Vue3 👏👏

波比小金刚

New Features

old	new
defineProperty	Proxy
Options API	Vue Composition API
--	Tree Shaking
--	Custom Renderer API
--	Fragment/Teleport/Suspense
flow	Better TS Support
--	Better Tpl Compile

...

框架

1、Vue Compostion API
2、LifeCycle
3、Fragment/Teleport
4、defineAsyncComponent/Suspense
5、ref
6、watchEffect/watch
7、emits
8、v-model
9、...

VCA vs React Hooks

1、Immutable & Mutable

```
// vue
const data = reactive({ count: 0 });
data.count++

// react
const [count, updateCount] = useState(0);
updateCount(count++)

// tips:
// 1. 没有谁好谁坏，各自有优势有劣势，mutable 更符合直觉
//    immutable 的稳定性在实现 Concurrent 上有优势
```

VCA vs React Hooks

2、vca 不用关注顺序

```
// react hooks 依赖顺序
// 这个代码是不能运行的，因为 react hooks 依赖执行的顺序
// 而条件语句会破坏这种顺序的追踪
if(some condition) {
  useEffect(() ⇒ { /* some logic */})
}

// vue - 没有这个问题！
```

VCA vs React Hooks

3、执行次数

```
// react
const Counter = () => {
  const [count, updateCount] = useState(0); // 每次渲染都会执行
  return  ...
}

// vue3
<template>
 //  ...
</template>
<script lang="ts">
  export default {
    setup() { // 只执行一次
      const data = reactive({
        count: 0,
        increase: () => {
          count++
        }
      });
      return { ...toRefs(data) }
    }
  }
</script>
```

VCA vs React Hooks

4、缓存 依赖？

```
// react 中为了减少不必要的re-run代码带来的开销，可以缓存

// 这很容易导致问题，比如你忘了还有依赖项
// 或者说去判断哪些依赖项要记得加上也是一个脑力劳动
const increase = useCallback(() => /* some logic */, []);
const counter = useMemo(() => /* some value */, []);

// vue3 没这些问题
```

VCA vs React Hooks

react hooks	vue3 vca
useState/useReducer	reactive() / ref()
setXXX()	data.xxx = yyy
useMemo	computed
useRef	ref
useEffect	life cycle
useContext	provider/inject
...	...

Suspense demo

```
// 不管是 react 还是 vue，本质都是在玩儿 promise
// react 是通过 throw 一个 promise
// vue 是 setup 返回一个 promise
```

任意门 demo

```
// 实现的原理

// 1. 当解析到 Teleport 的时候，调用 createBlock 方法，
//    给这个 vnode 打一个 tag （1 >> 6）表示它是任意门

// 2. render 的时候根据 tag 走不同的处理流程，典型的策略模式

// 3. Teleport.process

// 4. 本质就是将 Teleport 的 children 拿出来，挂到 to 下面去
```

emits

```
// 为了减少体积，事件的代码很多摘出去了

// 发
setup(props, context) {
  setTimeout(() => {
    context.emit("aaa", "allen");
  }, 1200);
  return {};
}

// 收
export default {
  emits: {
    aaa: val => {
      console.log(val);
      return true;
    }
  }
}
```

ref

主要处理基本类型

```
// 简单实现原理
function ref(val) {
  let __val = val

  return {
    get value() {
      return __val
    },

    set value(new_val) {
      if (__val !== new_val) {
        __val = new_val;
      }
    }
  }
}
```

提升/优化

1、Proxy
2、New Diff
3、New Tpl Compile(hoistStatic)
4、Tree Shaking Support
5、cacheHandlers
6、SSR
7、Better TS Support
8、...

why proxy??

Object.defineProperty

```
// defineProperty 需要指定 key 去拦截，需要增加类似 $set 这样的操作去 hack
// proxy 是对象的完全代理，不用考虑新增加的属性
// 并且 proxy 提供了多达十几种的属性操作

// proxy 是面向未来的规范标准，活跃维护
// defineProperty 就是老业务代码了，基本不会有大的变动了

// proxy 不用递归
export function reactive(target: object) {
  return createReactiveObject(target)
}

function createReactiveObject(target: Target) {
  const observed = new Proxy(
    target,
    baseHandlers // {get, set, deleteProperty}
  )
  return observed
}
```

Tree Shaking

```
// vue2
import Vue from 'vue'
Vue.nextTick()

// vue3
import { nextTick } from 'vue'
nextTick()

// tree-shaking 是编译器层面的功能，并不是 vue3 的功能
// vue3 只是做了一些迎合它的改进，减少体积

// 必须使用 esm 方式引入才行，但是挂在 default 的即使不用，也不会消除
// 所以尽量不要 export default
```

编译优化

```
// 这样的纯静态资源虽然不常有，diff 的时候跳过，是不是就优化了？
// 1. 跳过静态内容，只对比动态内容
<div>
  <section class="article_content">
    <h1>hello world!</h1>
  </section>
<div>

// 2. 有动态内容了，但是我们是不是可以切割开，静态的和动态的
// 如何切？？如何拿到整颗 vdom 树中动态节点？？
<div>
  <section class="article_content">
    <h1>hello world!</h1>
    <p>{{ dynamicContent }}</p>
  </section>
<div>

// 3. vdom ← 编译器
const vnode = {
  tag: 'div',
  // ...
  children: [
    { tag: 'h1', children: 'hello world!' },
    { tag: 'p', children: ctx.dynamicContent },  // 这是动态节点
  ]
}
```

```
// 5. 知道哪里是动态的了，就可以打一个标记是吧
// tag: 1 - 内容动态 2 - class 动态  ...
const vnode = {
  tag: 'div',
  //  ...
  children: [
    { tag: 'h1', children: 'hello world!' },
    { tag: 'p', children: ctx.dynamicContent, patchFlag: 1 },  // 这是动态节点
  ]
}

// 6. 一般是有粒度的嘛，比如 react 是 fiber 粒度
// vue3 里面有 Block，其实就是一个 vnode 节点，作为一个基本单元
// 可以把这个 Block 下面的所有动态节点抽出来，挂在某变量上
// 这样后续 diff 的时候，知道哪些变了，应该怎么处理，是不是就可以精确命中
// 且绕过一些无用工作
const vnode = {
  tag: 'div',
  //  ...
  children: [
    { tag: 'h1', children: 'hello world!' },
    { tag: 'p', children: ctx.dynamicContent, patchFlag: 1 },  // 这是动态节点
  ],
  dynamicChildren: [
    { tag: 'p', children: ctx.dynamicContent, patchFlag: 1 }
  ]
}
```

```
// 7. 没那么简单
// 思考？
<div>
  <section v-if="true">{{ a }}</section>
  <section v-else><div>{{ a }}</div></section>
</div>

// 单纯最外层的 Block(div)，收集到所有的子变化节点，但是碰到 v-if v-for
// 就不稳定了，因为 Block 没有变化，所以不会更新了。
// 实际上问题的本质在于 dynamicChildren 的 diff 是忽略 vdom 树层级的

// 8. 但是 section 如果也有一层 Block，然后 Block 可以收集 Block的话？
// vdom
{
  type: 'div',
  children: [ /*省略*/ ],
  dynamicChildren: [
    { type: 'section',  { key: 0 }, dynamicChildren: [ ... ]},
    { type: 'section',  { key: 1 }, dynamicChildren: [ ... ]}
  ]
}

// 就是 Block Tree
Block(div)
 -Block(section)
 -Block(section)
```

```
// 9. 适当的回退策略
// 当然，避免不了有一些结构，比如 Fragment 这种，diff 其 dynamicChildren 会得到错误结果
// 这种情况下就回退到 diff 其 chidlren 就好了。怎么判断？？

// 对比，编译器会打 tag STABLE_FRAGMENT，没这个就会退回去
<p v-for="item in list" /> // 不稳定的 dynamicChildren 结构
<p v-for="item in [1, 2, 3, 4]" /> // 稳定的 dynamicChildren 结构

// 10. render 开销的优化
// 本质上 render 函数就是层层执行 createVNode 之后再吐给 render 函数
<div>
  <p>text</p>
</div>

// 对应👇
function render() {
  return (openBlock(), createBlock('div', null, [
    createVNode('p', null, 'text') // 这个内容其实是静态的，每次render 都执行一次很浪费
  ]))
}
// 怎么办？还记得变量提升的面试题么？？？
const hoist1 = createVNode('p', null, 'text')

function render() {
  return (openBlock(), createBlock('div', null, [
    hoist1
  ]))
}
```

```
// 11. 除了提升树单位的结构外，还可以提升属性，也是可以优化性能的
<p title="123" aaa="bbb">{{ ccc }}</p>

// 👇
const hoistProp = { title: '123', aaa: 'bbb' }

render(ctx) {
    return (openBlock(), createBlock('div', null, [
        createVNode('p', hoistProp, ctx.ccc)
    ]))
}

// 12. 继续优化，如果大量列表，静态的哦，提升了之后会多很多内存占用，可以字符串化
<div>
  <p></p>
  <p></p>
  // ... 20 个 p 标签
  <p></p>
</div>

// 👇
const hoistStatic = createStaticVNode('<p></p><p></p><p></p> ... 20个 ... <p></p>')

render() {
  return (openBlock(), createBlock('div', null, [
     hoistStatic
  ]))
}
```

```
// 13. event cache
// react 之前一直有这个问题
<div onClick={() => {}}/>

// 这样每次都传了一个新的函数，也就是新的 props，肯定re-render啊
// vue3 可以开启 cache，本质上就是将这个函数缓存了一下
```

下一节课，实现一个简单的vue3

作业: 利用代理实现www.baidu.com这样访问返回页面

自由提问时间，不限本节课内容