生命周期

课堂目标

1. 掌握组件生命周期

资源

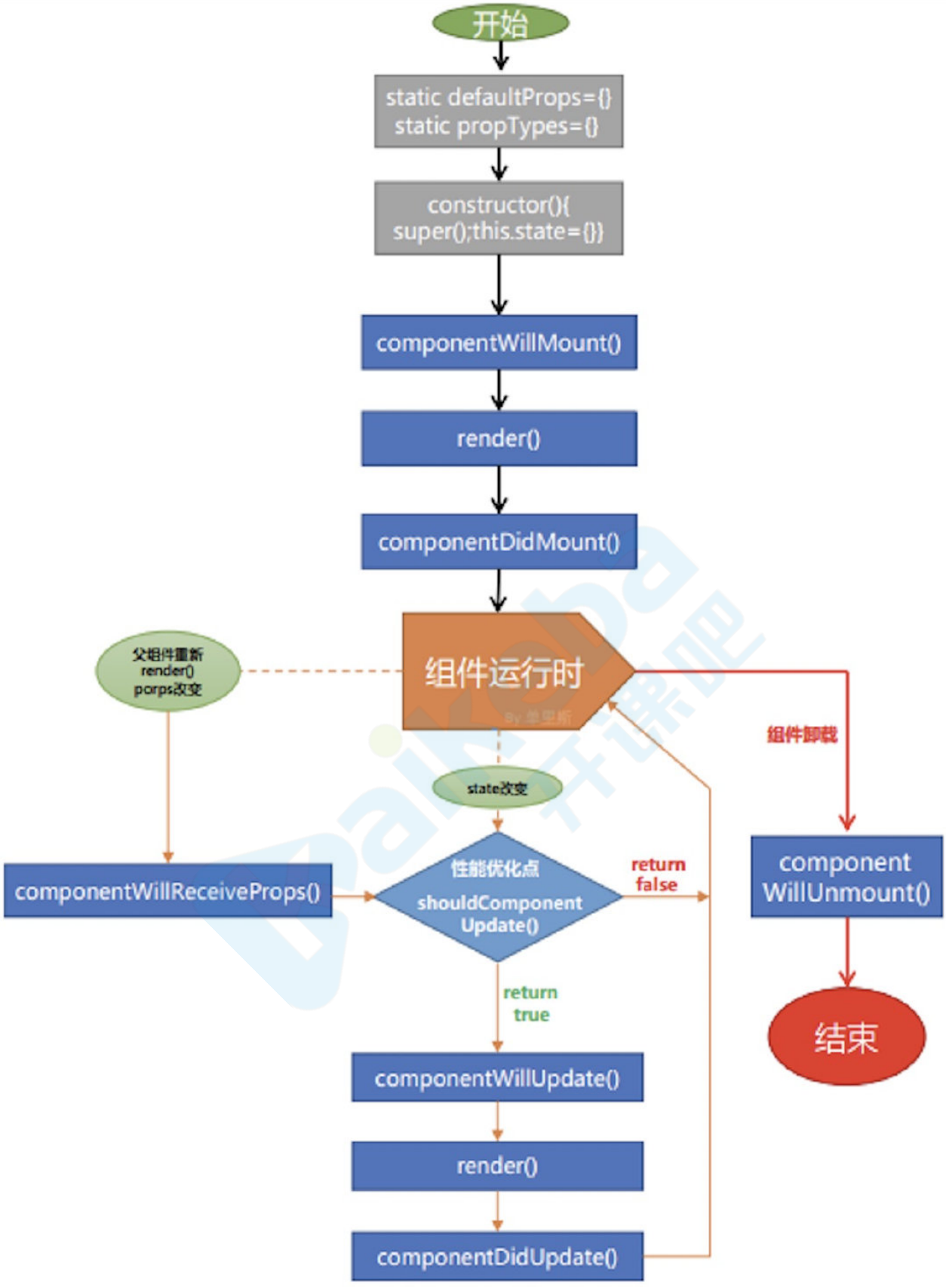
1. 组件的生命周期
2. 生命周期图谱

知识要点

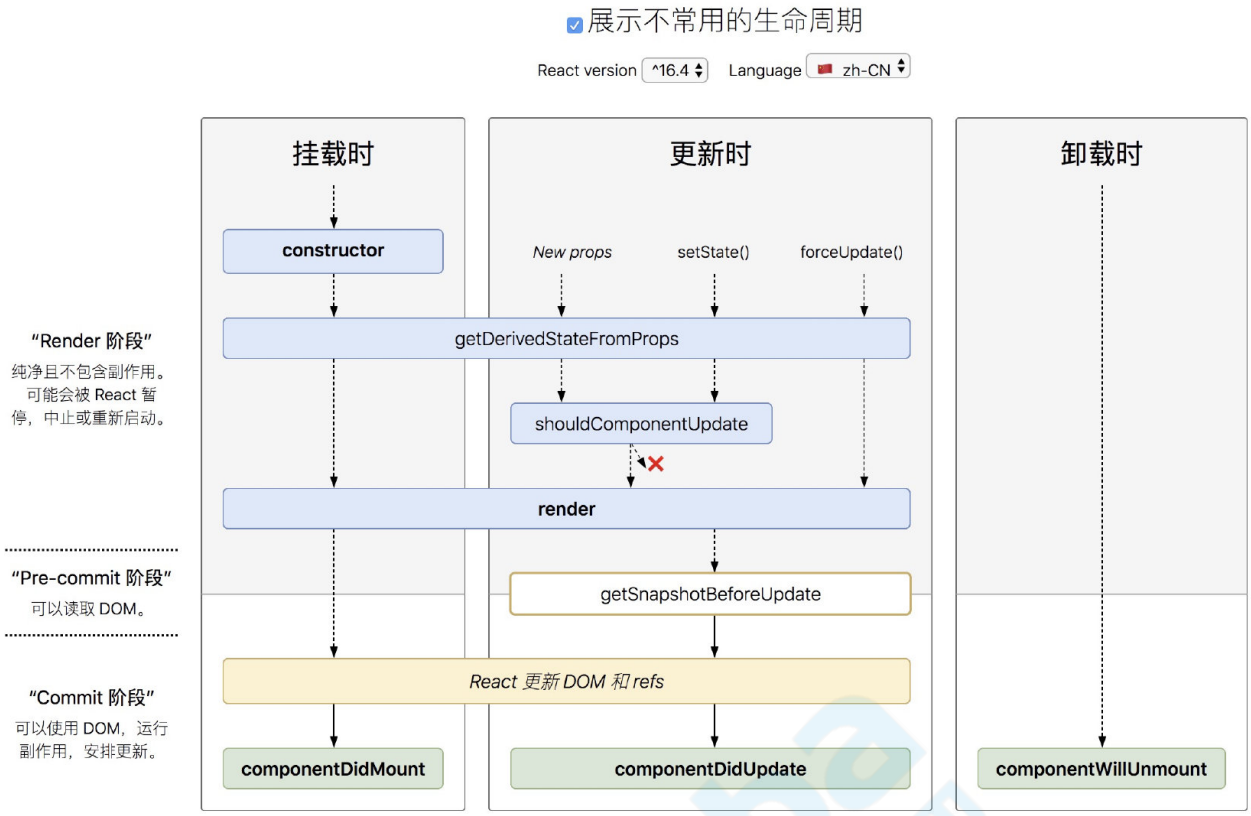
生命周期方法

生命周期方法，用于在组件不同阶段执行自定义功能。在组件被创建并插入到 DOM 时（即挂载中阶段（mounting）），组件更新时，组件取消挂载或从 DOM 中删除时，都有可以使用的生命周期方法。

React V16.3之前的生命周期



V16.4之后的生命周期：



V17可能会废弃的三个生命周期函数用getDerivedStateFromProps替代，目前使用的话加上UNSAFE_：

- componentWillMount
- componentWillReceiveProps
- componentWillUpdate

引入两个新的生命周期函数：

- static getDerivedStateFromProps
- getSnapshotBeforeUpdate

如果不想手动给将要废弃的生命周期添加 `UNSAFE_` 前缀，可以用下面的命令。 统一添加UNSAFE_前缀

```
npx react-codemod rename-unsafe-lifecycles <path>
```

新引入的两个生命周期函数

getDerivedStateFromProps

```
static getDerivedStateFromProps(props, state)
```

`getDerivedStateFromProps` 会在调用 render 方法之前调用，并且在初始挂载及后续更新时都会被调用。它应返回一个对象来更新 state，如果返回 null 则不更新任何内容。

请注意，不管原因是什么，都会在每次渲染前触发此方法。这与 `UNSAFE_componentWillReceiveProps` 形成对比，后者仅在父组件重新渲染时触发，而不是在内部调用 `setState` 时。

getSnapshotBeforeUpdate

```
getSnapshotBeforeUpdate(prevProps, prevState)
```

在render之后，在componentDidUpdate之前。

`getSnapshotBeforeUpdate()` 在最近一次渲染输出（提交到 DOM 节点）之前调用。它使得组件能在发生更改之前从 DOM 中捕获一些信息（例如，滚动位置）。此生命周期的任何返回值将作为参数传递给 `componentDidUpdate(prevProps, prevState, snapshot)`。

验证生命周期

范例：创建LifeCyclePage.js

```
import React, { Component } from "react";
import PropTypes from "prop-types";
/*
V17可能会废弃的三个生命周期函数用getDerivedStateFromProps替代，目前使用的话加上
UNSAFE_:
- componentWillMount
- componentWillReceiveProps
- componentWillUpdate
 */

export default class LifeCyclePage extends Component {
  static defaultProps = {
    msg: "omg"
  };
  static propTypes = {
    msg: PropTypes.string.isRequired
  };
  constructor(props) {
    super(props);
    this.state = {
      count: 0,
    };
    console.log("constructor", this.state.count);
  }

  static getDerivedStateFromProps(props, state) {
    // getDerivedStateFromProps 会在调用 render 方法之前调用，
    //并且在初始挂载及后续更新时都会被调用。
```

```
    //它应返回一个对象来更新 state，如果返回 null 则不更新任何内容。
    const { count } = state;
    console.log("getDerivedStateFromProps", count);
    return count < 5 ? null : { count: 0 };
  }
  //在render之后，在componentDidUpdate之前。
  getSnapshotBeforeUpdate(prevProps, prevState, snapshot) {
    const { count } = prevState;
    console.log("getSnapshotBeforeUpdate", count);
    return null;
  }
  /* UNSAFE_componentWillMount() {
    //不推荐，将会被废弃
    console.log("componentWillMount", this.state.count);
  } */
  componentDidMount() {
    console.log("componentDidMount", this.state.count);
  }
  componentWillUnmount() {
    //组件卸载之前
    console.log("componentWillUnmount", this.state.count);
  }
  /* UNSAFE_componentWillUpdate() {
    //不推荐，将会被废弃
    console.log("componentWillUpdate", this.state.count);
  } */
  componentDidUpdate() {
    console.log("componentDidUpdate", this.state.count);
  }

  shouldComponentUpdate(nextProps, nextState) {
    const { count } = nextState;
    console.log("shouldComponentUpdate", count, nextState.count);
    return count !== 3;
  }

  setCount = () => {
    this.setState({
      count: this.state.count + 1,
    });
  };

  render() {
    const { count } = this.state;
    console.log("render", this.state);
    return (
      <div>
        <h1>我是LifeCycle页面</h1>
        <p>{count}</p>
```

```
        <button onClick={this.setCount}>改变count</button>
        {/* {!!(count % 2) && <Foo />} */}
        <Child count={count} />
      </div>
    );
  }
}

class Child extends Component {
  UNSAFE_componentWillReceiveProps(nextProps) {
    //不推荐，将会被废弃
    // UNSAFE_componentWillReceiveProps() 会在已挂载的组件接收新的 props 之前被调用
    console.log("Foo componentWillReceiveProps");
  }
  componentWillUnmount() {
    //组件卸载之前
    console.log(" Foo componentWillUnmount");
  }
  render() {
    return (
      <div
        style={{ border: "solid 1px black", margin: "10px", padding: "10px" }}
      >
        我是Foo组件
        <div>Foo count: {this.props.count}</div>
      </div>
    );
  }
}
```