

## 2-ES6 Refresher

Monday, April 11, 2022 9:34 PM

### 6- Arrow Functions

主意下面filter的用法，过滤保留符合条件的元素

```
const jobs = [
  { id: 1, isActive: true },
  { id: 2, isActive: true },
  { id: 3, isActive: false },
];

const activeJobs = jobs.filter(function(job) { return job.isActive; });
const activeJobs = jobs.filter(job => job.isActive);
```

### 8- Array.map Method

Template literal: `\${color}`

```
const colors = ['red', 'green', 'blue'];
const items = colors.map(color => `<li>${color}</li>`);
console.log(items);
```

### 9- Object Destructuring

```
const address = {
  street: '',
  city: '',
  country: ''
};

const street = address.street;
const city = address.city;
const country = address.country;

const { street: st } = address;
```

### 10- Spread Operator

```
const first = { name: "Mosh" };
const second = { job: "Instructor" };

const clone = { ...first };
```

对象是“通过引用”被存储和复制的。被复制的新对象是指向原对象的引用地址的。修改新对象就是修改原对象。两个对象（除非复制）只有内容一样，===判断返回false，因为引用地址不同。clone对象（创建新对象，不复制引用）：object.assign

```

1 let user = { name: "John" };
2
3 let permissions1 = { canView: true };
4 let permissions2 = { canEdit: true };
5
6 // 将 permissions1 和 permissions2 中的所有属性都拷贝到 user 中
7 Object.assign(user, permissions1, permissions2);
8
9 // 现在 user = { name: "John", canView: true, canEdit: true }

```

除了spread，assign也可以浅clone对象。扩展运算符也是浅拷贝。深拷贝（object里还有object，浅拷贝，内层的object还是指向原来的），可以使用lodash库的deepclone。

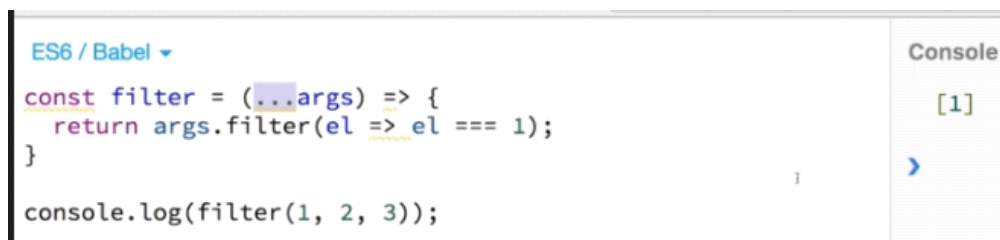
```

1 let user = {
2   name: "John",
3   age: 30
4 };
5
6 let clone = Object.assign({}, user);

```

<https://zh.javascript.info/object-copy>

查询一个array中的特定值：使用filter函数。点点点意味着，将所有参数，拼成一个数组，所以才可以使用filter函数。



```

ES6 / Babel
const filter = (...args) => {
  return args.filter(el => el === 1);
}
console.log(filter(1, 2, 3));

```

Console

[1]

## 11- Classes

We have an object with at least one method, we need a blueprint(蓝图) to create object of that type.

移动整段代码：全选，按住alt，箭头向下/上移动

第一张图片中，如果有两个person，具有相似的结构，name，walk。就应该写一个class，参考第二张图片中。

```

const person = {
  name: "Mosh",
  walk() {
    console.log("walk");
  }
};

```

```

class Person {
  constructor(name) {
    this.name = name;
  }

  walk() {
    console.log("walk");
  }
}

const person = new Person('Mosh');
person.

```

## 12- Inheritance/继承/嗨瑞ten死/

```

class Person {
  constructor(name) {
    this.name = name;
  }

  walk() {
    console.log("walk");
  }
}

class Teacher extends Person {
  teach() {
    console.log("teach");
  }
}

const teacher = new Teacher('Mosh');
teacher.

```

继承类如果重新定义了constructor, 需要call父类的, super()

```
class Teacher extends Person {  
  constructor(name, degree) {  
    super(name);  
    this.degree = degree;  
  }  
  
  teach() {  
    console.log("teach");  
  }  
}  
  
const teacher = new Teacher("Mosh", "MSc");
```

### 13- Modules

Each file, we call it module

## 3-Components

Tuesday, April 12, 2022 9:29 PM

### 2- Setting Up the Project

```
$npm i bootstrap@4.1.1
```

```
index.js
1 import React from 'react';
2 import ReactDOM from 'react-dom';
3 import './index.css';
4 import App from './App';
5 import registerServiceWorker from './registerServiceWorker';
6 import 'bootstrap/dist/css/bootstrap.css';
```

### 3- Your First React Component

Imrc

```
import React, { Component } from 'react';
```

Cc

```
class Counter extends Component {
  state = {}
  render() {
    return ( )
  }
}

export default Counter;
```

Jsx可以被compile是因为import react这句话

### 4- Specifying Children

div和react.fragment的区别，在浏览器f12的element中，能看到div，看不到react.fragment

### 6- Setting Attributes

方法一：className

方法二：style

```
styles = {
  fontSize: 50,
  fontWeight: "bold"
};

<span style={this.styles} className="badge badge-primary m-2">
```

方法三：{{ }}

```
<span style={{ fontSize: 30 }} className="badge badge-primary m-2">
```

## 7- Rendering Classes Dynamically

```
render() {  
  let classes = "badge m-2 badge-";  
  classes += this.state.count === 0 ? "warning" : "primary";  
  
  return (  
    <div>  
      <span className={classes}>{this.formatCount()}</span>  
      <button className="btn btn-secondary btn-sm">Increment</button>  
    </div>  
  );  
}
```

使用refactor，可以直接产生需要的function，是代码更简明且结构化

```
render() {  
  return (  
    <div>  
      <span className={this.getBadgeClasses()}>{this.formatCount()}</span>  
      <button className="btn btn-secondary btn-sm">Increment</button>  
    </div>  
  );  
}  
  
getBadgeClasses() {  
  let classes = "badge m-2 badge-";  
  classes += this.state.count === 0 ? "warning" : "primary";  
  return classes;  
}
```

## 8 - Rendering Lists

li这个元素需要设置unique key，因为如果这个元素的state在虚拟DOM中变化了，react需要迅速指出是哪个元素改变了，在真实DOM中哪里需要改变。

```
<ul>{this.state.tags.map(tag => <li key={tag}>{tag}</li>)}</ul>
```

## 9- Conditional Rendering

jsx不是templating engine，jsx中不能出现if else

Conditional Rendering方法一：



```

class Counter extends Component {
  state = {
    count: 0,
    tags: []
  };

  renderTags() {
    if (this.state.tags.length === 0) return <p>There are no tags!</p>;

    return <ul>{this.state.tags.map(tag => <li key={tag}>{tag}</li>)}</ul>;
  }

  render() {
    return <div>{this.renderTags()}</div>;
  }
}

export default Counter;

```

方法二：

```
{this.state.tags.length === 0 && "Please create a new tag!"}
```

&&这个符号可以有两个及两个以上判断，如果每个都是true返回最后一个，如果有一个false，返回false

## 11- Binding Event Handlers

如果一个函数被一个object call如下图中第一行，this返回这个object。

如果一个函数单独被call，如下图中第二行，this返回window。

如果使用的strict mode，则返回undefined。

```

// obj.method();
// function();

```

constructor意味着这个class初始化的时候会call这个con函数，super是类继承，因为class Counter extends Component

```

constructor() {
  super();
  this.handleIncrement = this.handleIncrement.bind(this);
}

handleIncrement() {
  console.log("Increment Clicked", this);
  // obj.method();
  // function();
}

```

使用箭头函数也可以实现bind this的效果。是因为，匿名箭头函数中的 this 将从外部作用域继承 this。更多解释: <https://zh-hans.reactjs.org/docs/faq-functions.html>

```

handleIncrement = () => {
  console.log("Increment Clicked", this);
  // obj.method();
  // function();
};

```

### 13- What Happens When State Changes

当state发生变化的时候，react会计划一次render，异步的，在未来，也许5毫秒也许10毫秒

### 14- Passing Event Arguments

in onClick we need to pass the function reference, so here we cannot handle call increment

and pass an argument like one, we need to pass a function,

上面的话是说，不能写成下面这样，需要call 一个函数，以便传递参数。

```
onClick={this.handleIncrement(1)}
```

就是写成一下这样

```

onClick={this.doHandleIncrement}
handleIncrement = (product) => {
  console.log(product);
  this.setState({ count: this.state.count + 1 });
};

doHandleIncrement = () => {
  this.handleIncrement({ id: 1 });
};

```

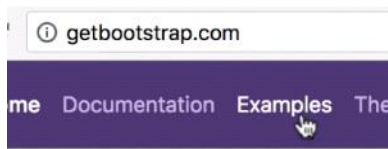
就没必要写这么复杂，直接：



```
onClick={ () => this.handleIncrement({ id: 1 }) }
```

## 15- Setting Up the Vidly Project

```
$npm i bootstrap@4.1.1 font-awesome@4.7.0
```



## 17- Building the Movies Component

需要什么就去bootstrap document里找，比如找table

```
table.table>thead>tr>th*4
```

tr: row, th: hear, td: data

```
<table>
  <tr>
    <th>Person 1</th>
    <th>Person 2</th>
    <th>Person 3</th>
  </tr>
  <tr>
    <td>Emil</td>
    <td>Tobias</td>
    <td>Linus</td>
  </tr>
  <tr>
    <td>16</td>
    <td>14</td>
    <td>10</td>
  </tr>
</table>
```

```
return <table className="table">
  <thead>
    <tr>
      <th>Title</th>
      <th>Genre</th>
      <th>Stock</th>
      <th>Rate</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>4
    </td>
  </tr>
</tbody>
</table>
```

## 19- Conditional Rendering

下图中count, 利用destructor, 简化变量, 得到movies的length

```
const { length: count } = this.state.movies;

if (count === 0)
  return <p>There are no movies in the database.</p>;

return (
  <p>Showing {count} movies in the database.</p>
)
```

## 4-Composing Components

Thursday, April 14, 2022 5:52 PM

### 3- Passing Data to Components

key不会被当成props传递下去

### 8 - Updating the State

为啥要传递一模一样的两个props, key 和 ID, 因为key是react的内置变量, 不会被当成props传递下去。

传递参数的时候, 不用把props中的所参数都一一写出来。props应该被写在一个object中

```
<Counter
  key={counter.id}
  onDelete={this.handleDelete}
  value={counter.value}
  id={counter.id}
  selected={counter.selected}
/>
<Counter
  key={counter.id}
  onDelete={this.handleDelete}
  counter={counter}
/>
```

### 9- Single Source of Truth

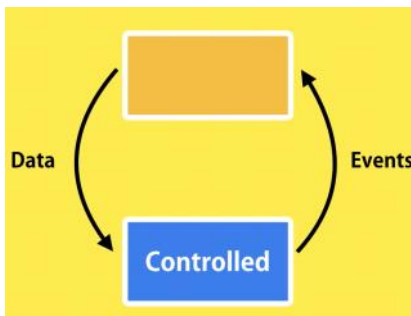
Each of our components have our own local state, our counters component, has an array of counters, and each counter component has a value. This value is currently disconnected from the value property of each counter object that we have in this array. Here's the implementation

```
class Counter extends Component {
  state = {
    value: this.props.counter.value
  };
}
```

the value property of our state object, based on what we get from our props. This piece of code is executed only once when an instance of a counter

### 10- Removing the Local State

将component中的state删掉, 完全依赖props, 这就是controlled component (没有自己的local state)



## 12- Lifting the State Up

函数调用VS.函数传递/引用

更多解释: <https://zh-hans.reactjs.org/docs/faq-functions.html>

以下两者是等价的:

```
//  onClick={() => this.props.onDelete(this.props.counter.id)}
onClick={this.props.onDelete.bind(this, this.props.counter.id)}
```

## 15 - Lifecycle Hooks

以下方法, 在这三个情况下, 被依次调用



## 16 - Mounting Phase

mount:

**an instance of a class is created.**

Constructor: 是一个初始化的好时机

componentDidMount: 是call ajax的好机会, get data from server. Did mounted之后意味着, 这个component已经在真实DOM中了

当一个component被render了, 他的所有子组件也被递归地 (recursively) render

App – Constructor

App – Rendered

NavBar – Rendered

Counters – Rendered

Counter – Rendered

App – Mounted

## 17- Updating Phase

整个组件tree被从render，不代表整个DOM被更新。当一个组件被rendered，我们basically得到一个react element。更新虚拟DOM

在以下函数中，如果应该进行新旧比较，如果有变化，就ajax request 更新数据  
下面的if中比较之前的props与当前的props是否一致，如果有变化就进行。。。

```
class Counter extends Component {
  componentDidUpdate(prevProps, prevState) {
    console.log("prevProps", prevProps);
    console.log("prevState", prevState);
    if (prevProps.counter.value !== this.props.counter.value) {
      // Ajax call and get new data from the server
    }
  }
}
```

## 18- Unmounting Phase

在这个组件即将被从DOM中移除的时候，这是个做清洁的好时机

**called just before a component is removed from the DOM.**

当delete一个counter的时候

**app component is changed. So, our entire component is re rendered, we have app, navbar, counters, as well**

其他几个组件都会rendered，然后

**we have a new virtual DOM which has one less counter. React will compare this virtual DOM with the old one, it figures out that one of our counters is removed, so then it will call component will unmount before removing this counter from the DOM.**

做清洁的时机，否则可能会内存leaks

**So if you have set up timers, or listeners, we can clean those up**

icon



font awesome 4.7



All

News

Images

Shopping

Videos

More

Settings

Tools

About 24,700,000 results (0.43 seconds)

## Font Awesome 4.7.0

<https://fontawesome.com/v4.7.0/> ▼

Font Awesome gives you scalable vector icons that can instantly be customized — size, color, drop shadow, and anything that can be done with the power of ...

### Icons

You asked, Font Awesome delivers with 41 shiny new icons ...

### Fa-search

fa-search · Unicode: f002 · Created: v1.0 · Categories: Web ...

## 5-Pagination, Filtering, and Sorting

Saturday, April 16, 2022 10:44 PM

### 4- Pagination- Displaying Pages

为了生产一个固定长度的array:

```
$npm i lodash@4.17.10  
const pages = _.range(1, pageCount + 1);
```

### 7- Pagination- Type Checking with PropTypes

如果是reusable component, 最好用type checking, check传递到其中的props的type。这也是很好的doc, 可以不用看里面, 直接看props的要求

```
$npm i prop-types@15.6.2
```

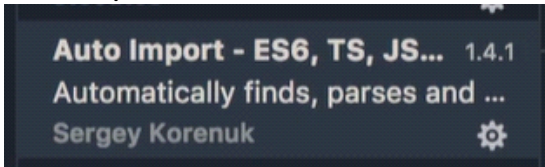
### 15- Sorting- Extracting MoviesTable

Reusable component在common文件夹下, 其他在component文件夹下

## 6- Routing

Wednesday, April 20, 2022 9:50 PM

### 2- Setup



### 3- Adding Routing

```
$npm i react-router-dom@4.3.1
```

在index.js中写下面这句

```
import { BrowserRouter } from 'react-router-dom';

ReactDOM.render(
  <BrowserRouter>
    <App />
  </BrowserRouter>,
  document.getElementById("root")
);
registerServiceWorker();
```

```
import { Route } from 'react-router-dom';
```

### 4- Switch

返回的原则是，以path开头的component，如果不写switch会返回最后一个match的

绝对路径：

方法一：exact必须一模一样的url才会返回

方法二：switch，会返回第一个一样的，之后的route都会被忽略

### 5- Link

每次换router位置，都会发出request，下载全部bundle（包含全部html信息）。为了避免如此，使用link代替html中的<a>

### 6- Route Props

利用router包裹的component会自动拥有以下三个属性

#### Props

- ▶ history: {...}
- ▶ location: {...}
- ▶ match: {...}

React router doc:

You've visited this page 3 times. Last visit: 7/12/18

## React Router: Declarative Routing for React.js - React Training

<https://reacttraining.com/react-router/core>

Philosophy. This guide's purpose is to explain the mental model to have when using **React Router**. We call it "Dynamic Routing", which is quite different from the ...

[Route](#) · [Basic](#) · [Quick Start](#)

You've visited this page 5 times. Last visit: 7/12/18

### 7- Passing Props

想要传其余参数参，利用以下方法

```
<Route
  path="/products"
  render={(props) => <Products sortBy="newest" {...props} />}
/>
```

### 8- Route Parameters

```
<Switch>
  <Route path="/products/:id" />
  <h1>Product Details - {this.props.match.params.id} </h1>
```

### 9- Optional Parameters

加了? 号后，这两个参数就是可传可不传的，不传就会跳到posts页面。如果没有问号，不传就会跳到home

```
<Route path="/posts/:year?/:month?" component={Posts} />
```

### 10- Query String Parameters

其实应该避免使用optional parameter，因为Query String Parameters中包含问号。

```
$npm i query-string@6.1.0
```

When you dealing

with optional parameters, instead of including them in the route, you should include them in query string.

query string is what we append to a url using a question mark, localhost:3001/posts?sortBy=newest&approved=true

在location的search中

```
import React from "react";
import queryString from "query-string";

const Posts = ({ match, location }) => {
  const result = queryString.parse(location.search);
  console.log(result);
}
```

► {approved: "true", sortBy: "newest"}

## 11- Redirects

如果是错误的url就跳转到redirect, 指向的地方

```
<Route path="/not-found" component={NotFound} />
<Route path="/" exact component={Home} />
<Redirect to="/not-found" />
```

还有第二种用法, 如果url中用户输入了message, 就会直接跳转到posts

```
<Redirect from="/messages" to="/posts" />
```

## 12- Programmatic Navigation

push, replace

this push method will add a new address in the browser history, so you can click the back button and go back to where you were. But replace basically replaces the current address so we won't have history.

点击save button就会trigger以下方法, 结果是会跳转到product位置。之后再按back就会回到save button所在的页面。如果是replace方法, back之后不会回到原来的页面。这个方法经常用在login page, 用户login之后, 你不希望他们按back还回到login page

```
handleSave = () => {
  // Navigate to /products
  this.props.history.push('/products');
};
```

## 13- Nested Routing

route这个component可以随时随地用, 不用非在app中



```
<ul>
  <li>
    <Link to="/admin/posts">Posts</Link>
  </li>
  <li>
    <Link to="/admin/users">Users</Link>
  </li>
</ul>

<div>
  <h1>Admin Dashboard</h1>
  <SideBar />
  <Route path="/admin/users" component={Users} />
  <Route path="/admin/posts" component={Posts} />
</div>
```

# Admin Dashboard

- [Posts](#)
- [Users](#)

## 7-Forms

Monday, April 25, 2022 4:29 PM

### 4- Refs

```
username = React.createRef();  
  
<input ref={this.username} />  
  
const username = this.username.current.value;
```

Ref has a property 'current':

**current which returns the actual DOM element. Now we can<sup>↗</sup>**

如果你想access DOM

**really need to access the DOM, that's the way to do it.**

只有当你非常清楚ref的作用时再用它

manage the focus of an input field, you really want to get a reference to that DOM element. Or maybe you really want to work with animations, or third party dom libraries. In those cases it's okay to use

方法一：利用ref

```
componentDidMount() {  
  this.username.current.focus();  
}
```

方法二：添加autoFocus属性

```
<input  
  autoFocus  
  ref={this.username}  
  id="username"  
  type="text"  
  className="form-control"/>
```

### 5- Controlled Elements

Controlled elements 与 controlled components 一样，他们没有自己的state，他们get all the data from props, they notify changes to the data by raising events

### 7- Common Errors

付给input的value (controlled element) , 不能是null或者undefined，否则会报错，可以设为"空"



## 8-Calling Backend Services

Tuesday, April 19, 2022 11:53 AM

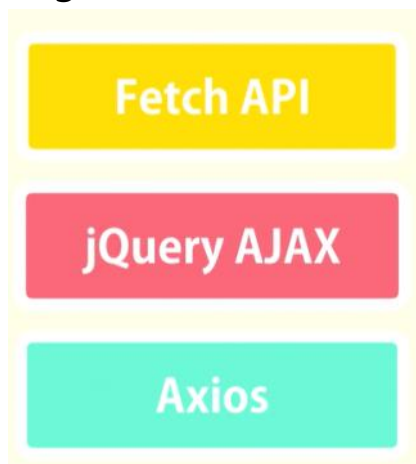
### 2- JSON Placeholder

假后端

<https://jsonplaceholder.typicode.com>

### 3- Http Clients

angular有自己自带的module, react中你可以选择一下三个, 推荐axios



作者喜欢axios

```
$npm i axios@0.18
```

### 4- Getting Data

axios返回一个promise

promise有一个initial state, 是pending。当operation结束以后, pending会变成resolved或者rejected

```
// pending > resolved (success) OR rejected (failure)
```

利用then方法, 可以得到异步操作的结果, 这是原始的方法。我们现在用await方法, 得到response, await必须用在异步方法中。

```
componentDidMount() {  
  // pending > resolved (success) OR rejected (failure)  
  const promise = axios.get("https://jsonplaceholder.typicode.com/posts");  
  promise.then()  
}  
  
async componentDidMount() {  
  // pending > resolved (success) OR rejected (failure)  
  const promise = axios.get("https://jsonplaceholder.typicode.com/posts");  
  const response = await promise;  
  console.log(response);  
}
```

结果中有一个属性，叫data，是我们需要的

```
async componentDidMount() {
  const { data: posts } = await axios.get(
    "https://jsonplaceholder.typicode.com/posts"
  );
  this.setState({ posts });
}
```

## 5- Creating Data

async位置不同

```
async componentDidMount() {
  // pending > resolved (success) OR rejected (failure)
  const { data: posts } = await axios.get(apiEndpoint);
  this.setState({ posts });
}

handleAdd = async () => {
  const obj = { title: 'a', body: 'b' };
  const { data: post } = await axios.post(apiEndpoint, obj);
};
```

## 7- Updating Data

patch可以只改一个property， put改all properties

```
handleUpdate = post => {
  post.title = "UPDATED";
  axios.put(apiEndpoint + '/' + post.id, post);
  axios.patch(apiEndpoint + '/' + post.id, { title: post.title });
};
```

state的变化问题：当state中的一个item在handle change中变化，state直接就变了，但是页面不会变化，需要setState之后，页面才会变化。下图中，单一post的title变了，页面没变化，但是this.state的这个post的title已经变了。所以才可以用indexOf来找这个post，因为在state中，这个post已经是新的了，可以找到。

```
handleUpdate = async (post) => {
  console.log("state", this.state.posts[0]);
  post.title = "UPDATED";
  // const { data } = await axios.put(`${apiEndpoint}/${post.id}`, post);
  // console.log("data", data);
  console.log("post", post);
  console.log("state", this.state.posts[0]);
  // const posts = [...this.state.posts];
  // const index = posts.indexOf(post);
  // // posts[index] = { ...post };
  // posts[index] = { ...data };
  // this.setState({ posts });
};
```



## 10- Expected vs Unexpected Errors

```
// Expected (404: not found, 400: bad request) – CLIENT ERRORS
// - Display a specific error message
//
// Unexpected (network down, server down, db down, bug)
// - Log them
// - Display a generic and friendly error message
```

如果成功send request则有值，否则为null。如果全程没问题则respond有值，否则null

```
} catch (ex) {
  ex.request
  ex.response
}
```

## 11- Handling Unexpected Errors Globally

使用axios.interceptor拦截器

可以拦截request也可以拦截response

有两个参数，都是函数，第一个函数是在成功之后被call，第二个是在这个response中有error的时候被call

异常捕获exception

```
try {
  try_statements
}
catch (exception_var) {
  catch_statements
}
finally {
  finally_statements
}
```

Promise: [https://www.w3schools.com/js/js\\_promise.asp](https://www.w3schools.com/js/js_promise.asp)

```

let myPromise = new Promise(function(myResolve, myReject) {
  // "Producing Code" (May take some time)

  myResolve(); // when successful
  myReject();  // when error
});

// "Consuming Code" (Must wait for a fulfilled Promise)
myPromise.then(
  function(value) { /* code if successful */ },
  function(error) { /* code if some error */ }
);

```

When the producing code obtains the result, it should call one of the two callbacks:

Result	Call
Success	myResolve(result value)
Error	myReject(error object)

如果这个promise成功了，其中就包含myResolve，包含一些信息。利用promise.then的时候，就会trigger第一个函数参数。

```

<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Promise</h2>

<p id="demo"></p>

<script>
function myDisplayer(some) {
  document.getElementById("demo").innerHTML = some;
}

let myPromise = new Promise(function(myResolve, myReject) {
  let x = 6;

  // some code (try to change x to 5)

  if (x == 0) {
    myResolve("OK");
  } else {
    myReject("Error");
  }
});

myPromise.then(
  function(value) {myDisplayer(value);},
  function(error) {myDisplayer(error);}
);
</script>

</body>
</html>

```

打断并且给出一个error

Promise.reject(new PermissionDenied());

VS.

throw new PermissionDenied();

<https://stackoverflow.com/questions/33445415/javascript-promises-reject-vs-throw>

Promise.reject(error);返回一个promise object, 具体解释看下图

## Promise.reject()

The `Promise.reject()` method returns a `Promise` object that is rejected with a given reason.

### Try it

JavaScript Demo: Promise.reject()

```
1 function resolved(result) {
2   console.log('Resolved');
3 }
4
5 function rejected(result) {
6   console.error(result);
7 }
8
9 Promise.reject(new Error('fail')).then(resolved, rejected);
10 // expected output: Error: fail
11
```

Run ›

Reset

> Error: fail

## 14- Displaying Toast Notifications

```
$npm i react-toastify@4.1
import { ToastContainer } from 'react-toastify';
import 'react-toastify/dist/ReactToastify.css';
```

```
import { toast } from 'react-toastify';
```

Replace alert

```
toast.error("An unexpected error occurred.");
```

上下两图的区别是, error message 红色, 普通message彩色,

```
toast("An unexpected error occurred.");
```

## 15- Logging Errors

一个记录error的server, 需要注册

Secure | <https://sentry.io/welcome/>

```
$npm i raven-js@3.26.4
```

```
import Raven from 'raven-js';
import './index.css';
import "bootstrap/dist/css/bootstrap.css";

Raven.config('https://05323d37c9a947eba9daaaab1e6171a9@sentry.io/1249956', {
  release: '0-0-0',
  environment: 'development-test',
}).install()
```

raven代替console.log记录error

```
if (!expectedError) {
  Raven.captureException(error);|
  toast.error("An unexpected error occurred.");
}
```