



Core Concepts

Components

Components are the core building blocks of Preact applications. They manage state and render UI.

Functional Components (Hooks):

```
import { useState } from 'preact/hooks';

function MyComponent(props) {
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() =>
        setCount(count + 1)}>Increment</button>
    </div>
  );
}
```

Class Components:

```
import { Component } from 'preact';

class MyComponent extends Component {
  constructor(props) {
    super(props);
    this.state = { count: 0 };
  }

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

  render() {
    return (
      <div>
        <p>Count: {this.state.count}</p>
        <button onClick=
          {this.increment}>Increment</button>
      </div>
    );
  }
}
```

JSX & Rendering

Preact uses JSX to describe UI. The `render` function transforms JSX into DOM nodes.

JSX Example:

```
<div className="container">
  <h1>Hello, Preact!</h1>
  <p>This is a simple example.</p>
</div>
```

Rendering:

```
import { render } from 'preact';
import MyComponent from './MyComponent';

render(<MyComponent />,
  document.getElementById('app'));
```

Virtual DOM

Preact uses a virtual DOM to efficiently update the actual DOM. It compares the previous and current virtual DOM trees and only applies the necessary changes.

Hooks

useState

Declares a state variable.

```
import { useState } from 'preact/hooks';

const [count, setCount] = useState(0);
```

Updating the state triggers a re-render.

```
setCount(count + 1);
```

useEffect

Performs side effects in functional components.

```
import { useEffect } from 'preact/hooks';
```

```
useEffect(() => {
  // Code to run after render
```

```
document.title = `Count: ${count}`;
```

```
return () => {
  // Cleanup function
  (optional);
}, [count]);
// Dependency array
```

useRef

Creates a mutable ref object.

```
import { useRef } from 'preact/hooks';

const inputRef = useRef(null);
```

Accessing the current value:

```
inputRef.current.focus();
```

Component Lifecycle

Lifecycle Methods (Class Components)

`componentDidMount()` : Invoked immediately after a component is mounted.

```
componentDidMount() {
  // Perform initialization tasks here
}
```

`componentWillUnmount()` : Invoked immediately before a component is unmounted and destroyed.

```
componentWillUnmount() {
  // Perform cleanup tasks here
}
```

`shouldComponentUpdate(nextProps, nextState)` : Determines if the component should re-render.

```
shouldComponentUpdate(nextProps, nextState) {
  // Return true to update, false to prevent update
  return nextProps.value !== this.props.value;
}
```

`componentDidUpdate(prevProps, prevState)` : Invoked immediately after an update occurs.

```
componentDidUpdate(prevProps, prevState) {
  // Perform side effects based on the update
```

Preact CLI

Creating a New Project

Preact CLI is the recommended way to start a new Preact project.

```
npx preact-cli create my-app  
cd my-app  
npm install  
npm run dev
```

Common Commands

<code>npm run dev</code>	Starts the development server with hot reloading.
<code>npm run build</code>	Builds the project for production.
<code>npm run serve</code>	Serves the production build locally.

Configuration

Preact CLI uses a `preact.config.js` file to configure the build process.

```
// preact.config.js  
export default (config, env, helpers) =>  
{  
  // Customize webpack config here  
};
```