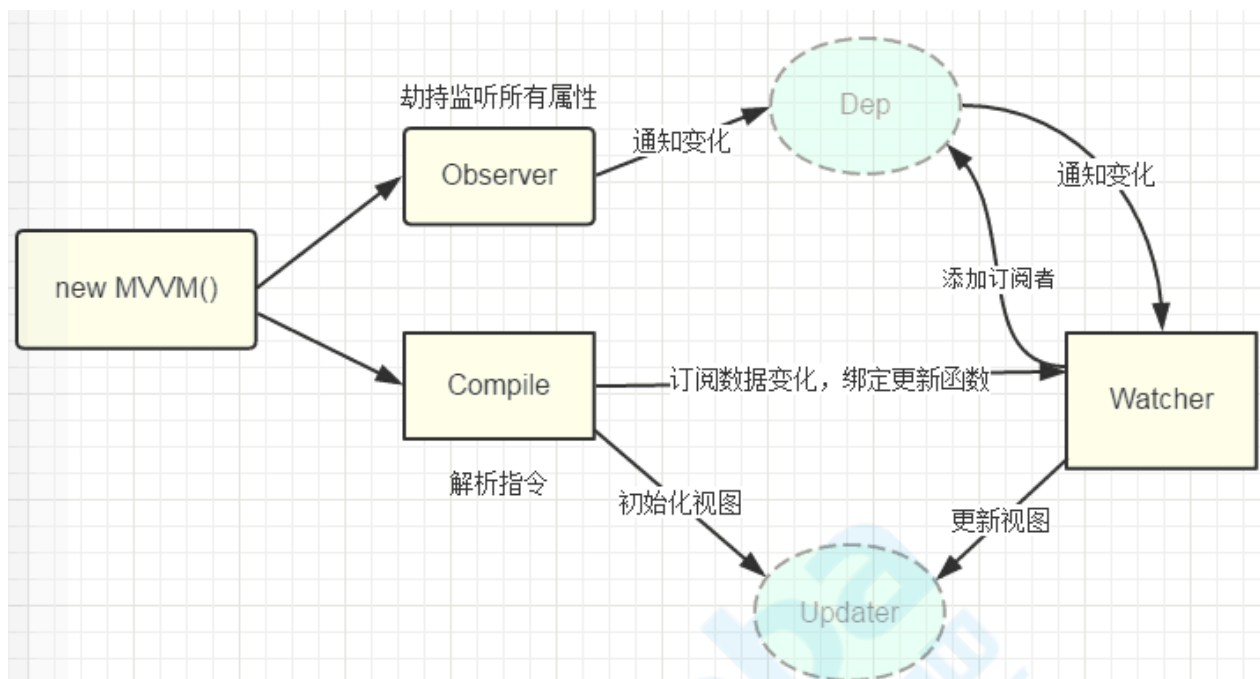


## Vue工作机制



### 初始化

在 `new Vue()` 时会调用 `_init()` 进行初始化，会初始化各种实例方法、全局方法、执行一些生命周期、初始化 `props`、`data` 等状态。其中最重要的是 `data` 的「响应化」处理。

初始化之后调用 `$mount` 挂载组件，主要执行编译和首次更新

### 编译

编译模块分为三个阶段

1. `parse`: 使用正则解析 `template` 中的 `vue` 的指令 (`v-xxx`) 变量等等 形成抽象语法树 `AST`
2. `optimize`: 标记一些静态节点，用作后面的性能优化，在 `diff` 的时候直接略过
3. `generate`: 把第一部生成的 `AST` 转化为渲染函数 `render function`

### 虚拟dom

`Virtual DOM` 是 `react` 首创，`Vue2` 开始支持，就是用 `JavaScript` 对象来描述 `dom` 结构，数据修改的时候，我们先修改虚拟 `dom` 中的数据，然后数组做 `diff`，最后再汇总所有的 `diff`，力求做最少的 `dom` 操作，毕竟 `js` 里对比很快，而真实的 `dom` 操作太慢

```
// vdom
{
  tag: 'div',
  props: {
    name: '开课吧',
    style: { color: red },
    onClick: xx
  }
}
```

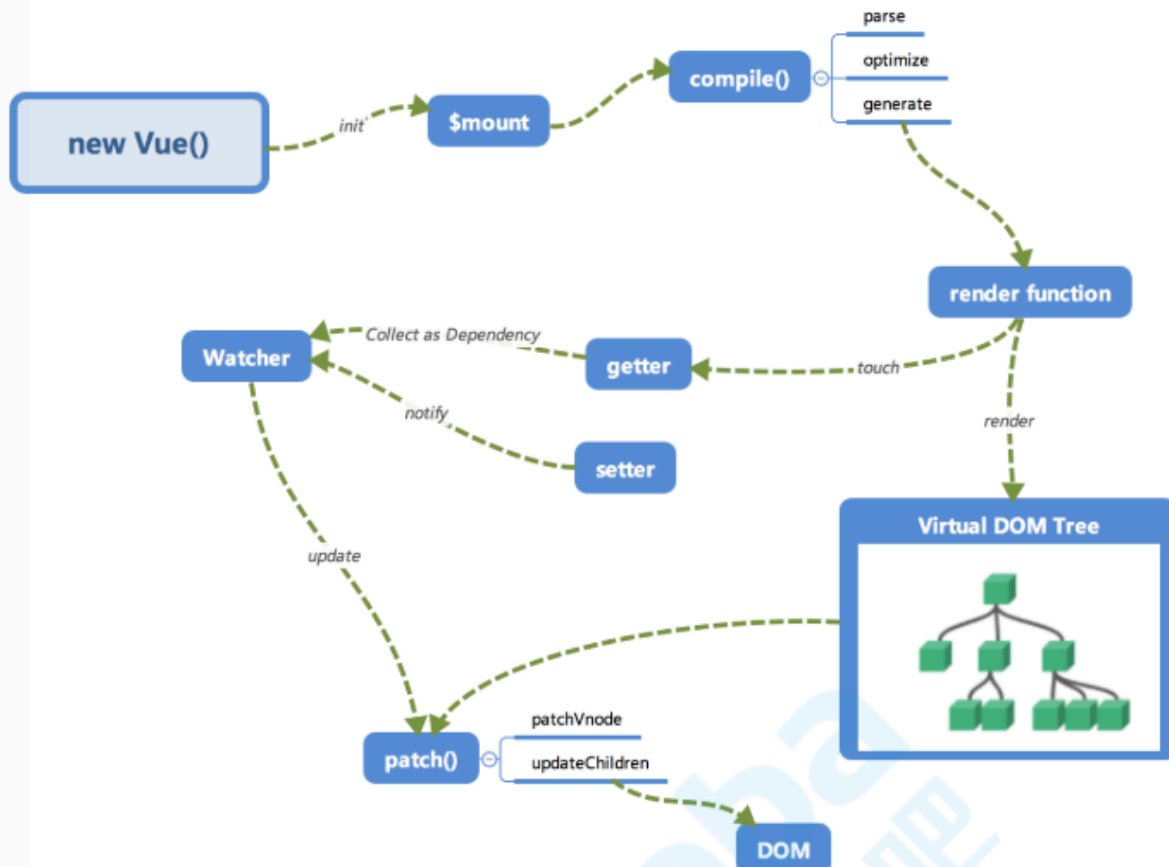
```
}
children: [
  {
    tag: 'a',
    text: 'click me'
  }
]
}
```

```
<div name="开课吧" style="color:red" @click="xx">
  <a>
    click me
  </a>
</div>
```

## 更新

数据修改触发setter，然后监听器会通知进行修改，通过对比新旧vdom树，得到最小修改，就是 patch，然后只需要把这些差异修改即可

## 实现kvue



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)
  }

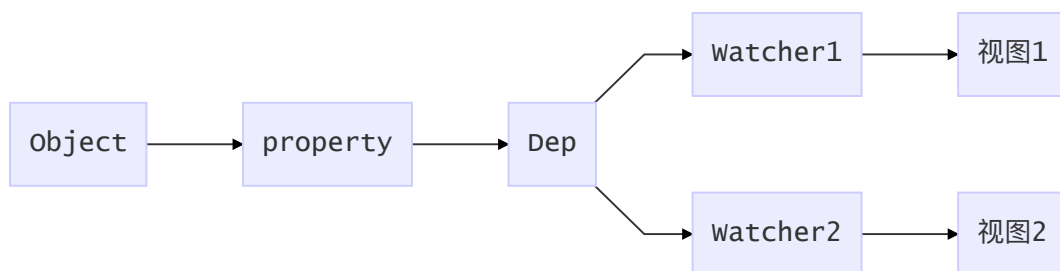
  notify() {
    this.deps.forEach(dep => dep.update())
  }
}

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])
  }
}

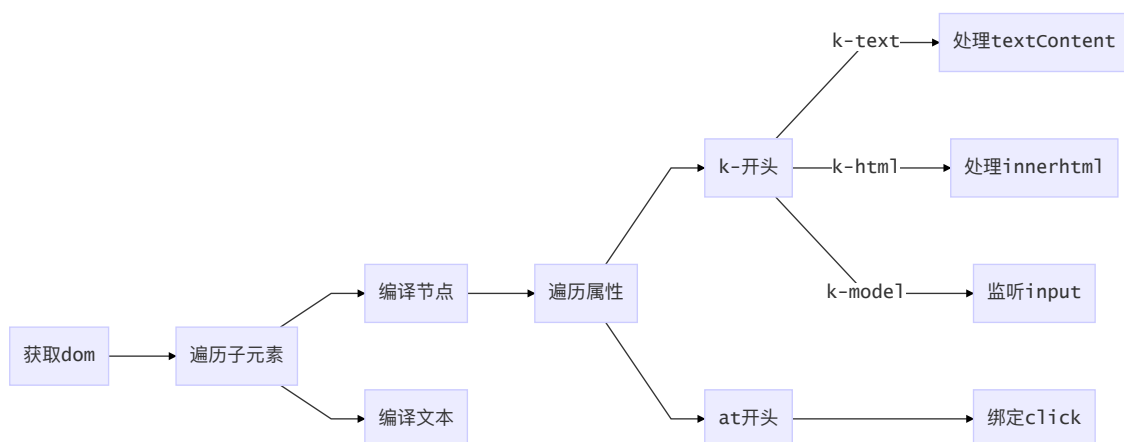
```

property/Dep/Watcher对应关系



compile源码

编译过程图



```
// new Compile(e1, vm)

class Compile {
  constructor(e1, vm) {
    this.$vm = vm;
    this.$e1 = document.querySelector(e1);

    if (this.$e1) {
      // 提取宿主中模板内容到Fragment标签，dom操作会提高效率
      this.$fragment = this.node2Fragment(this.$e1);
      // 编译模板内容，同时进行依赖收集
      this.compile(this.$fragment);
      this.$e1.appendChild(this.$fragment);
    }
  }

  node2Fragment(e1) {
    const fragment = document.createDocumentFragment();
    let child;
    while ((child = e1.firstChild)) {
      fragment.appendChild(child);
    }
    return fragment;
  }

  compile(e1) {
    const childNodes = e1.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;
  }
}

```

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>

```

## 作业

实现v-model，事件处理和v-html

## 拓展

数组响应化怎么做

## 检查点

- 试着说一下vue数据响应化过程
- 试着说一下双向绑定原理

