**React 16 Course - Component Lifecycle Methods**

March 11, 2018

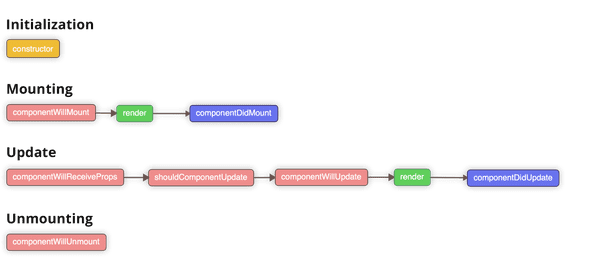
Like everything in existence – react components follow their natural cycle of creation, existence and destruction. This article is part of **WIP** React 2018 course for beginners.

**React Lifecycle Methods**

Let’s see what happens when you add your component to render tree.

Overall we have 4 big phases in components life.

* [Initialization](https://maksimivanov.com/posts/react_lifecycle_methods/#initialization),
* [Mounting](https://maksimivanov.com/posts/react_lifecycle_methods/#mounting)
* [Update](https://maksimivanov.com/posts/react_lifecycle_methods/#update)
* [Unmounting](https://maksimivanov.com/posts/react_lifecycle_methods/#unmounting)

[](https://maksimivanov.com/static/2ffa2b4eab0cfc3c8f74d76657f7b3ec/c4fa1/lifecycle.png)

In every phase there are several methods being called

Initialization

First goes the **initialization** phase.

And first one to be called is the constructor method. This is the place where you initialize your component, set your initial state, default props or bind this to functions that need it. Don’t use this method for anything other than setting up your component. Please, no AJAX calls or reading/writing to database.

Mounting

Initialization phase is followed by **mounting** phase.

First method that gets executed in this phase is componentWillMount and its usage is mostly same as of constructor – to set up initial state and default props. Warning, componentWillMount and other componentWill... methods are deprecated, use componentDidMount or constructor instead of this one.

Next render method is executed. You are already familiar with this one. This method mounts component to DOM.

After the component is mounted to DOM – componentDidMount gets called. As this function is being executed only once in the whole lifecycle – it’s a good place to put your AJAX requests.

Update

After **mounting** phase is completed – component goes to **update** phase.

First method to be called in this phase is componentWillReceiveProps(newProps). This method is getting called every time props that are passed to this component get updated. This method receives all the props so you can manually update your state depending on what props did change. Warning this method is deprecated along with most other componentWill... methods.

Next is shouldComponentUpdate(nextProps, nextState, nextContext). By default component gets updated every time props passed to it, it’s state or its context gets changed. This method allows you to prevent unnecessary update by returning false.

Then goes componentWillUpdate(nextProps, nextState). Deprecated as a few other componentWill... methods.

Then as in **mounting** phase, we have render method.

After render method was executed successfully we have componentDidUpdate which you can use to perform AJAX calls and other side-effect causing operations.

Since React 16 we have now have componentDidCatch(errorString, errorInfo) method. This method works similar to Javascript try/catch block but for components. You can use this method in parent component to catch errors happened in its children. It receives to parameters

* errorString – the message of an error
* info – an object with a single field componentStack which represent the stack trace back to where the error occurred.

Keep in mind that this method will only catch errors in the components below the parent in the tree. It won’t catch the errors happened in the component itself.

Unmounting

And the final phase of life of any react component is **unmounting** phase.

There is only one method in this phase. It’s componentWillUnmount and this is the only one compoenentWIll... method that didn’t get deprecated.

This method is executed just before your component gets removed from the **DOM**

Use this method to clean up after your component. Remove all timers or listeners created during the lifetime of your component to prevent memory leaks.

There are many different approaches to write React Components. When you go through lots of tutorials, you will notice that react components are varying greatly from tutorial to tutorial. However, every components are derived from functional and class component, which is the basic thing. So, today in this article, I wanna show you guys what React components can do by describing different types of React Components.

***What’s React Component?*** *A****React Component****is one of the core building blocks of React Apps. In other words, we can say that every application you will develop in React will be made up of pieces called components. Components make the task of building UIs much easier.*

**from:** [https://www.geeksforgeeks.org/reactjs](https://www.geeksforgeeks.org/reactjs-)/

Each component returns/renders some JSX code and it defines which HTML code React should render to the real DOM in the end. JSX is not HTML but it looks a lot like it. In React we mainly have two types of components: **Functional Components and Class Components**

However, there are other components in React that goes similar to these two components and today we will discuss about them as well.

**React Components:**

**1] Functional Components:**In simple words, Functional components are javascript functions. By writing a javascript function, we can create a functional component in React Apps. To make React app efficient, we use functional component only when we are sure that our component does not require to interact with any other components. Functional components do not require data from other components. Below shows an example of functional component in React:

function Title()  
{  
 return <h1>I am Title</h1>;  
}

**2] Class Components**The class components are similar to the functional component but has some additional features that makes class component a little more complex than the functional components. The functional components do not care about the other components in your app whereas the class components can work with each other. We can pass data from one class component to other class component. Below shows an example of class component in React:

class Title extends React.Component  
{  
 render(){  
 return <h1>I am Title</h1>;  
 }  
}

**3] Higher-Order Components**Higher order components, or known under the abbreviation HOCs are the component which takes a component as input and returns the component as output but with extended functionalities. React Higher-Order Components are popular advanced React pattern to deploy reusable logic and functionalityacross React components. Below shows an example higher-order component that transforms and returns usernames in uppercase:

const hoc = (MyComponent) => (props) => {  
 return (  
 <div>  
 <MyComponent {...props}>  
 {props.children.toUpperCase()}  
 </MyComponent>  
 </div>  
 )  
}

**4] Dumb Components**A Dumb Component can very easily be defined as a stateless functional component. A stateless component is much more efficient than a stateful one, because it doesn't require as many computer resources to render (memory, CPU, and GPU in terms of graphic-intensive apps). Below shows an example of dumb component in React:

export default Title => () => (  
 <h1>I am Title</h1>  
);

**5] Smart Components**A Smart Component is any class component that manages its own state. Smart Components are stateful components and works similar to class components. When working with Babel or ES6-style React, we’ve come to know this as any class-like object that extends Component.  
This includes either React.Component or in our case Other.Component. Below shows an example of smart component in React:

export default title => class MyComponent extends Other.Component {  
 render() {  
 return (  
 <h1>I am Title</h1>  
 );  
 }  
}

**6] Presentational Components**The presentation component is often called as stateless functional component that takes props and renders UI. A stateless functional components are plain JavaScript functions that do not have states. The components that receive state from the higher-order component will function as presentational components. State gets passed to them and they conditionally render UI based on it. They do not bother with the management of state. Presentational Component mainly concerned with how things look. Below shows an example of presentational component in React:

const List = props =>   
 (<ul>   
 {props.list.map(user =>   
 (<li>{items}</li> ))   
 }   
 </ul>)

**7] Container components**Container component is a class component that provides the data and behavior to presentational or other container components. A container does data fetching and then renders its corresponding sub-component. This component mainly concerned with how things work. Container components call flux actions and provides these as callbackss to the presentational component. Below shows an example of presentational component in React:

class ListContainer extends React.Component{  
constructor()   
 {   
 this.state = {   
 items: []   
 }   
 }   
 componentDidMount() {   
 axios.get('/list').then(  
 items=>this.setState({ list: items}))   
 )   
}   
render() {   
 return<Usersusers={this.state.items} />   
 }}

So these are the react components that you should be aware of.

When a client makes a request to an HTTP server — and the server successfully receives the request — **the server must notify the client if the request was successfully handled or not.**

HTTP accomplishes this with five categories of status codes:

* 100-level (Informational) – server acknowledges a request
* 200-level (Success) – server completed the request as expected
* 300-level (Redirection) – client needs to perform further actions to complete the request
* 400-level (Client error) – client sent an invalid request
* 500-level (Server error) – server failed to fulfill a valid request due to an error with server

Based on the response code, a client can surmise the result of a particular request.

### 3.1. Basic Responses

The simplest way we handle errors is to **respond with an appropriate status code.**

Here are some common response codes:

* 400 Bad Request – client sent an invalid request, such as lacking required request body or parameter
* 401 Unauthorized – client failed to authenticate with the server
* 403 Forbidden – client authenticated but does not have permission to access the requested resource
* 404 Not Found – the requested resource does not exist
* 412 Precondition Failed – one or more conditions in the request header fields evaluated to false
* 500 Internal Server Error – a generic error occurred on the server
* 503 Service Unavailable – the requested service is not available

### Standardized Response Bodies

While most REST APIs follow similar conventions, specifics usually vary, including the names of fields and the information included in the response body. These differences make it difficult for libraries and frameworks to handle errors uniformly.

In an effort to standardize REST API error handling, **the IETF devised**[**RFC 7807**](https://tools.ietf.org/html/rfc7807)**, which creates a generalized error-handling schema**.

This schema is composed of five parts:

1. type – a URI identifier that categorizes the error
2. title – a brief, human-readable message about the error
3. status – the HTTP response code (optional)
4. detail – a human-readable explanation of the error
5. instance – a URI that identifies the specific occurrence of the error

JVM perform some particular **types of operations**:

1. Loading of code
2. Verification of code
3. Executing the code
4. It provide run-time environment to the users