Promise.all() 原理解析及使用指南

作者：devpoint

2021-09-03 • 本文字数：2108 字 • 阅读完需：约 7 分钟

`Promise` 对象是 ECMAScript 6 中新增的对象，主要将 JavaScript 中的异步处理对象和处理规则进行了规范化。前面介绍了《Promise.any() 原理解析及使用指南》，本文来介绍另一个方法 `Promise.all(promises)`，能够一次并行处理多个 `promise`，并且只返回一个 `promise` 实例，那个输入的所有 `promise` 的 `resolve` 回调的结果是一个数组。

下面来看看 `Promise.all()` 是如何工作的。

1.工作原理

`Promise.all()` 是一个内置的辅助函数，接受一组 `promise`（或者一个可迭代的对象），并返回一个`promise`：

复制代码

```
const allPromise = Promise.all([promise1, promise2, ...]);

```

可以使用 `then` 方法提取第一个 `promise` 的值：

复制代码

```
allPromise.then((values) => {
    values; // [valueOfPromise1, valueOfPromise2, ...]
});

```

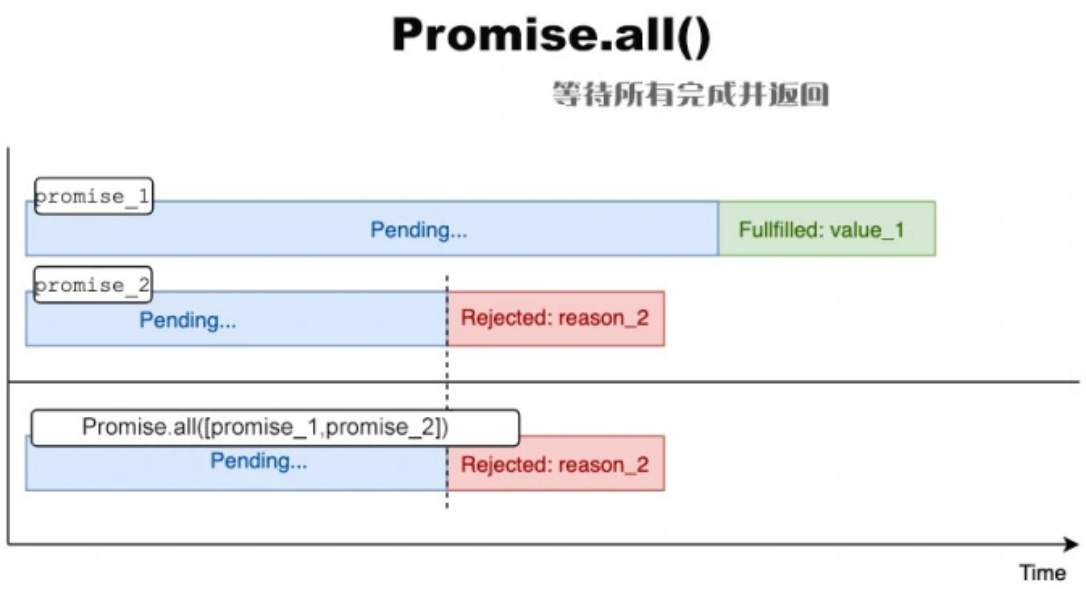
也可以使用 `async/await` 语法：

复制代码

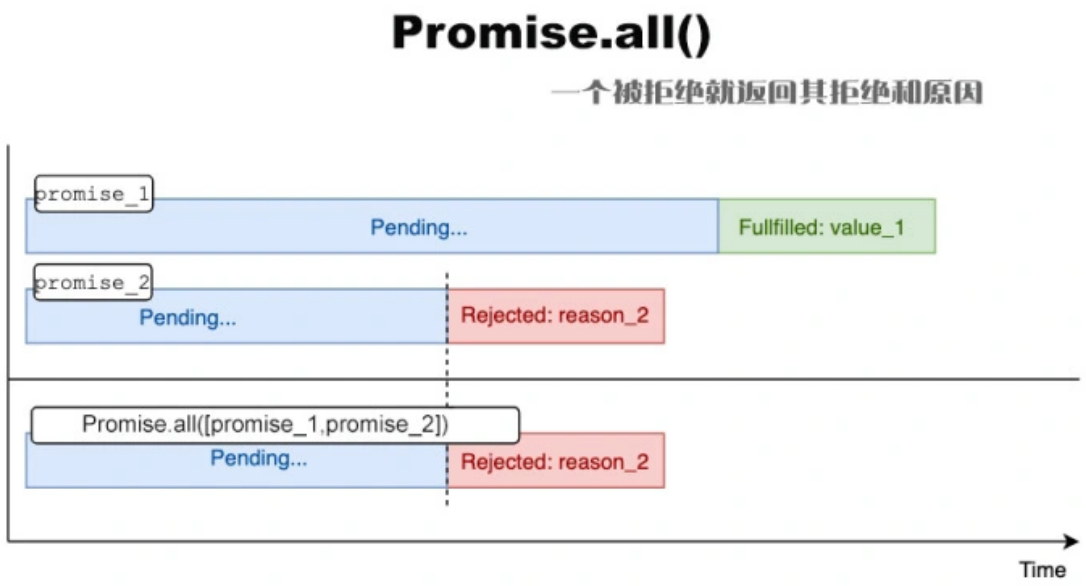
```
const values = await allPromise;
console.log(values); // [valueOfPromise1, valueOfPromise2, ...]

```

`Promise.all()` 返回的 `promise` 被解析或拒绝的方式。

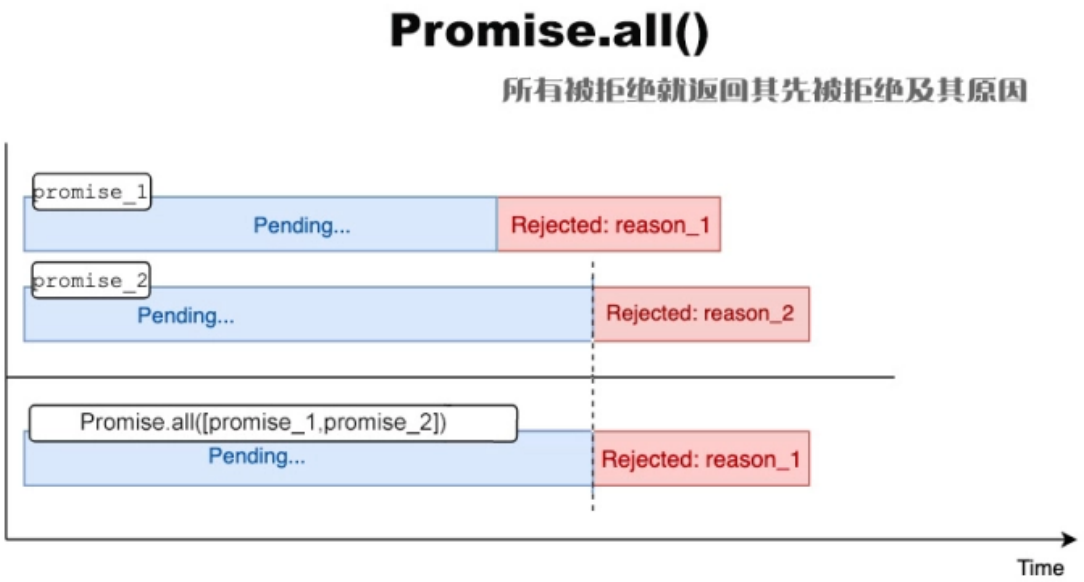
如果 `allPromise` 都被成功解析，那么 `allPromise` 将使用一个包含各个 `promise` 已执行完成后的值的数组作为结果。数组中 `promise` 的顺序是很重要的——将按照这个顺序得到已实现的值。



但是如果至少有一个 `promise` 被 `rejected`，那么 `allPromise` 会以同样的原因立即 `rejected`（不等待其他 `promise` 的执行）。



如果所有的 `promise` 被 `rejected` ，等待所有的`promise` 执行完成，但只会返回最先被`rejected` 的`promise` 的 `reject` 原因。



2. 使用指南

现在来看一下 `Promise.all()`的实际使用情况， 在这之前，先来定义 2 个简单的函数。

> 函数 `resolveTimeout(value, delay)` 将返回一个在经过 `delay` 时间后有 `resolve` 的 `promise` 。

复制代码

```
function resolveTimeout(value, delay) {
    return new Promise((resolve) => setTimeout(() => resolve(value), delay));
}

```

> 函数 `rejectTimeout(reason, delay)` 将返回一个在经过 `delay` 时间后有 `reject` 的 `promise` 。

```
function rejectTimeout(reason, delay) {
    return new Promise((r, reject) => setTimeout(() => reject(reason), delay));
}

```

接下来使用上面定义的 2 个辅助函数来试试 `Promise.all()` 。

2.1 完成所有 `promises`

下面定义了一个 `promise` 数组 `allPromise` ，所有的 `promise` 都能够成功的 `resolve` 值，如下：

```
function resolveTimeout(value, delay) {
    return new Promise((resolve) => setTimeout(() => resolve(value), delay));
}
const fruits = ["potatoes", "tomatoes"];
const vegetables = ["oranges", "apples"];

const allPromise = [
    resolveTimeout(fruits, 2000),
    resolveTimeout(vegetables, 1000),
];
const promise = Promise.all(allPromise);

// 等待... 2秒后
const list = async () => {
    try {
        const result = await promise;
        console.log(result);
    } catch (error) {
        console.log(error.errors);
    }
};

list(); // [ [ 'potatoes', 'tomatoes' ], [ 'oranges', 'apples' ] ]
```

```
24
```

从上面执行的结果来看 `Promise.all()` 返回的 `promise` 的 `resolve` 数组是按照执行前 `allPromise` 的顺序组成其结果。

> `promise` 数组的顺序直接影响结果的顺序，和 `promise` 执行完成的先后无关。

2.2 一个 `promise` 被 `rejected`

将上面数组 `allPromise` 的第一个 `promise` 出现异常被 `rejected`，如下代码：

复制代码

```
const promise = Promise.all([
    rejectTimeout(new Error("fruits is empty"), 5000),
    resolveTimeout(vegetables, 1000),
]);

// 等待...
const list = async () => {
    try {
        const result = await promise;
        console.log(result);
    } catch (error) {
        console.log(error);
    }
};

list(); // Error: fruits is empty
```

然而，在经过 `5秒` 之后，第一个 `promise` 由于异常被 `rejected` ，使得 `allPromise` 也被 `rejected` ，并返回跟第一个 `promise` 一样的错误信息：`Error: fruits is empty` ，即使在 `1秒` 后就完成的第二个 `promise` 的值也不被采纳。

接下来将数组 `allPromise` 的所有 `promise` 都抛出异常被 `rejected` ，通过定时器将 `rejected` 的顺序做个调整，如下：

复制代码

```
const promise = Promise.all([
    rejectTimeout(new Error("fruits is empty"), 5000),
    rejectTimeout(new Error("vegetables is empty"), 1000),
]);

// 等待...
const list = async () => {
    try {
        const result = await promise;
        console.log(result);
    } catch (error) {
        console.log(error);
    }
};

```

经过 `5秒` 之后完成执行，而结果显示为 `Error: vegetables is empty` ，不难看出 `allPromise` 被 `rejected` 的原因是最先 `rejected` 的`promise` 。

> `Promise.all()` 的这种行为被称为快速失败，如果 `promise` 数组中至少有一个 `promise` 被 `rejected` ，那么返回的 `promise` 也被拒绝。如果`promise` 数组中所有的都被 `rejected` ，那么返回的`promise` 被拒绝的原因是先`rejected`的那一个。

总结

`Promise.all()` 是并行执行异步操作并获取所有 `resolve` 值的最佳方法，非常适合需要同时获取异步操作结果来进行下一步运算的场合。

发布于: 2021-09-03 | 阅读数: 1702

Promise 异步任务 9月日更

devpoint
细节的追求者 · 2011-11-11 加入
专注前端开发，用技术创造价值！

评论

• 暂无评论 •