勘误

dispatch(type, payload) { const action = this.actions[type]; const ctx = { commit: this.commit.bind(this), state: this.state, dispatch: this.dispatch.bind(this) } return action(ctx, payload); }

作业

购物车

发布

- 发布 npm run build
- 下载 nginx
- 配置 conf/nginx.conf

```
http {
    include       mime.types;
    default_type  application/octet-stream;

    #log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
    #                  '$status $body_bytes_sent "$http_referer" '
    #                  '"$http_user_agent" "$http_x_forwarded_for"';

    #access_log  logs/access.log  main;

    sendfile        on;
    #tcp_nopush     on;

    #keepalive_timeout  0;
    keepalive_timeout  65;

    #gzip  on;
    #前端设置了baseUrl的对应配置
    server {
        listen       80;
        server_name  localhost;

        #charset koi8-r;

        #access_log  logs/host.access.log  main;
        root  C:\Users\ytO37\Desktop\kaikeba\projects\vue-test\dist;  #单页面项目的打
包后的dist目录
        # http://localhost/kcart/detail/1
        # http://localhost/kcart/js/about.8187b66d.js
        location /kcart {
            try_files $uri /kcart/index.html;
        }

        #nginx反向代理，实现接口转发
        location ^~ /api/ {
```

```
            proxy_pass http://localhost:3000;  #注意路径后边不要加/
        }
    }

}
```

- 起服务器 `start nginx`

今日知识

kvue源码

```
// 期待用法
// new KVue({
//     data:{msg:'hello'}
// })

class KVue {
  constructor(options) {
    this.$options = options;

    //处理data选项
    this.$data = options.data;
    // 响应化
    this.observe(this.$data);

    // new Watcher();
    // this.$data.test;
    // new Watcher();
    // this.$data.foo.bar;

    new Compile(options.el, this);

    if (options.created) {
        options.created.call(this);
    }
  }

  observe(value) {
      if (!value || typeof value !== 'object') {
       return;
      }
      // 遍历对象
      Object.keys(value).forEach(key => {
          this.defineReactive(value, key, value[key])
          // 代理到vm上
          this.proxyData(key);
      })
  }
  proxyData(key) {
```

```
    Object.defineProperty(this, key, {
        get(){
            return this.$data[key];
        },
        set(newVal){
          this.$data[key] = newVal;
        }
    })
  }
  defineReactive(obj, key, val) {
      const dep = new Dep();

      Object.defineProperty(obj, key, {
          get(){
              // 将Dep.target添加到dep中
              Dep.target && dep.addDep(Dep.target)
              return val;
          },
          set(newVal){
            if (newVal !== val) {
                val = newVal;
                // console.log(`${key}更新了: ${newVal}`);
                dep.notify();
            }
          }
      })
      // 递归
      this.observe(val);
  }

}

class Dep {
    constructor(){
        this.deps = [];
    }

    addDep(dep) {
        this.deps.push(dep)
    }

    notify() {
        this.deps.forEach(dep => dep.update())
    }
}

class Watcher {
    constructor(vm, key, cb) {
        this.vm = vm;
        this.key = key;
        this.cb = cb;

        Dep.target = this;
```

```
        this.vm[this.key];// 添加watcher到dep
        Dep.target = null;
    }
    update() {
        // console.log('属性更新了');
        this.cb.call(this.vm, this.vm[this.key])
    }
}
```

compile源码

```
// new Compile(el, vm)

class Compile {
  constructor(el, vm) {
    this.$vm = vm;
    this.$el = document.querySelector(el);

    if (this.$el) {
      // 提取宿主中模板内容到Fragment标签，dom操作会提高效率
      this.$fragment = this.node2Fragment(this.$el);
      // 编译模板内容，同时进行依赖收集
      this.compile(this.$fragment);
      this.$el.appendChild(this.$fragment);
    }
  }

  node2Fragment(el) {
    const fragment = document.createDocumentFragment();
    let child;
    while ((child = el.firstChild)) {
      fragment.appendChild(child);
    }
    return fragment;
  }
  compile(el) {
    const childNodes = el.childNodes;

    Array.from(childNodes).forEach(node => {
      // 判断节点类型
      if (node.nodeType === 1) {
        // element节点
        // console.log('编译元素节点'+node.nodeName);
        this.compileElement(node);
      } else if (this.isInterpolation(node)) {
        // 插值表达式
        // console.log('编译插值文本'+node.textContent);
        this.compileText(node);
      }

      // 递归子节点
```

```
    if (node.childNodes && node.childNodes.length > 0) {
      this.compile(node);
    }
  });
}

isInterpolation(node) {
  // 是文本且符合{{}}
  return node.nodeType === 3 && /\{\{(.*)\}\}/.test(node.textContent);
}

compileElement(node) {
  // <div k-model="foo" k-text="test" @click="onClick">
  let nodeAttrs = node.attributes;
  Array.from(nodeAttrs).forEach(attr => {
    const attrName = attr.name;
    const exp = attr.value;
    if (this.isDirective(attrName)) {
      const dir = attrName.substring(2);
      this[dir] && this[dir](node, this.$vm, exp);
    }
    if (this.isEvent(attrName)) {
      const dir = attrName.substring(1);
      this.eventHandler(node, this.$vm, exp, dir);
    }
  });
}
isDirective(attr) {
  return attr.indexOf("k-") === 0;
}

isEvent(attr) {
  return attr.indexOf("@") === 0;
}
compileText(node) {
  console.log(RegExp.$1);

  this.update(node, this.$vm, RegExp.$1, "text");
}

update(node, vm, exp, dir) {
  let updatrFn = this[dir + "Updater"];
  updatrFn && updatrFn(node, vm[exp]);
  // 依赖收集
  new Watcher(vm, exp, function(value) {
    updatrFn && updatrFn(node, value);
  });
}
text(node, vm, exp) {
  this.update(node, vm, exp, "text");
}
textUpdater(node, val) {
  node.textContent = val;
```

```
  }

  eventHandler(node, vm, exp, dir) {
    const fn = vm.$options.methods && vm.$options.methods[exp];
    if (dir && fn) {
      node.addEventListener(dir, fn.bind(vm));
    }
  }

  html(node, vm, exp) {
    this.update(node, vm, exp, "html");
  }

  model(node, vm, exp) {
    // data -> view
    this.update(node, vm, exp, "model");
    // view -> data
    node.addEventListener("input", e => {
      vm[exp] = e.target.value;
    });
  }

  htmlUpdater(node, value) {
    node.innerHTML = value;
  }

  modelUpdater(node, value) {
    node.value = value;
  }
}
```

测试代码

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
</head>

<body>
    <div id="app">
        <p>{{name}}</p>
        <p k-text="name"></p>
        <p>{{age}}</p>
        <p>
            {{doubleAge}}
        </p>
        <input type="text" k-model="name">
```

```
        <button @click="changeName">呵呵</button>
        <div k-html="html"></div>
    </div>
    <script src='./kvue.js'></script>
    <script src='./compile.js'></script>

    <script>
        const kaikeba = new KVue({
            el: '#app',
            data: {
                name: "I am test.",
                age: 12,
                html: '<button>这是一个按钮</button>'
            },
            created() {
                console.log('开始啦')
                setTimeout(() => {
                    this.name = '我是测试'
                }, 1500)
            },
            methods: {
                changeName() {
                    this.name = '哈喽，开课吧'
                    this.age = 1
                }
            }
        })
    </script>
    <!-- <script src="./kvue.js"></script>
    <script>
        const app = new KVue({
            data: {
                test: 'kaikeba',
                foo: {bar:'bar'}
            }
        })

        app.$data.test = '我变了'
        app.$data.foo.bar = '我变了'
        app.test = '我又变了'
    </script> -->
</body>

</html>
```