如何实现基于 Promise 的 API

在上一篇文章中，我们讨论了如何使用返回 promises 的 APIs。在本文中，我们将研究另一方面--如何实现返回 promises 的 APIs。跟使用基于 promise 的 APIs 相比，这是一个不太常见的任务，但它仍然值得了解。

前置条件:	基于计算机知识，对 JavaScript 基础有一个合理的了解，包括事件处理和 promises 基础。
目标:	了解如何实现基于 promise 的 APIs。

通常情况下，当你实现一个基于 promise 的 API 时，你会使用事件、普通回调，或者消息传递模型来包裹一个异步操作。你将会使用一个 `Promise` 对象来合理的处理操作的成功或者失败。

实现 alarm() API

在这个示例中我们将会实现一个基于 promise 的 alarm API，叫做 `alarm()` 。它将以被唤醒人的名字和一个在人被唤醒前以毫秒为单位的延迟作为参数。在延迟之后，本函数将会发送一个包含需要被唤醒人名字的 "Wake up!" 消息。

用 setTimeout() 包裹

我们将会使用 `setTimeout()` 来实现 `alarm()` 函数。 `setTimeout()` 以一个回调函数和一个以毫秒为单位的延迟作为参数。当调用 `setTimeout()` 时，它将启动一个设置为给定延迟的计时器，当时间过期时，它就会调用给定的回调函数。

在下面的例子中，我们使用一个回调函数和一个 1000 毫秒的延迟调用 `setTimeout()`：

HTML Play

```
<button id="set-alarm">Set alarm</button>
<div id="output"></div>
```

JS Play

```
const output = document.querySelector("#output");
const button = document.querySelector("#set-alarm");

function setAlarm() {
  window.setTimeout(() => {
    output.textContent = "Wake up!";
  }, 1000);
}

button.addEventListener("click", setAlarm);
```

Play

Set alarm

Promise() 构造器

我们的 `alarm()` 函数返回一个在定时器过期时才会被兑现的 `Promise`。它将会传递一个 "Wake up!" 消息到 `then()` 处理器中，也会在当调用者提供一个负延迟值时拒绝这个 promise。

这里的关键组件是 `Promise()` 构造器。`Promise()` 构造器使用单个函数作为参数。我们把这个函数称作 *执行器*（executor）。当你创建一个新的 promise 的时候你需要实现这个执行器。

这个执行器本身采用两个参数，这两个参数都是函数，通常被称作 `resolve` 和 `reject`。在你的执行器实现里，你调用原始的异步函数。如果异步函数成功了，就调用 `resolve`，如果失败了，就调用 `reject`。如果执行器函数抛出了一个错误，`reject` 会被自动调用。你可以将任何类型的单个参数传递到 `resolve` 和 `reject` 中。

所以我们可以像下面这样实现 `alarm()`：

JS

```
function alarm(person, delay) {
  return new Promise((resolve, reject) => {
    if (delay < 0) {
      throw new Error("Alarm delay must not be negative");
    }
    window.setTimeout(() => {
      resolve(`Wake up, ${person}!`);
    }, delay);
  });
}
```

此函数创建并且返回一个新的 `Promise`。对于执行器中的 promise，我们：

- 检查 `delay`（延迟）是否为负数，如果是的话就抛出一个错误。
- 调用 `window.setTimeout()`，传递一个回调函数和 `delay`（延迟）。当计时器过期时回调会被调用，在回调函数内，我们调用了 `resolve`，并且传递了 `"Wake up!"` 消息。

使用 alarm() API

这一部分同上一篇文章是相当相似的。我们可以调用 `alarm()`，在返回的 promise 中调用 `then()` 和 `catch()` 来设置 promise 兑现和拒绝状态的处理器。

JS Play

```
const name = document.querySelector("#name");
const delay = document.querySelector("#delay");
const button = document.querySelector("#set-alarm");
const output = document.querySelector("#output");

function alarm(person, delay) {
  return new Promise((resolve, reject) => {
    if (delay < 0) {
      throw new Error("Alarm delay must not be negative");
    }
    window.setTimeout(() => {
      resolve(`Wake up, ${person}!`);
    }, delay);
  });
}

button.addEventListener("click", () => {
```

```
  alarm(name.value, delay.value)
    .then((message) => (output.textContent = message))
    .catch((error) => (output.textContent = `Couldn't set alarm: ${error}`));
});
```

Play

Name: Matilda

Delay: 1000

Set alarm

尝试对 "Name" 和 "Delay" 设置不同的值。尝试为 "Delay" 设置一个负值。

在 alarm() API 上使用 async 和 await

自从 `alarm()` 返回了一个 `Promise`，我们可以对它做任何我们可以对其他任何 promise 做的事情：`Promise.all()`，和 `async` / `await`：

JS

Play

```
const name = document.querySelector("#name");
const delay = document.querySelector("#delay");
const button = document.querySelector("#set-alarm");
const output = document.querySelector("#output");

function alarm(person, delay) {
  return new Promise((resolve, reject) => {
    if (delay < 0) {
      throw new Error("Alarm delay must not be negative");
    }
    window.setTimeout(() => {
      resolve(`Wake up, ${person}!`);
    }, delay);
  });
}

button.addEventListener("click", async () => {
  try {
```

```
    const message = await alarm(name.value, delay.value);
    output.textContent = message;
  } catch (error) {
    output.textContent = `Couldn't set alarm: ${error}`;
  }
});
```

Play

参见

- Promise() 构造器
- 使用 Promises

Help improve MDN

Was this page helpful to you?

Yes No

Learn how to contribute.

This page was last modified on 2023年7月24日 by MDN contributors.