javascript

Promise

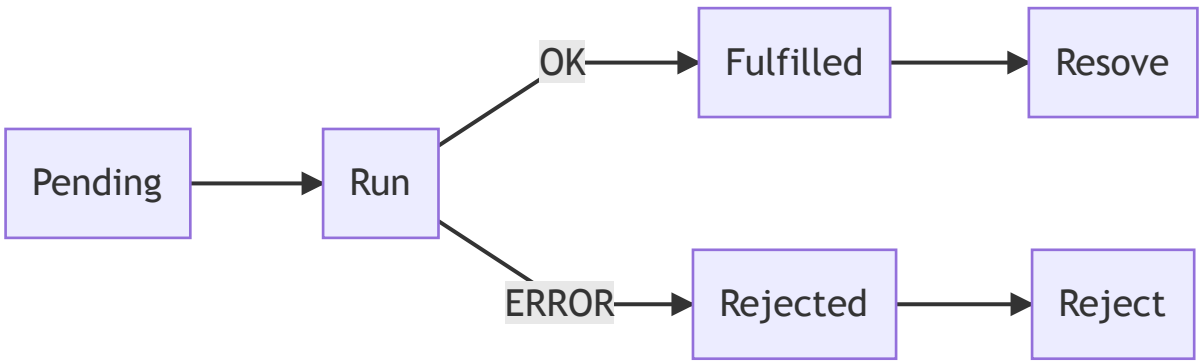
基础用法

```
new Promise( function(resolve, reject) {...} /* executor */  );

var promise = new Promise(function(resolve, reject){
   //do something
   //if ok
   resolve
   //if error
   reject
});

promise.then(f1(), f2()).catch(f2())

//f1 = resolve method
//f2 = reject method
```



链式调用：

```
job1.then(job2).then(job3).catch(handleError)

Promise.all[job1,job2].then(...)
Promise.race[job1,job2].then(...)
```

Api 定义： https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Promise

async/await

```
async function f() {

  let promise = new Promise((resolve, reject) => {
    setTimeout(() => resolve("done!"), 1000)
  });

  let result = await promise; // wait till the promise resolves (*)

  alert(result); // "done!"
}

f();
```

`await` literally makes JavaScript wait until the promise settles, and then go on with the result. That doesn't cost any CPU resources, because the engine **can do other jobs meanwhile: execute other scripts, handle events** etc.

When we use `async/await`, we rarely need `.then`, because `await` handles the waiting for us. And we can use a regular `try..catch` instead of `.catch`. That's usually (not always) more convenient.

The `async` keyword before a function has two effects:

1. Makes it always return a promise.
2. Allows to use `await` in it.

The `await` keyword before a promise makes **JavaScript wait until that promise settles**, and then:

1. If it's an error, the exception is generated, same as if `throw error` were called at that very place.
2. Otherwise, it returns the result, so we can assign it to a value.

Together they provide a great framework to write asynchronous code that is easy both to read and write.

With `async/await` we rarely need to write `promise.then/catch`, but we still shouldn't forget that they are based on promises, because sometimes (e.g. in the outermost scope) we have to use these methods. Also `Promise.all` is a nice thing to wait for many tasks simultaneously.