# 动态数据运算

JavaScript表达式

简单的字符串拼接

Method方法

每一次渲染都会从新运算，造成大量的性能浪费

methods: {

totalPrice: function () {

var result = 0;

this.books.forEach(book => {

result += book.price \* book.num;

});

return result;

}

}

Watch属性

通过侦听目标数据的变化，当目标数据变化时，vue回调我们的监听事件

watch: {

books: {

// 监听到数据变化之后的回调方法

handler:function(val,oldval){

this.totalPrice = 0;

val.forEach(item => {

this.totalPrice += item.num \* item.price;

});

},

deep:true// 对象内部的属性监听，也叫深度监听

}

},

**watch 属性允许执行异步操作（访问网络、定时器等）**

watch: {

books: {

// 监听到数据变化之后的回调方法

handler:function(val,oldval){

this.totalPrice = '计算中...';

var $this = this;

setTimeout(() => {

$this.totalPrice = 0;

val.forEach(item => {

$this.totalPrice += item.num \* item.price;

});

}, 1000);

},

deep:true// 对象内部的属性监听，也叫深度监听

}

},

当watch需要监听的目标数据足够多时，就不适用了

Computed 计算属性

computed: {

totalPrice: function () {

var totalPrice = 0;

this.books.forEach(item => {

totalPrice += item.num \* item.price;

});

return totalPrice;

}

}

我们声明totalPrice方法，根据books计算价格，vue会自动来确认他们之间的依赖关系

Computed提供了一个getter方法和一个setter方法，默认，我们只需要关注getter方法，

computed: {

fullName: {

// getter

get: function () {

return this.firstName + ' ' + this.lastName

},

}

}

**重写setter方法**

computed: {

fullName: {

// getter

get: function () {

console.log('get方法');

return this.firstName + ' ' + this.lastName

},

// setter

set: function (newValue) {

console.log('set方法：' + newValue);

var names = newValue.split(' ')

this.firstName = names[0]

this.lastName = names[names.length - 1]

}

}

}

computed 属性默认提供了缓存机制，如果依赖数据并没有发生响应式变化，computed 属性不会重新计算，而只是读取缓存数据。我们可以通过cache: false,取消掉computed的缓存机制