课程目标

深入浅出响应式

What? How?

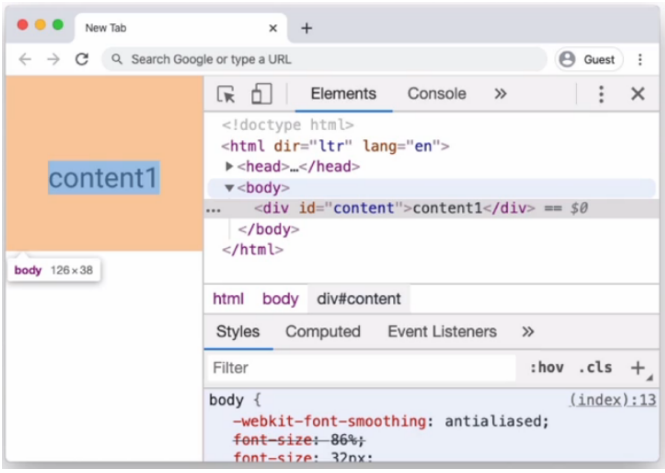
什么响应式?

```
let x;
let y;
let f = n => n * 100 + 100;

x = 1;
y = f(x);
console.log(y);      // 200

x = 2;
y = f(x);
console.log(y);      // 300

x = 3;
y = f(x);
console.log(y);      // 400
```

每个都有一个y映射 f(x)的值

```
let x;
let y;
let f = n => n * 100 + 100;
let onXChanged = cb => {...};

onXChanged(() => {
  y = f(x);
  console.log(y);
});

x = 1; // 200

x = 2; // 300

x = 3; // 400
```

通过监听x的变化，相应的变化对应的映射



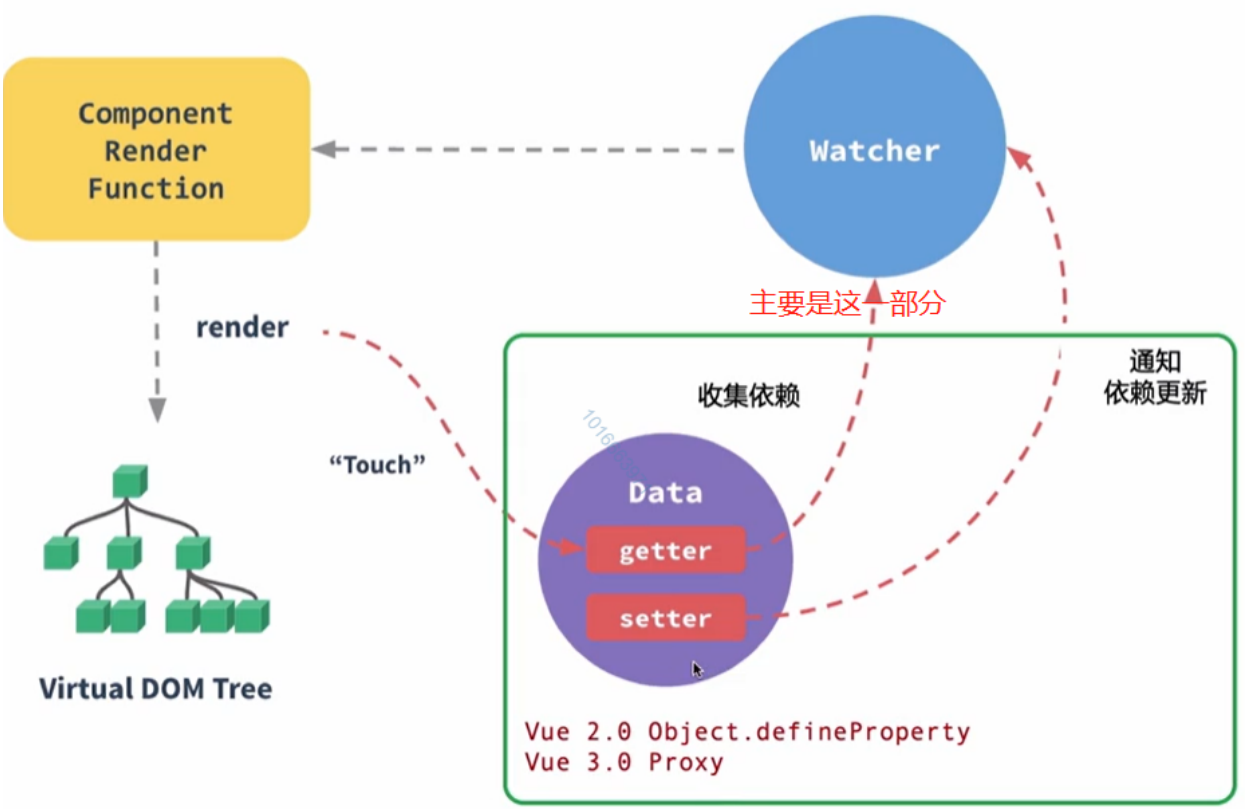

```
let x;
let y;
let f = n => n * 100 + 100;
let onXChanged = cb => {...};

onXChanged(() => {
  document.querySelector('#content')
    .innerText = `content${f(x)}`;
});

x = 1;

x = 2;

x = 3;
```

监听X的变化，一旦变化执行相应回调，重新渲染。

响应式是一种面向数据流和变化传播的编程方式.这意味着可以在编程当中很方便地表达静态或者动态的数据流，而相关的计算模型会自动将变化的值通过数据流进行传播。

vue的响应式具体是如何实现的呢?

首先需要监听数据变化，要知道响应式数据在何时更改了(Vue 2.0 是通过Object.definProperty将data中的属性都遍历一遍，并转化为get和set，并且在get函数中将使用到该数据的上下文进行一次收集，我们称之为依赖收集，相应的在set函数当中，当修改这个data数据的时候会通知依赖更新的操作。这里为什么需要一个依赖收集呢？那是因为一个data数据的修改，可能涉及模板中多处的变化，所以需要依赖收集，等依赖更新的时候进行一次批量的操作。vue3.0是通过Proxy对象进行代理，在代理里面进行依赖收集和更新依赖操作)



实操

未发封装版依赖收集类的版本

```
let x ;
let y ;
let f = n=>n*100+200;
//监听x变化的函数
```

```
let activeCallback;//定义当前的回调函数
let onXchanged = function(callback){
 activeCallback = callback
 activeCallback() //初始的调用调用
}


 //vue2.0 使用的是Object.definedProperty()创建对象 get获取访问属性 set修改属性监听，所以x搞一个引用数据类型
 let ref = (initValue)=>{
  let value = initValue
  return Object.defineProperty({},'value',{
  get() { //依赖收集
  return value
  },
  set(newValue) {
  value = newValue
  activeCallback() //当修X属性的时候，也需要进行相应的依赖更新
  }
  })
 }

 //创建一个对象，value的初始值为1
 x=ref(1)
  onXchanged(()=>{
  y =f(x.value)
  console.log(y)
 })

 x.value = 2 //300
 x.value = 3 //400
 x.value = 4 //500
```

封装依赖收集的类

```
//vue响应式源码核心代码实现,封装deps依赖收集的类
let x ;
let y ;
let f = n=>n*100+200;
let activeCallback;//定义当前的回调函数
//依赖收集的类
```

```
class Dep {
 constructor() {
 this.deps = new Set() //Set存在add方法
  }
  //收集依赖
  depend(){
  if(activeCallback){
  this.deps.add(activeCallback)
  }
  }
  //通知依赖更新
  notify(){
  this.deps.forEach(dep=>dep())
  }


}
//X监听函数
let onXchanged = function(callback){
 activeCallback = callback
 activeCallback() //初始的调用调用
 activeCallback = null // 销毁当前的activeCallback
}
// Object.defineProperty() 创建对象
let ref = (initValue)=>{
 let value = initValue
 let dep = new Dep() //实例化dep类
 return Object.defineProperty({},'value',{
 get() {
 //依赖收集
 dep.depend()
 return value
 },
 set(newValue) {
 value = newValue
 dep.notify()//当修X属性的时候，也需要进行相应的依赖更新
 }
 })
}
//初始化函数调用
```

```
x = ref(1)
// X监听函数初始化调用
onXchanged(()=>{
 y =f(x.value)
 console.log(y)
})
//修改x的值，触发ref对象中的set函数
x.value = 2
x.value = 3
```