Improving your App with Filters and Mixins

Some cases may be helpful. Help structure your app and output.

Local Filter

A filter is a syntax used in template that helps transform output. All the filters have to be implemented. For example, make a certain text to be upper case.

```
\\ App.vue
<script>
...
  data() {
    return {
      text: 'Hello World'
    }
  },
  filters: {
    'to-uppercase'(value) {
      return value.toUpperCase();
    },
    toUppercase(value) {

    }
  }
...
</script>

<template>
...
  <!-- apply filter -->
  <p> {{ text | toUppercase }} </p>  <!-- similar to
```

```
Angular 2 -->
...
</template>
```

Global Filters and How to chain multiple Filters

Using **Vue.filter()** to register a filter globally.

```
\\ main.js
...
Vue.filter('to-lowercase', function(value) {
  return value.toLowerCase();
});
...
```

```
\\ App.vue
<script>
...
  data() {
    return {
      text: 'Hello World'
    }
  },
  filters: {
    'to-uppercase'(value) {
      return value.toUpperCase();
    },
    toUppercase(value) {

    }
  }
...
</script>
```

```
<template>
...
  <!-- chaining filter: text => all uppercase => all
lowercase -->
  <p> {{ text | toUppercase | to-lowercase }} </p>
<!-- similar to Angular 2 -->
...
</template>
```

Computed Properties

Sometimes, a computed property is better than using filter.

```
\\ App.vue
<script>
...
  data() {
    return {
      text: 'Hello World',
      fruits: ['Apple', 'Banana', 'Mango', 'Melon']
    }
  },
  filters: {
    'to-uppercase'(value) {
      return value.toUpperCase();
    }
  },
  computed: {
    // only calculate when fruits is change
    filteredFruits() {
      return this.fruits.filter((element) => {
        return element.match(this.filterText);
      });
    }
  }
...
</script>
```

```
<template>
...
  <!-- chaining filter: text => all uppercase => all
lowercase -->
  <input v-model="filterText">
  <ul>
    <li v-for="fruit in filteredFruits "> {{ fruit }}
</li>
  </ul>
...
</template>
```

Mixins

Consider the following code duplication. Both the **parent** (App.vue) and the **child** have the **same** data and computed properties. How can we avoid such code duplication? For that we can use **Mixins**.

By outsource the common code to another javascript (.js) file, we created a mixin.

```
// fruitMixin.js - export a const javascript object
export const fruitMixin = {
  data() {
    return {
      fruits: ['Apple', 'Banana', 'Mango', 'Melon']
    }
  },
  computed: {
    filteredFruits() {
      return this.fruits.filter((element) => {
        return element.match(this.filterText);
      });
    }
  },
```

```
}
```

The code in the **List.vue** and **App.vue** would be:

```
// List.vue
<script>
import { fruitMixin } from './fruitMixin';

export default {
  mixins: [fruitMixin]
}
</script>
```

```
// App.vue
<script>
import List from './List.vue';
import { fruitMixin } from './fruitMixin';

export default {
  mixins: [fruitMixin],
  data() {
    return {
      text: 'Hello World',
    }
  },
  filters: {
    toUppercase(value) {
      return value.toUpperCase();
    }
  }
  components: {
    appList: List   // <app-list>
  }
}
```

```
</script>
```

Once using **Mixin**, there is no need to explicitly add the code for both **fruits** and **filteredFruits** in both parent and child component.

Mixins Merged

created() hook in mixin is always executed before **created()** in the components.

Global Mixin

If we want to have a mixin that can be used in all components, we can use global mixin by registering with **Vue.mixin()** in **main.js**. It is not recommended to use in the production environment.

```
// main.js
...
Vue.mixin( {
  created() {
    console.log('Global mixin - created hook');
  }
});
...
```

The global mixin will be added to all components automatically. The global **created()** hook is called first then local mixin's and then component's.

Mixins and Scope

What happen if we change the mixin's data property. The mixin data is not shared but replicated for every Vue's instance.

```
// App.vue
<script>
import List from './List.vue';
import { fruitMixin } from './fruitMixin';

export default {
  mixins: [fruitMixin], // import as a mixin
  data() {
    return {
      text: 'Hello World',
    }
  },
  filters: {
    toUppercase(value) {
      return value.toUpperCase();
    }
  }
  components: {
    appList: List   // <app-list>
  }
}
</script>

<template>
...
  <!-- adding a new element to mixin's data -->
  <button @click="fruits.push('Berry')>Add new
item<button>
  <input v-model="filterText">
  <ul>
    <li v-for="fruit in filteredFruits "> {{ fruit }}
</li>
  </ul>
  <hr>

  <!-- add List.vue component -->
  <app-list></app-list>
...
</template>
```