React 1

Jiří Zacpal

KATEDRA INFORMATIKY
UNIVERZITA PALACKÉHO V OLOMOUCI

KMIWEBA Webové aplikace

React

- Přirozeným krokem po předchozích seminářích je začít tvořit části aplikace nebo dokonce aplikaci celou pomocí JavaScriptu (přesunutí back-endu na frontend). Zásadní potíže nám ale způsobi DOM, respektive neefektivita s jakou JS s DOM pracuje (jakákoliv změna na stránce představuje přepočítání a renderování celého DOM).
- Tento problém byl vyřešen pomocí virtuálního DOM a shadow DOM. Virtuální DOM představuje vnitřní reprezentaci skutečného DOM. Veškerá práce se odehrává s virtuálním DOM a následně jsou potřebné změny (efektivně) promítnuty do skutečného DOM. Shadow DOM umožňuje zapouzdřit část DOM do samostatně stojícího celku (prohlížeč se nemusí starat co je uvnitř).
- Další potíže nám způsobuje nízkoúrovňovost JS. I poměrně jedndoduché úkony s DOM je nesnadné zapsat a při zavedení virtuálního DOM se situace ještě výrazně komplikuje. Zásadním konceptem je přechod z imperativního stylu programování na deklarativní, přesněji řečeno na reaktivní programování (varianta deklarativního paradigma).
- Viruální a shadow DOM, stejně tak reaktivní paradigma je adaptováno v mnoha JS framworcích, které jsou určeny pro tvorbu UI komponent nebo webových aplikací. V principu lze na webovou aplikaci nahlížet jako na sadu UI komponent. Příklady takovýchto framweorků jsou React, nebo Vue, Angular, Lit.
- Uspokojivně přiblížit všechny hlavní frameworky a jejich filozofii (každý je určený pro nějaký účel) není úplně jednoduché, proto se zmaříme na v současné době na nejrozšířenější React.

Zprovoznění

- Příště ukážeme, jak vytvářet pomocí Reactu samostatné aplikace, nyní si vystačíme s jednoduchou integrací do webové aplikace/stránky. V principu se takto React běžně používá, má to ale své limitace.
- Vložení knihovny z CDN. Knihovna má dvě základní komponenty, React a ReactDOM. Níže je ukázáno vložení vývojové verze.

```
<script src="https://unpkg.com/react@18/umd/react.development.js"></script>
<script src="https://unpkg.com/react-dom@18/umd/react dom.development.js"></script>
```

- Pro produkci je nutné použít react.production.min.js a react-dom.production.min.js.
- Pro ladění React knihovny existuje řada vývojářských nástrojů, ty ale vyžadují komunikaci skrze HTTP. Je tedy výhodnější (a v některých případech nezbytné) při vývoji používat webový server.

Nízkoúrovňový přístup

- Základem je metoda ReactDOM.createRoot(), která vytváří z vybraného elementu React root DOM element (technicky se zde realizuje propojení s Virtual a shadow DOM). Dodejme, že je ustálené, že React root element má atribut id nastaven na root.
- Pomocí metody React.createElement() vytváříme elementy, které následně React zobrazuje pomocí metody .render().

```
// nízkoúrovňový přístup
const root = ReactDOM.createRoot(document.getElementById('root'));

// React.createElement(element, properies, children)
const element = React.createElement("h1", {id: "element-1"}, "Hello world");

root.render(element);
```

React komponenta

- Stěžejním pojmem je React komponenta, které zapouzdřuje část webové aplikace/stránky a vytváří React elementy.

```
const SG1 = ["Jack O'neill", "Samantha Carter", "Daniel Jackson", "Teal'c"];

// komponenta
function TeamMembers(props) {
  return React.createElement("ul",
  {className: "team-members"},
  props.map((member, i) =>
    React.createElement("li", {key: i}, member)));
}

// poznámka: key je pro React, jinak warning
root.render(TeamMembers(SG1));
```

React komponenta

- Komponety je možné vytvářet i pomocí objektů.

```
class TeamMembers extends React.Component {
  constructor(props) {
    super(props); // mandatorni
    this.props = props;
  }

  render() {
    return React.createElement("ul",
      {className: "team-members"},
      this.props.members.map((member, i) =>
          React.createElement("li", {key: i}, member)));
    }
}

root.render(React.createElement(TeamMembers, {members: SG1}, null));
```

React komponenta

- Poznámka: ES6 syntaxi je možné se vyhnout. Dodejme, že mezi uvedenými přístupy je několik rozdílů jejichž pochopení vyžaduje velmi pokročilé znalosti JS.

- Z pohledu programování jsou komponenty pure funkce, které akceptují vlastnosti (props) a vrací výstup závislý na těchto vlastnostech, přičemž, vlastnosti jsou read only.

JSX

- Nízkúrovňový přístup hezky ukazuje přístup Reactu k práci s DOM. Pro složitější situace se ale nehodí (zbytečně komplikované). React využívá JSX (JavaScript and XML) zápis pro popis React elementů. Webové prohlížeče této syntaxi nerozumí a proto je třeba překlad do JS (do řeči React.createElement()).

- Překlad zajišťuje knihovna Babel. My si ji nyní integrujeme přímo v prohlížeči, ale takové řešení by nemělo být nikdo použito v produkci.

```
<script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
```

JSX

- Pro správné fungování musí mít element script atribut type="text/babel".

```
function TeamMembers(props) {
  return <ul>
    {props.map((member, i) =>
      <li key={i}>{member}</li>)}
      </ul>;
}

root.render(TeamMembers(SG1));
```

JSX

- JSX zapisuje komponenty jako nepárové elemementy a to vždy s /. Jelikož je klíčové slovo class vyhrazeno v JS, zapisuje se atribut class jako className. Výrazy JS se zapisují mezi { }.
- React při změně komponenty mění jen nutné části (to vede na obrovské zrychlení oproti práci s klasickým DOM). Například (kód je třeba pohlížet v nástrojích pro vývojáře).

```
const SG1new = ["Jack O'neill", "Samantha Carter", "Jonas Quinn", "Teal'c"];

setTimeout(() => root.render(<TeamMembers members={SG1new} />), 4000);
```

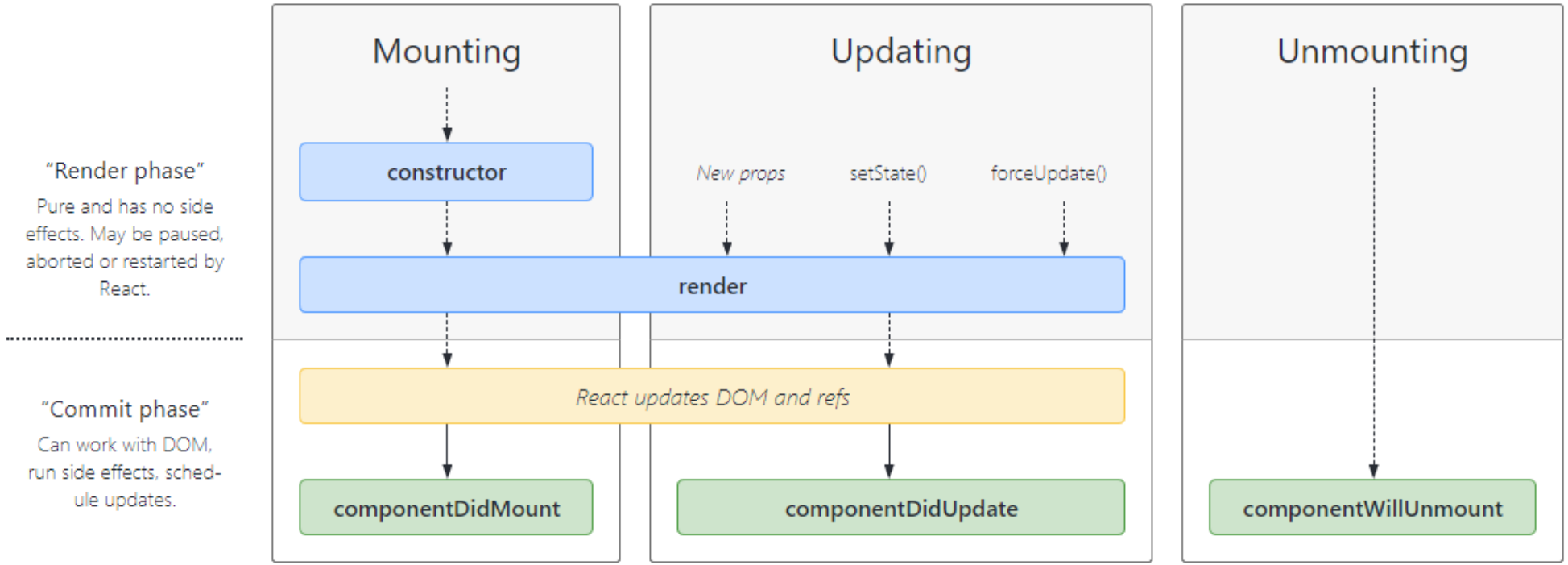
- Příklad komponenty složené z několika další.
- Pokud chceme aby byl výstup složen z více komponent, je třeba použít fragment, který se zkrácene v JSX zapisuje mezi <> </>

Stav komponenty

- Komponenty si mohou udržovat vlastní stav. Následující kód nelze použít při funkčním zápisu komponenty (je třeba použít React hook, to ukážeme příště).

```
class Clock extends React.Component {
  constructor(props) {
    super(props);
    this.state = {date: new Date()};
  }
  // komponenta je zobrazena v DOM
  componentDidMount() {
    this.timer = setInterval(() => this.tick(), 1000);
  }
  // komponenta je odstraněna z DOM
  componentWillUnmount() {
    clearInterval(this.timer);
  }
  // setState primarni funkce pro update DOM
  tick() {
    this.setState({date: new Date()});
  }
  render() {
    return (
      <time>It is {this.state.date.toLocaleTimeString()}.</time>
    );
  }
}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Clock />);
```

Životní cyklus komponenty



Zpracování událostí

- V JSX je mírně jiná syntaxe a chování při zpracování událostí. Výchozí události je třeba bránit explicitně.

```
function Form() {
  function handleSubmit(e) {
    e.preventDefault();
    console.log('You clicked submit.');
  }

  return (
    <form onSubmit={handleSubmit}>
      <button type="submit">Submit</button>
    </form>
  );
}
```

Zpracování událostí

- Největší problémy jsou způsobeny chováním this.

```
class Toggle extends React.Component {
    constructor(props) {
      super(props);
      this.state = {isToggleOn: true};

      // tento příkaz je klíčový (nahradí this ve funkce handleClick this
komponenty)
      this.handleClick = this.handleClick.bind(this);
    }
```

Zpracování událostí

```
  handleClick() {
    this.setState(prevState => ({
      isToggleOn: !prevState.isToggleOn
    }));
  }

  render() {
    return (
      <button onClick={this.handleClick}>
        {this.state.isToggleOn ? 'ON' : 'OFF'}
        </button>
    );
  }
}
```

- Lze řešit i přes arrow operátor, případně public instance fields (výchozí v Create React App, který ukážeme příště).

Vizualizace komponent

- Klasické CSS. V případě inline CSS je třeba využít {{ }} (CSS je JS objekt, vlasnosti nemají pomlčky v názvech) nebo lze CSS vytvářet dynamicky.

Úkoly

ÚKOL 1

- Vytvořte komponentu Panel, která bude obsahovat tlačítka. Příklad použití a vizualizace následuje. Snažte se o co nejuniverzálnější řešení.

```
const buttons = [
  {text: "add", color: "#eee"},
  {text: "edit", color: "#eee"},
  {text: "delete", color: "#FF5733", inverse: true}
]

const buttons2 = [
  {text: "a", color: "#34eb8f"},
  {text: "b", color: "#1f8753", inverse: true},
  {text: "c", color: "#34eb8f"},
  {text: "d", color: "#1f8753", inverse: true},
  {text: "e", color: "#34eb8f"}
]
```

Úkoly

```
const buttons3 = [
  {text: "info", color: "#349beb", inverse: true},
]

root.render(
  <>
  <Panel shadow buttons={buttons} />
  <Panel buttons={buttons} />
  <Panel buttons={buttons2} />
  <Panel buttons={buttons3} />
  <Panel shadow buttons={buttons3} />
  </>
);
```