22. Promise 基本语法 + 错误捕获

1. 核心概念 (Core Concept)

Promise 是 JavaScript 中用于处理异步操作的一种机制。它代表了一个异步操作的最终完成（或失败）及其结果值。Promise 使得异步代码更容易编写和管理，避免了回调地狱（callback hell）。

2. 为什么需要它？ (The "Why")

- **改善可读性与可维护性:** Promise 结构化了异步流程，将异步操作从回调函数的层层嵌套中解放出来，使代码更接近同步风格，易于理解和维护。
- **更友好的错误处理:** 提供统一集中的错误处理机制，通过 `.catch()` 方法可以在链式调用末端捕获整个链条上的错误。
- **解决异步操作结果的传递问题:** Promise 状态一旦确定（fulfilled 或 rejected），其结果值或拒绝原因会永远保持不变，可以安全地在后续的 `.then()` 或 `.catch()` 中访问。

3. API 与用法 (API & Usage)

Promise 是一个构造函数或类。

构造函数：

```
new Promise(executor)
```

- `executor`：一个函数，接受两个参数 `resolve` 和 `reject`。
  - `resolve`：函数，当异步操作成功时调用（并将结果作为参数传递）。
  - `reject`：函数，当异步操作失败时调用（并将错误作为参数传递）。

实例方法:

- `promise.then(onFulfilled, onRejected)`：
  - 用于处理 Promise 成功（fulfilled）或失败（rejected）后的行为。
  - `onFulfilled`：可选函数，Promise 成功时调用，接收成功结果作为参数。
  - `onRejected`：可选函数，Promise 失败时调用，接收失败原因作为参数。
  - `then` 方法返回一个新的 Promise，这使得链式调用成为可能。
- `promise.catch(onRejected)`：
  - `.catch(onRejected)` 是 `.then(undefined, onRejected)` 的语法糖，专门用于处理 Promise 的拒绝（失败）情况。
  - 强烈建议使用 `.catch()` 来处理错误，因为它更具可读性，并且有助于捕获整个链上的错误。
- `promise.finally(onFinally)`：

- 不论 Promise 是成功还是失败，都会执行 `onFinally` 函数。
- `onFinally`: 函数，不接收任何参数。它的返回值会被忽略。通常用于执行一些清理工作。

经典代码示例 (来自 MDN / 官方文档风格):

示例 1: 基本的 Promise 承诺和解析

```
const myPromise = new Promise((resolve, reject) => {
  // 模拟一个异步操作，比如 setTimeout
  setTimeout(() => {
    const success = true; // 假设操作成功
    if (success) {
      resolve("操作成功完成!"); // 成功时调用 resolve
    } else {
      reject("操作失败!"); // 失败时调用 reject
    }
  }, 1000);
});

myPromise.then((result) => {
  console.log("成功:", result); // 打印 "成功: 操作成功完成!"
}).catch((error) => {
  console.error("失败:", error);
});
```

示例 2: Promise 链式调用和错误捕获

```
function step1() {
  return new Promise((resolve) => {
    setTimeout(() => {
      console.log("步骤 1 完成");
      resolve(1);
    }, 500);
  });
}

function step2(value) {
  return new Promise((resolve) => {
    setTimeout(() => {
      console.log("步骤 2 完成，接收到值:", value);
      resolve(value + 1);
    }, 500);
  });
}

function step3(value) {
  return new Promise((_, reject) => {
```

```
    setTimeout(() => {
      console.log("步骤 3 即将失败，接收到值:", value);
      reject(new Error("步骤 3 失败了!")); // 模拟失败
    }, 500);
  });
}

step1()
  .then(step2)      // 步骤 1 的结果传递给 step2
  .then(step3)      // step2 的结果传递给 step3
  .then(finalResult => {
    console.log("所有步骤成功完成:", finalResult); // 这里的代码不会执行，因为
step3 失败了
  })
  .catch(error => {
    console.error("捕获到错误:", error.message); // 错误会被这里的 catch 捕获
  })
  .finally(() => {
    console.log("异步流程结束"); // 无论成功或失败都会执行
  });
```

4. 关键注意事项 (Key Considerations)

- **Promise States:** Promise 有三种状态：`pending` (进行中), `fulfilled` (已成功), 和 `rejected` (已失败)。状态只能从 `pending` 变为 `fulfilled` 或 `rejected`，并且一旦改变，状态就不可逆转（immutable）。
- **Uncaught Rejection:** 如果一个 Promise 被 reject 了，但没有对应的 `.catch()` 或第二个 `.then()` 参数来处理它，就会触发全局的 `unhandledrejection` 事件，在 Node.js 环境下可能导致进程崩溃。确保每一个 Promise 链的末尾都有一个 `.catch()`。
- **`then` 的返回值:** `.then()` 和 `.catch()` 方法总是返回一个新的 Promise。这使得链式调用成为可能。新的 Promise 的状态和结果取决于 `onFulfilled` 或 `onRejected` 回调函数的返回值。如果回调函数返回一个值，新的 Promise 会以这个值 fulfill；如果返回一个 Promise，新的 Promise 会跟随返回的 Promise 的状态和结果；如果抛出错误，新的 Promise 会被 reject。
- **同步错误处理:** `executor` 函数中如果发生同步错误，也会立即触发 Promise 的 reject，并可以被 `.catch()` 捕获到。

5. 参考资料 (References)

- MDN Web Docs - Promise
- JavaScript Promise 教程 (阮一峰) (尽管是博客，但此系列在中文社区被视为权威且高质量的 Promise 讲解)

- ECMAScript® 2024 Language Specification - Promise Objects (核心规范，技术深度最高)