or changes worth documenting.