Reacting to Data Changes with Filters, Computed Properties, and Watchers

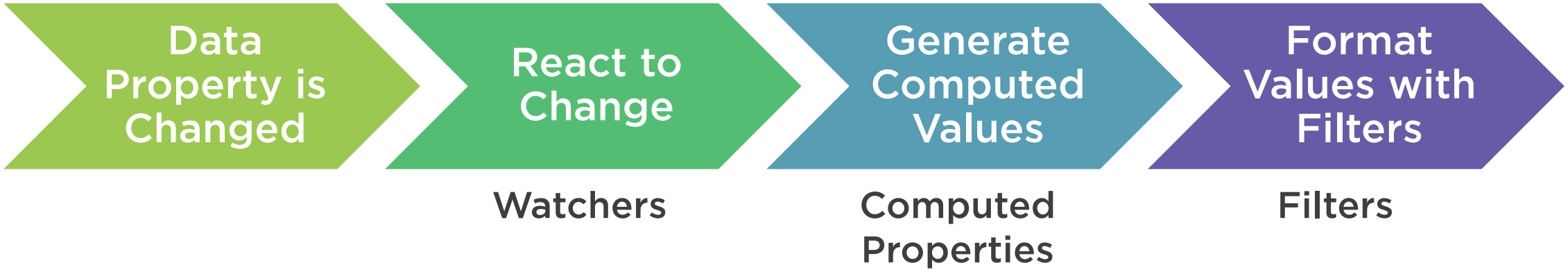
Chad Campbell
INDEPENDENT SOFTWARE CONSULTANT

@chadcampbell https://www.ecofic.com

Overview

- Watchers
- Computed properties
- Filters

Roadmap



Watchers, computed properties, and filters are optional

Monitoring Data Changes with Watchers

Watchers

Special functions that let you react to data property changes.

```
var growler = new Vue({
  el: '#growler’,
  data: {
    beers: [
      {name:'Ahool Ale',price:2.80},
      {name:'Agogwe Ale',price:2.38}
    ],
    shoppingCart: [],
    subTotal: 0.00
  },
  ...
});
```

```
<div id="growler">
  <table>
    <tr>
      <th>Beer</th>
      <th>Price</th>
      <th></th>
    </tr>

    <tr v-for="beer in beers">
      <td>{{ beer.name }}</td>
      <td>{{ beer.price }}</td>
      <td><button :click="buy(beer)">
      buy</button></td>
    </tr>

    <tr>
      <td>Subtotal</td>
      <td>{{ subTotal }}</td>
      <td></td>
    </tr>
  </table>
</div>
```

```
var growler = new Vue({
  ...,
  methods: {
    updateSubTotal: function() {
      var s = this.shoppingCart.length;
      var t = 0;
      for (var i=0; i<s; i++) {
        t += this.shoppingCart[i].price;
      }
      this.subTotal = t;
    },
    buy: function(beer) {
      this.shoppingCart.push(beer);
    }
  }
});
```

```
<div id="growler">
  <table>
    <tr>
      <th>Beer</th>
      <th>Price</th>
      <th></th>
    </tr>

    <tr v-for="beer in beers">
      <td>{{ beer.name }}</td>
      <td>{{ beer.price }}</td>
      <td><button :click="buy(beer)">
      buy</button></td>
    </tr>

    <tr>
      <td>Subtotal</td>
      <td>{{ subTotal }}</td>
      <td></td>
    </tr>
  </table>
</div>
```

```
var growler = new Vue({
  el: '#growler’,
  data: {
    beers: [...],
    shoppingCart: [],
    subTotal: 0.00
  },
  watch: {
    shoppingCart: function() {
      this.updateSubTotal();
    }
  },
  ...
});
```

```
<div id="growler">
  <table>
    <tr>
      <th>Beer</th>
      <th>Price</th>
      <th></th>
    </tr>

    <tr v-for="beer in beers">
      <td>{{ beer.name }}</td>
      <td>{{ beer.price }}</td>
      <td><button :click="buy(beer)">
      buy</button></td>
    </tr>

    <tr>
      <td>Subtotal</td>
      <td>{{ subTotal }}</td>
      <td></td>
    </tr>
  </table>
</div>
```

Watcher Definition

```
watch: {
  shoppingCart: function() {
    this.updateSubTotal();
  }
}
```

Defining a Watcher's Behavior

```
watch: {
  shoppingCart: 'updateSubTotal'
}
```

Defining a Watcher's Behavior

Calling a function by reference

```
watch: {
  subTotal: function (latest, original) {
    this.calculateSalesTax();
  }
}
```

Defining a Watcher's Behavior

Getting the before and after values

Vue doesn't keep data copies for Objects and Arrays

```
created: function() {
  this.$watch('shoppingCart.length', function(l, o) {
    growler.updateSubTotal();
  });
}
```

Watching Array Length Changes

Use of a simple dot-delimited path

Defining a Watcher's Depth

Watchers: By Default

Use a shallow monitoring approach

Compares values by reference

```
data: {
  shoppingCart: {
    items: [],
    subTotal: 0.00
  shoppingCart: []
  }
}
```

```
data: {
  shoppingCart: []
}
```

Expanding the Shopping Cart

Adding multiple properties

Updating the Watcher

```
watch: {
  shoppingCart: {
    handler: function(latest, original) {
      this.updateSubTotal()
    },
    deep: true
  }
```

Turn deep monitoring on only when it makes sense

Faster Rendering with Computed Properties

Computed Property

Functions whose results are cached until their depending values change.

Result will only be re-generated if a depending value changes

Initializing Computed Properties

```
var growler = new Vue({
 el: '#growler',
 data: { canConnect: false },
 computed: {
  isOnline: function() {
   return this.canConnect ? 'Yes':'No';
  }
 },
 created: function() {
  axios.get('https://www.ecofic.com')
   .then(function (res) {
     growler.canConnect = true;
   })
   .catch(function (err) {
     growler.canConnect = false;
   });
 }
});
```

```
<div id=“growler">
  <div>
    Online? {{ isOnline }}
  </div>
</div>
```

Computed Properties

Great when you want to cache values

“Getter” by default

Using Computed Properties as Accessors

Accessor

Computed properties that can get or set property values

Adding a get Function

Without get

```
computed: {
  isOnline: function() {
    return this.canConnect ?
      'Yes' : 'No';
  }
}
```

With get

```
computed: {
  isOnline: {
    get: function () {
      return this.canConnect ?
        'Yes' : 'No';
    }
  }
}
```

```
isOnline: {
  get: function () { return this.canConnect ? 'Yes' : 'No'; },
  set: function(newValue) {
    this.canConnect = newValue;
  }
}
```

Adding a set

Making an accessor a setter

A setter is a great way to impose validation on values

Formatting with Filters

International Beer Unit Data

Returned	Converted
33 i.b.u.	33 IBU
28 i.b.u.	28 IBU
31 i.b.u.	31 IBU

Filters

Special type of function

Use for common text conversions

```
var growler = new Vue({
  el: '#growler',
  data: { results: [
    {name:'Ahool Ale',ibu:'33 i.b.u.'}
  ]},
  filters: {
    convertIBU: function(value) {
      if (!value) { return ''; }
      value = value.toString();
      value = value.replace(/\./g, '');
      return value.toUpperCase();
    }
}});
```

```
<div v-for="result in results">
  <div>{{ result.name }}</div>
  <small class="text-muted">
    {{ result.ibu | convertIBU }}
  </small>
</div>
```

Filters cannot be used with the v-html or v-text directives

```
var growler = new Vue({
  el: '#growler',
  data: { results: [
    {name:'Ahool Ale',ibu:'33 i.b.u.'}
  ]},
  filters: {
    convertIBU: function(value, empty) {
      if (!value) { return empty; }
      value = value.toString();
      value = value.replace(/\./g, '');
      return value.toUpperCase();
    }
}});
```

```
<div v-for="result in results">
  <div>{{ result.name }}</div>
  <small class="text-muted">
    {{ result.ibu | convertIBU('--‘) }}
  </small>
</div>
```

The value is always the first parameter passed to a filter.

Filters available via the `$options` property

Programmatically Calling a Filter

```
convertIBU: function(value, empty) {
  if (!value) { return empty; }
  value = value.toString();

  if (this.growler) {
    value = this.growler.$options
      .filters.removePeriods(value);
    value = this.growler.$options
      .filters.toUpperCase(value);
  }
  return value;
},
```

```
removePeriods: function(value) {
  return value.replace(/\./g, '');
},

toUpperCase: function(value) {
  return value.toUpperCase();
}
```

Filters are added to the `$options` after instantiation.

Chained via a pipe (|)

Chaining Filters Together

```
filters: {
  convertIBU: function(value, empty) {
    if (!value) { return empty; }
    return value.toString();
  },

  removePeriods: function(value) {
    return value.replace(/\./g, '');
  },

  toUpperCase: function(value) {
    return value.toUpperCase();
  }
}
```

```
<div v-for="result in results">
  <div>{{ result.name }}</div>
  <small>{{ result.ibu |
    convertIBU('--') |
    removePeriods |
    toUpperCase }}</small>
</div>
```

Comparing Filters to Methods

Filters Should:

Only take in a value and return a new value

Not change the value of any properties in a view

Filters Are Easier to Read

Using Methods

```
<div>{{ toUpperCase(removePeriods(result.ibu)) }}</div>
```

Using Filters

```
<div>{{ result.ibu | removePeriods | toUpperCase }}</div>
```

In General

Filters are intended to be used across views

Methods are specific to an instance

In Review

Filters are great a way to handle basic text transformations

Computed properties are for more complex transformations

Watchers are for asynchronous operations

Comparing Filters, Computed Properties, Methods, and Watchers

	Filters	Computed Properties	Methods	Watchers
React to data changes	Yes	Yes	No	Yes
Reusable across apps	Yes	No	No	No
Cached based on dependencies	No	Yes	No	No
Appropriate for async operations	No	No	Yes	Yes
Accepts Parameters	Yes	No	Yes	No
Fires after creation	Yes	Yes	Possible	No

Summary

Watchers

Computed properties

Filters

Follow-up Topics

- Axios
- Components
- Transitions
- Routing
- State Management
- Server-Side Rendering