Appendix: Refactoring the `appendCharacterCard` Function

Background

In the initial implementation, loading all the characters might have taken significant time, leading to a less responsive interface.

In this Appendix you will refactor the `appendCharacterCard` function has to improve the performance and user experience when loading character data. This refactoring incorporates placeholders and utilizes the Intersection Observer API, which allows the characters to be loaded asynchronously as they become visible within the viewport. This technique is known as lazy loading and is widely used in modern web development to optimize performance.

By implementing this pattern, the exercise now not only will challenge your understanding of asynchronous programming and DOM manipulation but also introduces an essential real-world technique for optimizing web performance.

Further Reading

- Learn Intersection Observer In 15 Minutes: Web Dev Simplified YouTube tutorial Video About Intersection Observer.
- JavaScript Intersection Observer Ultimate Guide: An in-depth guide to the Intersection Observer API on the Web Dev Simplified Blog.
- Lazy Loading Images: A comprehensive article on using the Intersection Observer for lazy loading images, but the principles can be applied to any DOM elements.

Detailed Instruction for implementing the refactored Function `appendCharacterCard`

1: Declare Asynchronous Function

Same as the previous implementation.

2: Begin Try Block

Same as the previous implementation.

3: Hide Character Card Container and Show Spinner

Same as the previous implementation.

4: Clear Previous Character Cards

Same as the previous implementation.

5: Fetch Movie Data

Same as the previous implementation.

6: Create Characters Container and Title

Same as the previous implementation.

7: Hide Spinner

Set the display property of the loading spinner to `none` to indicate the completion of loading movie data.

8: Create Placeholders

Iterate through the character URLs, creating a `<div>` placeholder for each character. Store it in a `placeholder` constant. Set a class name of `character-placeholder` to this `placeholder` `div`. Set a data attribute with an `index` key and store in it the index of each iteration. Append each placeholder to the characters container.

9: Initialize Intersection Observer

Create a new Intersection Observer by declaring a constant `observer`. Inside the observer's callback function, the code will process multiple **entries** that represent the elements being observed (in this case, the character placeholders).

9.1: Iterate Through Entries

Iterate through each entry in the `entries` array. An entry represents the state of a target element and its level of intersection with the root container (viewport by default).

9.2: Check if Entry is Intersecting

Inside the iteration, use an `if` statement to check if the entry is intersecting with the viewport. If it is intersecting, the following steps will be executed for that particular entry.

9.3: Retrieve Placeholder and Index

Retrieve the placeholder element from `entry.target` and the index from the `dataset.index` property of the placeholder. The index will be used to get the corresponding URL from the `characterUrls` array.

9.4: Fetch Character Details

Call the previously defined `getCharacterDetails` function asynchronously with the appropriate character URL. Await the resolved data and assign it to a constant `characterDetails`.

9.5: Create Character Card

Use the fetched `characterDetails` to create a character card by calling the `createCharacterCard` function. Assign the HTML string returned by the function to a constant `characterCard`.

9.6: Replace Placeholder with Character Card

Replace the `outerHTML` of the placeholder with the `characterCard` HTML string. This effectively replaces the placeholder with the actual character card in the DOM.

9.7: Unobserve Placeholder

Call the `observer.unobserve` method on the `placeholder`. This stops the observer from watching the specific placeholder, reducing unnecessary checks once the character data has been loaded.

10: Observe All Placeholders

Use `querySelectorAll` to select all the placeholders and iterate over each placeholder. In each iteration call `observer.observe` on each `placeholder`, starting the observation process.

11: Display Character Card Container

Set the display property of the character card container to `block`, making it visible once all placeholders have been set up.

12: Error Handling

In the catch block, call the `handleError` function with the `charactersSpinner`, `charactersCardContainer`, and the `error` object, to display an error message and hiding the spinner.

Conclusion

IntersectionObserver

In the given function, the `IntersectionObserver` is used to detect when the character placeholders come into view. Here's how it works:

- An instance of `IntersectionObserver` is created with a callback function that will be invoked whenever one of the observed elements intersects with the viewport.
- Inside the callback, the `entries` parameter provides information about the observed elements, including whether they are currently intersecting (`entry.isIntersecting`).
- If a placeholder is intersecting, the corresponding character details are fetched, the character card is created, and the placeholder's `outerHTML` is replaced with the character card's HTML.

observer.observe

The `observe` method is part of the `IntersectionObserver` API and is used to start observing an element for intersections with the viewport or another scrolling container.

In this refactored version of the code, the `observe` method is used to start observing all the placeholders:

- `document.querySelectorAll('.character-placeholder')` is used to select all the placeholders in the document.
- A `forEach` loop is then used to call `observer.observe(placeholder)` on each placeholder, telling the `IntersectionObserver` to start watching those elements.
- Once a placeholder has been replaced with the actual character card, `observer.unobserve(placeholder)` is called to stop observing that specific element, as it no longer needs to be tracked.

Summery

The combination of placeholders, `IntersectionObserver`, and the `observe` method allows for an efficient and performant lazy loading strategy in the given function. By only fetching and rendering content as it becomes visible, this approach can significantly improve the user experience, especially when dealing with large lists of data. It leverages modern browser APIs and provides a native solution for handling asynchronous content loading without relying on third-party libraries.