27. 实战：Async、await 错误处理的 3 种方式

1. 核心概念 (Core Concept)

`async/await` 是 ECMAScript 2017 标准引入的异步编程语法糖，它基于 Promise 构建，旨在用更同步化的方式书写异步代码，提高代码的可读性和可维护性。尽管简化了异步流程，但错误处理仍是使用 `async/await` 时必须重点关注的问题。

2. 为什么需要它？ (The "Why")

1. **提高可读性:** 将基于回调或 `.then().catch()` 的链式调用转换为类似同步代码的结构，更符合人类的思维习惯。
2. **简化异步流控制:** `await` 关键字暂停代码执行，直到 Promise 完成并返回结果或报错，使得处理多个依赖顺序的异步操作变得非常直观。
3. **更好的错误处理:** 结合同步错误处理机制（如 `try...catch`）来处理异步错误，避免了回调地狱中错误处理分散的问题。

3. API 与用法 (API & Usage)

`async` 关键字用于定义一个异步函数，该异步函数总是返回一个 Promise。`await` 关键字只能在 `async` 函数内部使用，用于等待 Promise 完成。如果 Promise 成功解决，`await` 返回解决的值；如果 Promise 被 reject，`await` 会抛出一个错误。因此，错误处理的核心在于如何捕获 `await` 可能抛出的这个错误。

以下是处理 `async/await` promise reject 的三种常用方式：

方式一：使用 `try...catch` 块

这是处理 `async/await` 代码中最常见、最推荐的方式，因为它利用了 JavaScript 传统的同步错误处理机制。

```
async function fetchData() {
  try {
    // 假设 someAsyncOperation 返回一个 Promise
    const result = await someAsyncOperation();
    console.log("成功获取数据:", result);
    return result; // 可以返回成功的值
  } catch (error) {
    // 如果 someAsyncOperation 的 Promise 被 reject，错误会被捕获
    console.error("获取数据失败:", error);
    // 可以 re-throw 错误，或返回一个默认值，或进行其他处理
    throw error; // 可选：将错误继续向上抛出
  }
}
```

```
// 调用异步函数
fetchData()
  .then(() => console.log("Fetch operation completed."))
  .catch(err => console.error("Outer catch caught error:", err)); // 如果内部 re-throw 则会被外部捕获
```

说明:

- `try` 块中包含可能会抛出错误的异步操作 (`await`)。
- 如果 `await` 的 Promise 被 reject，执行会立即跳转到 `catch` 块。
- 在 `catch` 块中，可以访问到被 reject 的原因（通常是一个 Error 对象）。

方式二：使用 `.catch()` 方法链在末尾

虽然 `async/await` 使代码看起来像同步，但 `async` 函数本身返回的是一个 Promise。因此，可以在调用 `async` 函数的末尾链式调用 `.catch()` 来处理整个异步函数中未被内部 `try...catch` 捕获的错误。

```
async function potentiallyFailingAsyncOperation() {
  // 这个函数可能会由于 await 的 Promise reject 而抛出错误
  const result = await anotherFailingPromise();
  return result;
}

// 调用 async 函数并在外部使用 .catch()
potentiallyFailingAsyncOperation()
  .then(data => {
    console.log("Operation succeeded:", data);
  })
  .catch(error => {
    // 捕获 potentiallyFailingAsyncOperation 函数中未被内部 try...catch 捕获的 Promise reject 错误
    console.error("Operation failed:", error);
  });
```

说明:

- 这种方式适用于处理整个 `async` 函数执行过程中发生的错误，当函数内部没有使用 `try...catch` 或者需要更高层次的错误处理时。
- 如果 `async` 函数内部使用了 `try...catch` 并且没有 `throw` 错误，那么外部的 `.catch()` 不会触发。

方式三：对单个 `await` Promise 使用 `.catch()`

这种方法不像前两种常见，但对于只关心某个特定异步操作的错误，并且不希望中断后续（不依赖于此操作结果的）代码执行时可能有用。

```
async function processData() {
  // 假设 operationA 不依赖 operationB 的结果
  const resultA = await operationA().catch(error => {
    console.error("Operation A failed, proceeding:", error);
    // 返回一个默认值或 null，以便后续代码可以继续执行
    return null;
  });

  // 即使 operationA 失败并返回 null，下面的代码仍会执行
  if (resultA !== null) {
    console.log("Operation A succeeded with:", resultA);
  }

  const resultB = await operationB(); // operation B 的错误需要单独处理或由外部 .catch 捕获
  console.log("Operation B succeeded with:", resultB);
}

processData().catch(error => {
    console.error("An unhandled error occurred (e.g., in operation B):", error);
});
```

说明:

- 通过在 `await` 的 Promise 后面直接链式调用 `.catch()`，可以局部处理该 Promise 的 reject。
- `.catch()` 返回一个新的 Promise。如果原始 Promise reject，`.catch()` 的回调执行，并且这个新的 Promise 会 resolve（除非回调中再次抛出错误或返回一个 rejected Promise）。
- 这种方式破坏了 `try...catch` 的结构性好处，通常不推荐作为主要错误处理方式，除非有特定的局部错误处理需求。

4. 关键注意事项 (Key Considerations)

1. **Promise 的 reject 会变为 `try...catch` 的 Error：** `await` 关键字会“解包”Promise。如果 Promise resolve，你得到值；如果 Promise reject，`await` 会“抛出”那个 reject 的原因，这使得 `try...catch` 可以捕获它。
2. **未捕获的 reject：** 如果在一个 `async` 函数中使用 `await`，但既没有使用 `try...catch` 包裹，也没有在调用该 `async` 函数的地方使用 `.catch()`，那么 Promise 的 reject 将成为一个未处理的 Promise reject，可能导致 Node.js 环境进程崩溃，或在浏览器中触发 `unhandledrejection` 事件。
3. **错误边界：** `try...catch` 块是错误处理的边界。在 `try` 块中，第一个抛出的错误（或者第一个 `await` 的 Promise reject）将导致执行跳到 `catch` 块，并停止 `try` 块中剩余代码的执行。

4. **错误传递：** 在 `catch` 块中，可以选择处理错误、静默失败、记录日志，或者使用 `throw error;` 将同一个错误或一个新的错误向上抛出，以便外层的 `try...catch` 或 `.catch()` 可以捕获它。

5. 参考资料 (References)

- **MDN Web Docs:** Async function
- **MDN Web Docs:** await
- **JavaScript.info:** Async/await (提供清晰的概念解释和示例)
- **官方规范 (ECMAScript):** ECMAScript 2017 Specification (https://tc39.es/ecma262/#sec-async-function-definitions, https://tc39.es/ecma262/#sec-await-operator) - *请注意，直接阅读规范可能非常技术性，MDN 和其他教程是更好的入门资源。*