Drawer

Sometimes, `Dialog` does not always satisfy our requirements, let's say you have a massive form, or you need space to display something like `terms & conditions`, `Drawer` has almost identical API with `Dialog`, but it introduces different user experience.

Basic Usage

Callout a temporary drawer, from multiple direction

:::demo You must set `visible` for `Drawer` like `Dialog` does to control the visibility of `Drawer` itself, it's `boolean` type. `Drawer` has to parts: `title` & `body`, the `title` is a named slot, you can also set the title through attribute named `title`, default to an empty string, the `body` part is the main area of `Drawer`, which contains user defined content. When opening, `Drawer` expand itself from the **right corner to left** which size is **30%** of the browser window by default. You can change that default behavior by setting `direction` and `size` attribute. This show case also demonstrated how to use the `before-close` API, check the Attribute section for more detail

```
<el-radio-group v-model="direction">
  <el-radio label="ltr">left to right</el-radio>
  <el-radio label="rtl">right to left</el-radio>
  <el-radio label="ttb">top to bottom</el-radio>
  <el-radio label="btt">bottom to top</el-radio>
</el-radio-group>

<el-button @click="drawer = true" type="primary" style="margin-left: 16px;">
  open
</el-button>

<el-drawer
  title="I am the title"
  :visible.sync="drawer"
  :direction="direction"
  :before-close="handleClose">
  <span>Hi, there!</span>
</el-drawer>

<script>
  export default {
    data() {
      return {
        drawer: false,
        direction: 'rtl',
      };
    },
    methods: {
      handleClose(done) {
        this.$confirm('Are you sure you want to close this?')
          .then(_ => {
            done();
          })
          .catch(_ => {});
      }
    }
  };
</script>
```

:::

No Title

When you no longer need a title, you can remove title from drawer.

:::demo Set the `withHeader` attribute to **false**, you can remove the title from drawer, thus your drawer can have more space on screen. If you want to be accessible, make sure to set the `title` attribute.

```
<el-button @click="drawer = true" type="primary" style="margin-left: 16px;">
  open
</el-button>

<el-drawer
  title="I am the title"
  :visible.sync="drawer"
  :with-header="false">
  <span>Hi there!</span>
</el-drawer>

<script>
  export default {
    data() {
      return {
        drawer: false,
      };
    }
  };
</script>
```

:::

Customization Content

Like `Dialog`, `Drawer` can do many diverse interaction as you wanted.

:::demo

```
<el-button type="text" @click="table = true">Open Drawer with nested table</el-
button>
<el-button type="text" @click="dialog = true">Open Drawer with nested form</el-
button>
<el-drawer
  title="I have a nested table inside!"
  :visible.sync="table"
  direction="rtl"
  size="50%">
   <el-table :data="gridData">
      <el-table-column property="date" label="Date" width="150"></el-table-column>
      <el-table-column property="name" label="Name" width="200"></el-table-column>
      <el-table-column property="address" label="Address"></el-table-column>
    </el-table>
</el-drawer>

<el-drawer
  title="I have a nested form inside!"
  :before-close="handleClose"
  :visible.sync="dialog"
  direction="ltr"
  custom-class="demo-drawer"
  ref="drawer"
  >
  <div class="demo-drawer__content">
    <el-form :model="form">
      <el-form-item label="Name" :label-width="formLabelWidth">
        <el-input v-model="form.name" autocomplete="off"></el-input>
      </el-form-item>
      <el-form-item label="Area" :label-width="formLabelWidth">
        <el-select v-model="form.region" placeholder="Please select activity area">
          <el-option label="Area1" value="shanghai"></el-option>
          <el-option label="Area2" value="beijing"></el-option>
        </el-select>
      </el-form-item>
    </el-form>
    <div class="demo-drawer__footer">
      <el-button @click="cancelForm">Cancel</el-button>
      <el-button type="primary" @click="$refs.drawer.closeDrawer()"
:loading="loading">{{ loading ? 'Submitting ...' : 'Submit' }}</el-button>
    </div>
  </div>
</el-drawer>

<script>
export default {
  data() {
    return {
      table: false,
      dialog: false,
      loading: false,
      gridData: [{
        date: '2016-05-02',
        name: 'Peter Parker',
        address: 'Queens, New York City'
      }, {
        date: '2016-05-04',
        name: 'Peter Parker',
        address: 'Queens, New York City'
      }, {
        date: '2016-05-01',
        name: 'Peter Parker',
        address: 'Queens, New York City'
      }, {
        date: '2016-05-03',
        name: 'Peter Parker',
        address: 'Queens, New York City'
      }],
      form: {
        name: '',
        region: '',
        date1: '',
        date2: '',
        delivery: false,
        type: [],
        resource: '',
        desc: ''
      },
      formLabelWidth: '80px',
      timer: null,
    };
  },
  methods: {
    handleClose(done) {
      if (this.loading) {
        return;
      }
      this.$confirm('Do you want to submit?')
        .then(_ => {
          this.loading = true;
          this.timer = setTimeout(() => {
            done();
            // animation takes time
            setTimeout(() => {
              this.loading = false;
            }, 400);
          }, 2000);
        })
        .catch(_ => {});
    },
    cancelForm() {
      this.loading = false;
      this.dialog = false;
      clearTimeout(this.timer);
    }
  }
}
</script>
```

:::

Nested Drawer

You can also have multiple layer of `Drawer` just like `Dialog`. :::demo If you need multiple Drawer in different layer, you must set the `append-to-body` attribute to **true**

```
<el-button @click="drawer = true" type="primary" style="margin-left: 16px;">
  open
</el-button>

<el-drawer
  title="I'm outer Drawer"
  :visible.sync="drawer"
  size="50%">
  <div>
   <el-button @click="innerDrawer = true">Click me!</el-button>
   <el-drawer
     title="I'm inner Drawer"
     :append-to-body="true"
     :before-close="handleClose"
     :visible.sync="innerDrawer">
     <p>_(:з」∠)_</p>
   </el-drawer>
  </div>
</el-drawer>

<script>
  export default {
    data() {
      return {
        drawer: false,
        innerDrawer: false,
      };
    },
    methods: {
      handleClose(done) {
        this.$confirm('You still have unsaved data, proceed?')
          .then(_ => {
            done();
          })
          .catch(_ => {});
      }
    }
  };
</script>
```

:::

:::tip

The content inside Drawer should be lazy rendered, which means that the content inside Drawer will not impact the initial render performance, therefore any DOM operation should be performed through `ref` or after `open` event emitted.

:::

:::tip

Drawer provides an API called `destroyOnClose`, which is a flag variable that indicates should destroy the children content inside Drawer after Drawer was closed. You can use this API when you need your `mounted` life cycle to be called every time the Drawer opens.

:::

:::tip

If the variable bound to `visible` is managed in Vuex store, the `.sync` can not work properly. In this case, please remove the `.sync` modifier, listen to `open` and `close` events of Dialog, and commit Vuex mutations to update the value of that variable in the event handlers.

:::

Drawer Attributes

Parameter	Description	Type	Acceptable Values	Defaults
append-to-body	Controls should Drawer be inserted to DocumentBody Element, nested Drawer must assign this param to **true**	boolean	—	false
before-close	If set, closing procedure will be halted	function(done), done is function type that accepts a boolean as parameter, calling done with true or without parameter will abort the close procedure	—	—
close-on-press-escape	Indicates whether Drawer can be closed by pressing ESC	boolean	—	true
custom-class	Extra class names for Drawer	string	—	—
destroy-on-close	Indicates whether children should be destroyed after Drawer closed	boolean	-	false
modal	Should show shadowing layer	boolean	—	true
modal-append-to-body	Indicates should shadowing layer be insert into DocumentBody element	boolean	—	true
direction	Drawer's opening direction	Direction	rtl / ltr / ttb / btt	rtl

show-close	Should show close button at the top right of Drawer	boolean	—	true
size	Drawer's size, if Drawer is horizontal mode, it effects the width property, otherwise it effects the height property, when size is `number` type, it describes the size by unit of pixels; when size is `string` type, it should be used with `x%` notation, other wise it will be interpreted to pixel unit	number / string	-	'30%'
title	Drawer's title, can also be set by named slot, detailed descriptions can be found in the slot form	string	—	—
visible	Should Drawer be displayed, also support the `.sync` notation	boolean	—	false
wrapperClosable	Indicates whether user can close Drawer by clicking the shadowing layer.	boolean	-	true
withHeader	Flag that controls the header section's existance, default to true, when withHeader set to false, both `title attribute` and `title slot` won't work	boolean	-	true

Drawer Slot

Name	Description
—	Drawer's Content
title	Drawer Title Section

Drawer Methods

Name	Description
closeDrawer	In order to close Drawer, this method will call `before-close`.

Drawer Events

Event Name	Description	Parameter
open	Triggered before Drawer opening animation begins	—
opened	Triggered after Drawer opening animation ended	—
close	Triggered before Drawer closing animation begins	—

closed	Triggered after Drawer closing animation ended	—