The `infuseData` Callback

The Challenge

When Replyke fetches data for a list, it retrieves only the information it manages internally. While this is sufficient for many applications, there are cases where entities need additional external data to be meaningful.

For example: A product entity might need inventory details from an external inventory management system.

Without a mechanism to merge external data into Replyke's entities, developers are left with incomplete data objects, limiting the application's functionality and user experience.

The Solution

The `infuseData` callback provides a seamless way to integrate external data into entities fetched by Replyke. This function is passed to the `EntityListProvider` and is invoked for each entity in the list. The `infuseData` function:

1. Receives a `foreignId` for each entity.
2. Fetches additional data for that entity from an external source (e.g., another database or API).
3. Merges the fetched data with the entity, creating a complete data object that includes both Replyke-managed and externally sourced information.

This approach ensures that developers can leverage Replyke's powerful list capabilities while maintaining access to essential external data.

How It Works

Internally, the `useInfusedData` hook handles the logic for merging data. For each entity:

- It checks if the external data for the entity is already cached to avoid redundant fetches.
- If not, it uses the provided `infuseData` function to fetch the required data.
- The fetched data is then merged with the entity, and the complete object is returned in the `infusedEntities` array.

The `entities` array remains "pure," containing only the data fetched by Replyke. The `infusedEntities` array provides the enriched versions of these entities.

Example

Here is a practical example of how to use the `infuseData` callback:

```
import React from 'react';
import { EntityListProvider, useEntityList } from 'replyke';

const fetchInventoryData = async (foreignId: string) => {
  try {
    const response = await fetch(`/api/inventory/${foreignId}`);
    if (!response.ok) {
      throw new Error('Failed to fetch inventory data');
    }
    return await response.json();
  } catch (error) {
    console.error(`Error fetching inventory data for ${foreignId}:`, error);
    return null;
  }
};

const ProductFeed = () => {
  const {
    infusedEntities,
    loadMore,
    resetEntities,
    loading,
    hasMore,
  } = useEntityList();

  return (
    <div>
      <h1>Product Feed</h1>
      <ul>
        {infusedEntities.map((entity) => (
          <li key={entity.foreignId}>
            <h2>{entity.title}</h2>
            <p>Stock: {entity.infusion.stock}</p>
            <p>Price: ${entity.infusion.price}</p>
          </li>
        ))}
      </ul>
      {loading && <p>Loading...</p>}
      {hasMore && <button onClick={loadMore}>Load More</button>}
      <button onClick={resetEntities}>Refresh Feed</button>
    </div>
  );
};
```

```
const App = () => (
  <EntityListProvider infuseData={fetchInventoryData}>
    <ProductFeed />
  </EntityListProvider>
);

export default App;
```

Key Takeaways

- The `infuseData` callback bridges the gap between Replyke's list data and your application's external data requirements.
- It ensures that entities can be enriched with external data dynamically.
- The process is efficient, leveraging caching to minimize redundant fetches.

This feature empowers developers to create rich, data-complete applications while maintaining the flexibility and power of Replyke entity lists.

Last updated on May 7, 2025