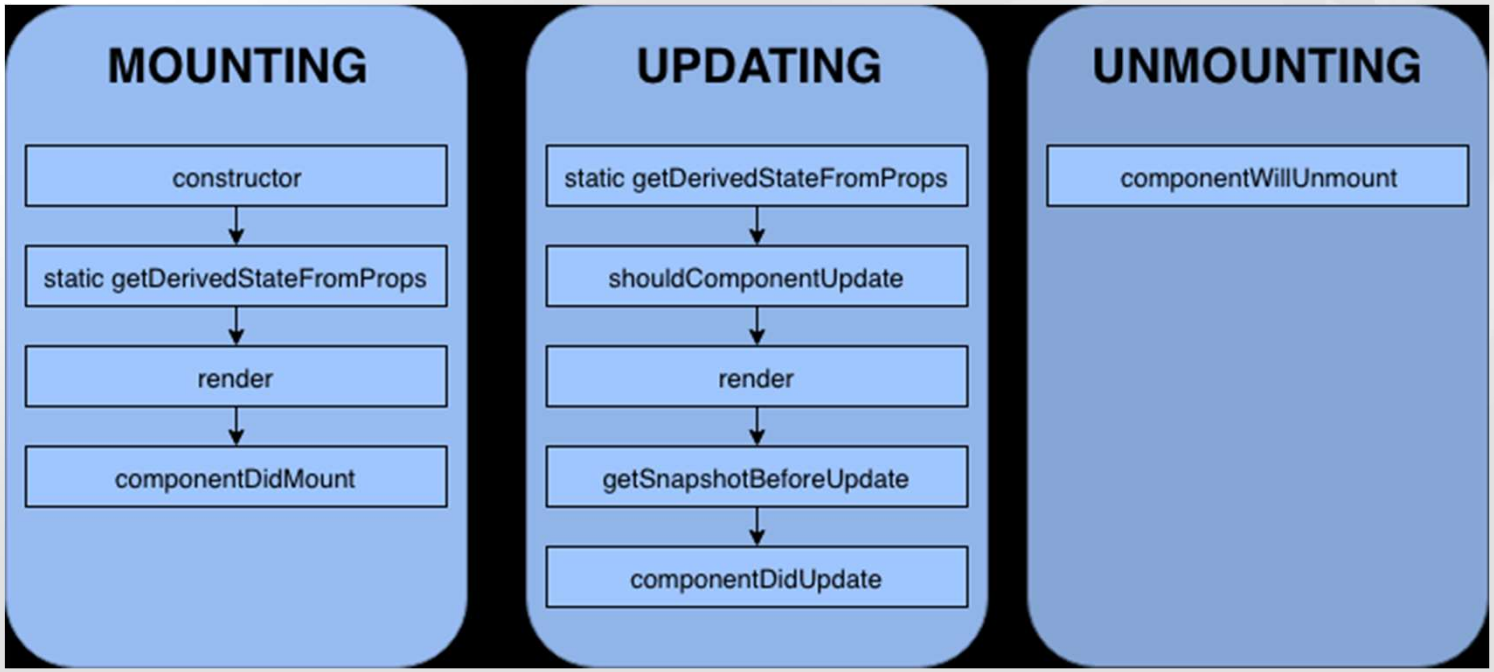
Chapitre 7 : Cycle de vie

Présentation

- Le cycle de vie d'un composant regroupe les étapes de la vie d'un composant.
- Les composants basés sur des classes passent par toute une série d'étapes au cours de leur vie.
- React permet de réagir à ces étapes en implémentant dans les classes des méthodes aux noms spécifiques, appelées méthodes de cycle de vie.
- Un cycle de vie d'un component correspond donc à un changement de son état.
- Il existe 3 cycles de vie : mounting,updating et unmounting.
- Le **mounting** d'un component (monter un component en français) correspond à l'affichage d'un component à l'écran. Ce cycle de vie englobe l'initialisation du component jusqu'à son affichage à l'écran.
- **Unmounting** (démonter), par déduction, est la suppression d'un component de l'écran.
- **Updating** est le cycle de vie qui correspond à la mise à jour du component. C'est lorsque l'on met à jour des données et que le composant est re-rendu.

Méthodes de cycle de vie



Mounting - Le montage

constructor

- Comme dans la programmation objet, le constructeur est ce qui est appelé en premier.
- Il intervient dès que le composant doit apparaître dans le DOM virtuel.
- Le constructeur de class reçoit en premier paramètre ses props.
- Si on l'implémente, il sera important de bien appeler la classe parente avec le mot-clé super afin de lui fournir les props.
- Attention, durant cet évènement, les éléments du DOM n'existent pas.
- Si le composant doit utiliser un état local, il faut lui affecter l'état initial à this.state directement dans le constructeur.
- Le constructeur est le seul endroit où on doit attribuer directement this.state.

```
constructor(props) {
super(props);
      this.state = { color: "red"};
}
```

Mounting - Le montage
getDerivedStateFromProps 1/2

- getDerivedStateFromProps permet à un composant de mettre à jour son état interne suite à un changement de ses props.
- Elle doit renvoyer un objet qui met à jour l'état, ou null faute de mise à jour.
- Cette méthode existe pour les rares cas où l'état dépend bien des changements de props au fil du temps.
- La méthode getDerivedStateFromProps () est appelée juste avant le rendu des éléments dans le DOM.

App.js :

```
import React from 'react';
import Header from './Header';
class App extends React.Component {
  render() {
    return (      <Header nouvPrix="70" />       );
  }
}
export default App;
```

Mounting - Le montage
getDerivedStateFromProps 2/2

Header.js :

```
import React from 'react';
class Header extends React.Component {
  constructor(props) {
    super(props);
    this.state = {prix: "40"};
  }
  static getDerivedStateFromProps(props, state) {
    return {prix: props.nouvPrix };
   }
  render() {
    return (    <h1>Le prix est {this.state.prix}</h1>    );
  }
}
export default Header;
```

Le résultat obtenu est : **Le prix est 70**

Mounting - Le montage

render

- Cette méthode est celle du rendu.
- Elle intervient pour rendre le JSX dans le DOM virtuel (et donc générer le HTML).
- C'est à ce moment là qu'on a l'état du composant à jour.
- Que ce soit les props ou le state, les données sont disponibles et prêtes à être manipulées afin de rendre ce qu'on souhaite.

```
render() {
  return (
    <h1>Le continu du component</h1>
  );
}
```

Mounting - Le montage
componentDidMount 1/2

- componentDidMount() est appelée immédiatement après que le composant est monté (inséré dans l'arbre).
- C'est ici qu'on doit placer les initialisations qui requièrent l'existence de nœuds du DOM.
- Car le DOM est correctement chargé et dans cette fonction on pourrait manipuler les éléments du DOM.
- On a le droit d'appeler setState() directement dans componentDidMount().
- L'exemple suivant, initialement le résultat donne :

Le prix est 30

- Puis après quelques secondes l'affichage change ce qui permettra de voir qu'il y a eu deux rendus :

Le prix est 50

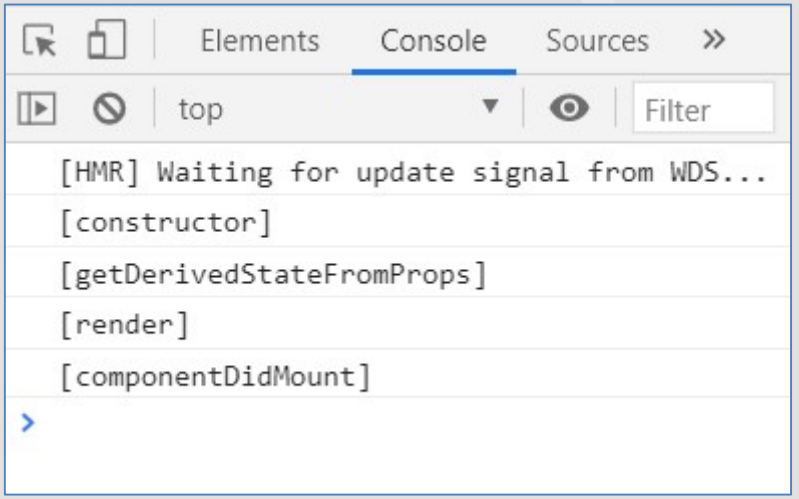
Mounting - Le montage
componentDidMount 2/2

```
import React from 'react';
class Header extends React.Component {
  constructor(props) {
    super(props);
    this.state = {prix: 30};
  }
  componentDidMount() {
    setTimeout(() => {
      this.setState({prix: 50})
    }, 1000)
  }
  render() {
    return (
      <h1>Le prix est {this.state.prix}</h1>
    );
  }
} export default Header;
```

Mounting - Le montage

Ordre des d'exécution des différents éléments

```
import React, { Component } from 'react'
class Ordre extends Component {
    constructor(props) {
        super(props);
        this.state = {composant: "Ordre"};
        console.log('[constructor]')     }
    static getDerivedStateFromProps(props, state) {
        console.log('[getDerivedStateFromProps]');
         return {composant: props.composant };     }
    componentDidMount() {
        console.log('[componentDidMount]');     }
    render() {
        console.log('[render]')
        return <div>{this.state.composant}</div>
    }
}
export default Ordre;export default Header;
```



Updating- La mise à jour

- Le cycle de vie de la mise à jour s'active dans 2 cas :
  - La modification du state, de l'état local
  - La modification d'une props de l'un des parents
- Dès qu'il intervient un de ces 2 cas, le composant va déclencher un re-render.
- Dès lors, il va engager les différentes méthodes de cette catégorie, puis re-rendre le JSX avec la mise à jour de son état.

Updating- La mise à jour
getDerivedStateFromProps 1/3

- Lors des mises à jour il s'agit de la première méthode appelée lorsqu'un composant est modifié.

App.js :

```
import React from 'react';
import Couleurs from './Couleurs';
class App extends React.Component {
   render() {
    return (
      <Couleurs favcol="bleu"/>
    );
  }
}
export default App;
```

Updating- La mise à jour
getDerivedStateFromProps 2/3

```
import React from 'react';
class Couleurs extends React.Component {
  constructor(props) {
    super(props);
    this.state = {favoritecolor: "jaune"};   }
  static getDerivedStateFromProps(props, state) {
    console.log("[getDerivedStateFromProps state ]",state.favoritecolor) ;
    console.log("[getDerivedStateFromProps props ]",props.favcol) ;
    return {favoritecolor: props.favcol };   }
  changeColor = () => {
    this.setState({favoritecolor: "rouge"});    }
  render() {
    return (
      <div>   <h1>Ma couleur préférée est le  {this.state.favoritecolor}</h1>
     <button type="button" onClick={this.changeColor}>Changer couleur</button>    </div>
    );
  }
} export default Couleurs;
```

Updating- La mise à jour
getDerivedStateFromProps 3/3

Dans cet exemple on a un bouton qui change la couleur préférée en rouge, mais comme la méthode getDerivedStateFromProps () est appelée, la couleur préférée est toujours rendue en bleu parce que la méthode met à jour l'état avec la couleur de l'attribut favcol (**props.favcol)**.

Etat initial :

En cliquant sur le bouton :

Updating- La mise à jour
shouldComponentUpdate() 1/2

- shouldComponentUpdate() est utilisée pour indiquer à React que la sortie d'un composant n'est pas affectée par la modification en cours de l'état local ou des props.
- Dans la méthode shouldComponentUpdate (), on peut renvoyer une valeur booléenne qui spécifie si React doit continuer ou non le rendu.
- La valeur par défaut est true.
- shouldComponentUpdate() est appelée avant le rendu quand de nouvelles props ou un nouvel état local sont reçues.

Updating- La mise à jour
shouldComponentUpdate() 2/2

- Dans cet exemple shouldComponentUpdate retourne false, L'affichage du message persiste toujours « Ma couleur préférée est le vert » même si on appuie sur le bouton « Changer couleur ».
- Néanmoins le message se transforme en « Ma couleur préférée est le rouge », si le return est true.

```
import React from 'react';
class Couleurs  extends React.Component {
  constructor(props) {
    super(props);
    this.state = {favoritecolor: "vert"};   }
  shouldComponentUpdate() {
    return false;   }
  changeColor = () => {
    this.setState({favoritecolor: "rouge"});    }
  render() {
    return (
      <div>
      <h1>Ma couleur préférée est le  {this.state.favoritecolor}</h1>
      <button type="button" onClick={this.changeColor}>Changer couleur</button>
      </div>
    );
  }
} export default Couleurs ;
```

Updating- La mise à jour
render

- La méthode render () est bien sûr appelée lorsqu'un composant est mis à jour, il doit restituer le HTML dans le DOM, avec les nouvelles modifications.

```
import React from 'react';
class Couleurs  extends React.Component {
  constructor(props) {
    super(props);
    this.state = {favoritecolor: "jaune"};   }

  changeColor = () => {
    this.setState({favoritecolor: "bleu"});    }

  render() {
    return (
      <div>
      <h1>Ma couleur préférée est le  {this.state.favoritecolor}</h1>
      <button type="button" onClick={this.changeColor}>Changer couleur</button>
      </div>
    );
  }
}
export default Couleurs ;
```

Updating- La mise à jour
getSnapshotBeforeUpdate 1/3

- getSnapshotBeforeUpdate() est appelée juste avant que le rendu le plus récent ne soit validé, par exemple envoyé au DOM.
- Elle permet de capturer des informations du DOM courant avant qu'il ne subisse d'éventuelles modifications.
- Toute valeur renvoyée par cette méthode de cycle de vie sera passée comme argument à componentDidUpdate().

```
import React from 'react';

class Couleurs  extends React.Component {
   constructor(props) {
      super(props);
      this.state = {favoritecolor: "rouge"};
    }
    componentDidMount() {
      setTimeout(() => {
        this.setState({favoritecolor: "jaune"})
      }, 3000)
    }
```

Updating- La mise à jour
getSnapshotBeforeUpdate 2/3

```
getSnapshotBeforeUpdate(prevProps, prevState) {
    let rep = "Avant la mise à jour, la couleur favorite était le  " + prevState.favoritecolor;
    document.getElementById("div1").innerHTML = rep;
    return rep;
  }
  componentDidUpdate() {
    document.getElementById("div2").innerHTML =
    "La nouvelle couleur MAJ est " + this.state.favoritecolor;
  }
  render() {
    return (
      <div>
      <h1>Ma couleur préférée est le {this.state.favoritecolor}</h1>
      <div id="div1"></div>
      <div id="div2"></div>
      </div>
    );
  }    }   export default Couleurs;
```

Updating- La mise à jour
getSnapshotBeforeUpdate 3/3

Initialement on a le résultat suivant :

Ma couleur préférée est le rouge

Ensuite l'affichage suivant est généré :

Ma couleur préférée est le jaune

Avant la mise à jour, la couleur favorite était le rouge
La nouvelle couleur MAJ est jaune

Updating- La mise à jour
componentDidUpdate 1/2

- componentDidUpdate() est appelée immédiatement après que la mise à jour a eu lieu.
- Cette méthode n'est pas appelée pour le rendu initial.
- Elle donne l'opportunité de travailler sur le DOM une fois que le composant a été mis à jour.

```
import React from 'react';
class Header  extends React.Component {
 constructor(props) {
   super(props);
   this.state = {prix: 30};
 }
 componentDidMount() {
   setTimeout(() => {
     this.setState({prix: 50})
   }, 1000)
 }
 componentDidUpdate() {
   document.getElementById("div1").innerHTML =
   "La valeur modifiée du prix est : " + this.state.prix;
 }
```

Updating- La mise à jour
componentDidUpdate 2/2

```
render() {
    return (
      <div>
         <h1>Le prix est égal à :  {this.state.prix}</h1>
         <div id="div1"></div>
    </div>
    );
  }
}
export default Header;
```

Avant modification

Le prix est égal à : 30

Après modification

La valeur modifiée du prix est : 50

Le prix est égal à : 50

Unmounting- Le démontage

componentWillUnmount 1/4

- La phase suivante du cycle de vie est lorsqu'un composant est supprimé du DOM, ou démonté comme React l'appelle.
- React n'a qu'une seule méthode intégrée qui est appelée lorsqu'un composant est démonté qui s'appelle componentWillUnmount().
- Elle est appelée immédiatement avant qu'un composant soit démonté ou détruit. C'est-à-dire que le composant est sur le point d'être supprimé du DOM.
- Si on a besoin de nettoyer quoi que ce soit concernant ce composant, on peut le faire dans cette méthode.

Unmounting- Le démontage
componentWillUnmount 2/4

Header.js :

```
import React from 'react';
import Persons from './Persons';
class Header extends React.Component {
   constructor(){
      super();
      this.state={
        toggleUser:true
      }
    }
       render(){
      return (<div>
{ this.state.toggleUser ? <Persons /> : null }
<button onClick={()=>{this.setState({toggleUser:!this.state.toggleUser})}}>Supprimer Infos Utilisateur</button>
              </div>
             );
              }
   }  export default Header;
```

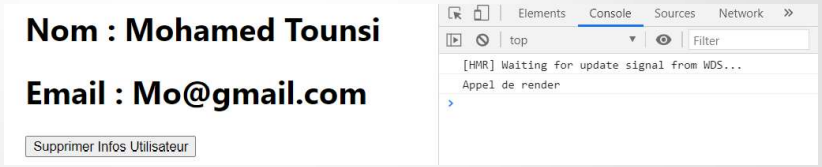
Unmounting- Le démontage
componentWillUnmount 3/4

Persons.js :

```
import React from 'react'
class Persons extends React.Component{
   componentWillUnmount(){
      console.log('Appel de componentWillUnmount')
      alert('Utilisateur supprimé');
   }
   render(){
      console.log('Appel de render')
      return(
         <div>
            <h1>Nom : Mohamed Tounsi</h1>
            <h1>Email : Mo@gmail.com</h1>
         </div>
      )
   }
}
export default Persons;
```

Unmounting- Le démontage
componentWillUnmount 4/4

Le résultat obtenu initialement affiche les informations relatives à l'utilisateur :



En cliquant sur le bouton :

Les informations relatives à l'utilisateur sont supprimées :