Создание простого приложения

- Несколько экземпляров приложения
- Массивы и объекты в свойстве data
- Массивы и объекты в реальных приложениях
- Хуки жизненного цикла
- Описание хуков
- Первое приложение

Несколько экземпляров приложения

Прежде чем привести пример создания нескольких экземпляров приложений – пару слов о базах данных. Именно из **баз данных**, в подавляющем количестве случаев, мы будем получать информацию для наших приложений.

База данных хранит информацию о некоторых **взаимосвязанных** сущностях.

Сущность (*англ*. entity) может представлять собой нечто **вещественное** (дом, человек, предмет) или **абстрактное** (банковская операция, отдел компании, маршрут автобуса). Таким образом, сущность представляет **тип хранимых данных**, которые, как правило, соответствуют **объектам реального мира**.

В физической модели сущность называется **таблицей**, каждая таблица должна представлять **одну сущность**. Сущности состоят из **атрибутов** (столбцов таблицы) и **записей** (строк в таблице).

Обычно базы данных состоят из нескольких **главных** сущностей, **связанных** с большим количеством **подчиненных** сущностей.

Пример.

Вам необходимо создать базу данных "Домашняя библиотека". Проанализировав предметную область, вы выделяете две **сущности** и создаете для хранения данных об объектах сущностей две **таблицы**:

- **Автор**

- **Книга**

Сущность не всегда равно таблица, но это уже дальше в теорию баз данных. Каждая таблица содержит **записи об объектах реального мира**, в нашем случае об авторах книг и самих книгах.

Записи (объекты) таблицы **Автор**:

- Пушкин А.С.
- Довлатов С.Д.
- Булгаков М.А.
- и т.д.

Записи (объекты) таблицы **Книга:**

- Мастер и Маргарита
- Зона
- Евгений Онегин
- и т.д.

Таблицы (и представленные в них сущности) проектируемой базы данных состоят между собой в некоторых отношениях: у одного автора много книг. Такой тип отношения называется **один-ко-многим**. В этом случае говорят – таблица Автор является **главной** по отношению к таблице Книга. Таблица Книга – **подчиненная**.

> **Примечание.** Подробнее о базах данных в курсе: **Базы данных с нуля. Теория и практика на примере СУБД MS Access**.

Таких примеров отношений быть очень много: Класс – Ученик, Музыкальная группа – Альбом, Альбом – Трек и т.д.

Таким образом, при выводе взаимосвязанных данных может быть удобным разместить информацию о **каждой сущности в отдельный экземпляр приложения**.

```
// создание 1-го экземпляра приложения
const app1 = Vue.createApp({
  /* ... */
```

```
});
vm1 = app1.mount('#container-1'); // монтирование 1-го шаблона
// создание 2-го экземпляра приложения
const app2 = Vue.createApp({
  /* ... */
});
vm2 = app2.mount('#container-2'); // монтирование 2-го шаблона
```

example_1. Под каждую сущность свой экземпляр приложения

```
<!--- шаблон для вывода группы -->
<div id="team">
     <h2>Информация о группе</h2>
     <h3> {{ name }} </h3>
     Страна: {{ country }} <br />
     Город: {{ city }} <br />
     Дата образования: {{ date }} <br />
     Стиль исполнения: {{ style }}
     </p>
</div>
<!--- шаблон для вывода альбома -->
<div id="album">
     <h3>Альбом</h3>
     <b>{{ name }} </b><br />
     Дата выхода: {{ date}} <br />
     Лейбл: {{ label }} <br />
     Формат: {{ format }} <br />
     Статус: {{ status }}
</div>
<script>
```

```
    // экземпляр для данных о группе
    var vmTeam = Vue.createApp({
        data() {
            return {
                name: "Pink Floyd",
                country: "Великобритания",
                city: "Лондон",
                date: "1965",
                style: "Прогрессивный рок, арт-рок,
                психоделический рок"
            }
        }
    }).mount("#team");
    // экземпляр для данных об альбоме
    var vmAlbum = Vue.createApp({
        data() {
            return {
                id: "4",
                name: "The Dark Side of The Moon",
                date: "17 марта 1973",
                label: "Harvest, Capitol, EMI",
                format: "LP, кассета, CD, SACD",
                status: "Платиновый (USA), Платиновый (GBR),
                Бриллиантовый (CAN)"
            }
        }
    }).mount("#album");
</script>
```

Массивы и объекты в свойстве data

Данные свойства **data** редко будут представлены простыми переменными. Как правило реальные приложения оперируют большими наборами данных – **массивами** и **объектами**.

Вывод содержимого массивов и объектов во Vue осуществляется так же, как и в чистом JavaScript. Посмотрим на примерах.

Пусть в опции **data** у нас есть свойство **pro**, которое содержит внутри себя некоторый **массив**:

```
var vm = Vue.createApp({
     data() {
          return {
          pro: ["PHP", "JavaScript", "Wordpress", "Bootstrap"]
          }
     }
}).mount("#log");
```

Выведем элементы массива в шаблон приложения и в отладочную консоль.

example_2. Массивы в свойстве data

```
<div id="log">
     <h2>Наша группа IT pechora-PRO:</h2>
     <ul>
          <li>{{ pro[0] }}</li>
          <li>{{ pro[1] }}</li>
          <li>{{ pro[2] }}</li>
          <li>{{ pro[3] }}</li>
          <li> и многое другое</li>
     </ul>
```

```
</div>
<script>
     var vm = Vue.createApp({
          data() {
               return {
                    pro: ["PHP", "JavaScript", "Wordpress",
                    "Bootstrap"]
               }
          }
     }).mount("#log");
     // посмотрим, что под капотом
     console.log(typeof vm.pro);
     console.log(vm.pro);
</script>
```

Пусть теперь в опции **data** хранится **объект**:

```
var vm = Vue.createApp({
     data() {
          return {
          tutor: {
               name: "Denis",
               experience: "5",
               lessonsOnline: 490,
               rating: 4.95,
               technologies: "HTML, JavaScript, VueJS, Bootstrap"
               }
          }
     }
```

```
}).mount("#log");
```

Выведем содержимое объекта в шаблон приложения и в отладочную консоль.

example_3. Объекты в свойстве data

```
<div id="log">
     <h2>Ваш личный репетитор</h2>
     <!-- к свойству объекта обращаемся через точку -->
     Имя: {{ tutor.name }} <br />
     Стаж: {{ tutor.experience }} <br />
     Уроков онлайн: {{ tutor.lessonsOnline }} <br />
     Рейтинг: {{ tutor.rating }} <br />
     Преподаваемые технологии: {{ tutor.technologies }}
</div>
<script>
     var vm = Vue.createApp({
          data() {
          return {
               tutor: {
               name: "Denis",
               experience: "5",
               lessonsOnline: 490,
               rating: 4.95,
               technologies: "HTML, JavaScript, VueJS, Bootstrap"
               }
               }
          }
     }).mount("#log");
     console.log();
```

```
</script>
```

Ну и, естественно, в опции **data** могут храниться **смешанные данные**.

example_4. Сложные структуры данных в свойстве data

```
<div id="log">
     <h2>Ваш личный репетитор</h2>
     <!-- к свойству объекта обращаемся через точку -->
     Имя: {{ tutor.name }} <br />
     Стаж: {{ tutor.experience }} <br />
     Уроков онлайн: {{ tutor.lessonsOnline }} <br />
     Рейтинг: {{ tutor.rating }} <p>
     Преподаваемые технологии:
     <!-- свойство technologies является массивом -->
     <ul>
          <li>{{ tutor.technologies[0] }}</li>
          <li>{{ tutor.technologies[1] }}</li>
          <li>{{ tutor.technologies[2] }}</li>
          <li>{{ tutor.technologies[3] }}</li>
     </ul>
</div>
<script>
     var vm = Vue.createApp({
          data() {
               return {
               // в data хранится объект -tutor
               // в объекте один из ключей - массив
               tutor: {
```

```
                name: "Denis",
                experience: "5",
                lessonsOnline: 490,
                rating: 4.95,
                technologies: ["HTML", "JavaScript", "VueJS",
                "Bootstrap"]
                }
                }
            }
        }).mount("#log");
        console.log();
</script>
```

При работе с объектами можно воспользоваться **альтернативным** способом доступа к его свойствам:

```
obj.a <=> obj['a']
```

```
<div id="app">
        {{ obj['a'] }}
        {{ obj['b'] }}
        {{ obj['c'] }}
</div>
```

Задача 1

Создайте два **экземпляра объекта Vue**. Первый экземпляр пусть хранит **информацию об авторе**. Второй экземпляр хранит **информацию о книгах автора**. Информация о книгах должна храниться в виде **массива**.

Выведите в шаблоны данные экземпляров. Данные о книгах автора выведите в виде списка.

Массивы и объекты в реальных приложениях

Массивы и свойства объекта могут содержать **значения любого типа** (простого или сложного). Это позволяет разработчику строить **иерархию данных**. Иерархия может быть настолько сложной, насколько этого требует решаемая задача.

Наиболее используемый способ наглядного отображения данных – **табличное** представление. Например, книги замечательного писателя Довлатова С.Д. могут быть представлены следующей таблицей:

Книги

Наименование	ISBN	Год издания	Издательство
Чемодан (сборник)	978-5-91181-427-4	2007	Азбука-классика
Компромисс	978-5-389-02277-5	2019	Азбука
Наши	978-5-389-10578-2	2016	Азбука-Аттикус

Таблица хороша как для визуального представления информации, так и для описания с помощью структурированных типов данных. Так, таблица представляет собой **массив**, в котором каждый элемент является **объектом**, обладающим определенным набором свойств.

Условно такую структуру данных можно описать следующим образом:

```
Книги = [
    {
        Наименование : Чемодан (сборник),
        ISBN : 978-5-91181-427-4,
        Год Издания : 2007
        Издательство : Азбука-классика
    },
    {
        Наименование : Компромисс,
        ISBN : 978-5-389-02277-5,
```

```
            Год Издания : 2019,
            Издательство : Азбука
      },
      {
            Наименование : Наши,
            ISBN : 978-5-389-10578-2,
            Год Издания : 2016,
            Издательство : Азбука-Аттикус
      }
]
```

Таким образом, представленная структура в программном коде может быть реализована следующим образом.

example_5. Массивы объектов в свойстве data

```
<script>
      // экземпляр не примонтирован к шаблону
      // в этом скрипте просто описание массива -book
      var vmBook = Vue.createApp({
            data() {
                  return {
                        book : [
                        {
                              title : "Чемодан (сборник)",
                              ISBN : "978-5-91181-427-4",
                              year : 2007,
                              publisher : "Азбука-классика"
                        },
                        {
```

```
                    title : "Компромисс",
                    ISBN : "978-5-389-02277-5",
                    year : 2019,
                    publisher : "Азбука"
                },
                {
                    title : "Наши",
                    ISBN : "978-5-389-10578-2",
                    year : 2016,
                    publisher : "Азбука-Аттикус"
                }
                ]
            }
        }
    });
</script>
```

Хуки жизненного цикла

Каждый экземпляр Vue при создании проходит через последовательность шагов инициализации — например, настраивает наблюдение за данными, компилирует шаблон, монтирует экземпляр в DOM, обновляет DOM при изменении данных. **Между этими шагами вызываются функции, называемые хуками жизненного цикла, с помощью которых можно выполнять свой код на определённых этапах**.

Например, хук **created** можно использовать для выполнения кода **после создания экземпляра**.

example_6. Хук на событие created

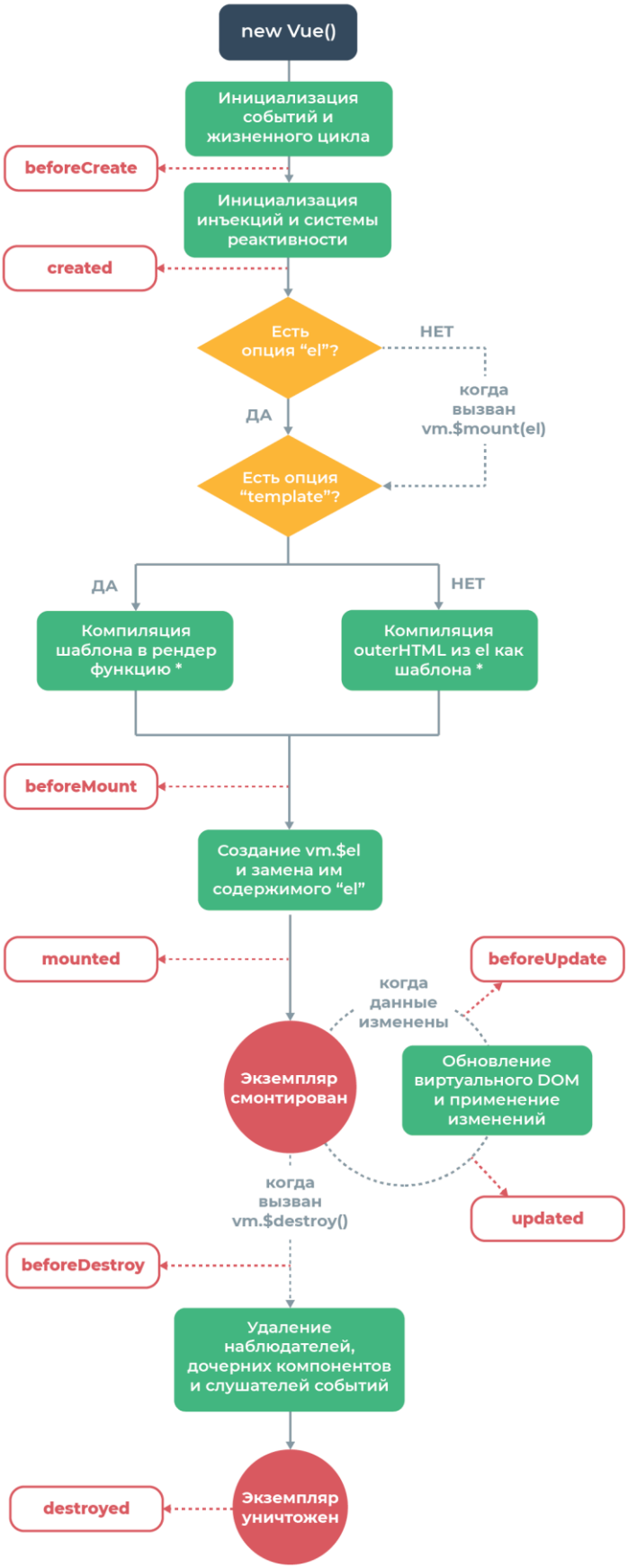
```
<div id="app">
     <h2>{{ message }}</h2>
</div>
<script>
     // инициализация экземпляра
     var vm = Vue.createApp({
          data() {
               return {
                    message: "Hello, I'm VueJS"
          }},
          // хук на создание экземпляра
          created() {
               console.log('Экземпляр приложения Vue создан')
          }
     }).mount("#app");
</script>
```

Существуют и другие хуки, вызываемые на различных стадиях жизненного цикла экземпляра. Вот, например, **некоторые** из них:

- **beforeCreate**
- **beforeMount**
- **mounted**
- **updated**

Хуки жизненного цикла помогают понять, как работает используемая библиотека. Блоки жизненного цикла позволяют узнавать, когда компонент создается, добавляется в модель DOM, обновляется или уничтожается.

На схеме из официальной документации VueJS показан **жизненный цикл экземпляра Vue**.


new Vue()
Инициализация событий и жизненного цикла
beforeCreate
Инициализация инъекций и системы реактивности
created
Есть опция “el”?
НЕТ
когда вызван vm.$mount(el)
ДА
Есть опция “template”?
ДА
НЕТ
Компиляция шаблона в рендер функцию *
Компиляция outerHTML из el как шаблона *
beforeMount
Создание vm.$el и замена им содержимого “el”
mounted
beforeUpdate
когда данные изменены
Экземпляр смонтирован
Обновление виртуального DOM и применение изменений
updated
когда вызван vm.$destroy()
beforeDestroy
Удаление наблюдателей, дочерних компонентов и слушателей событий
Экземпляр уничтожен
destroyed

Описание хуков

Хуки создания (инициализация)

Хуки создания — это самые первые хуки, которые запускаются в компоненте. Позволяют выполнять действия даже до добавления компонента в модель DOM. Используются, как правило, при рендеринге компонента.

beforeCreate

Хук **beforeCreate** запускается в начале инициализации компонента. Данные не сделаны реактивными, а события еще не настроены.

created

Вы можете получить доступ к реактивным данным и активным событиям с помощью хука created. Шаблоны и виртуальная модель DOM еще не смонтированы, и их рендеринг не выполнен.

Хуки монтирования (вставка DOM)

Хуки монтирования используются чаще всего. Они обеспечивают мгновенный доступ к компоненту до и после первого рендеринга.

Используйте хуки монтирования, если вам требуется получить доступ или изменить DOM вашего компонента непосредственно до или после начального рендеринга.

Не используйте хуки монтирования, если вам требуется **доставить данные компонента при инициализации**.

beforeMount

Хук **beforeMount** запускается до начального рендеринга и после компиляции шаблона или функций рендеринга.

mounted

Хук **mounted** дает полный доступ к реактивному компоненту, шаблонам и модели DOM после рендеринга.

Хуки обновления (повторный рендеринг)

Хуки обновления вызываются, когда **изменяется реактивное свойство**, используемое вашим компонентом, или когда что-то еще вызывает его повторный рендеринг.

Используйте хуки обновления, если вам нужно знать, когда ваш компонент выполняет повторный рендеринг, возможно для целей отладки или профилирования.

Не используйте хуки обновления, если вам нужно знать, когда изменяется реактивное свойство вашего компонента. Используйте для этой цели computed или properties.

beforeUpdate

Хук **beforeUpdate** запускается после изменения данных вашего компонента и начала цикла обновления и до исправления и повторного рендеринга модели DOM.

Используйте beforeUpdate, если вам нужно получить новое состояние любых реактивных данных вашего компонента до фактического рендеринга.

updated

Хук **updated** запускается после изменения данных вашего компонента и повторного рендеринга DOM.

Используйте **updated**, если вам требуется доступ к DOM после изменения свойств.

Хуки уничтожения (разрушение)

Хуки уничтожения позволяют выполнять действия во время уничтожения компонента, например, при очистке или отправке аналитических данных. Они срабатывают, когда ваш компонент уничтожается и удаляется из DOM.

beforeDestroy

beforeDestroy срабатывает прямо перед уничтожением. Ваш компонент все еще присутствует и полностью функционален.

Используйте beforeDestroy, если вам нужно очистить события или реактивные подписки.

destroyed

Когда вы достигнете хука **destroyed**, от вашего компонента уже практически ничего не останется. Все, что было к нему прикреплено, будет уничтожено.

Используйте destroyed, если вам необходимо провести заключительную очистку или сообщить удаленному серверу об уничтожении компонента.

Задача 2

Создайте **экземпляр объекта Vue**. Выведите в **консоль** пользовательские сообщения на следующие хуки объекта: **beforeCreate**, **created**, **beforeMount**, **mounted**, **beforeUpdate**, **updated**.

Задача 3

Подключите в скрипт внешние данные. Обратите внимание на тип данных переменных **JavaScript**. Переменная **dataUser** – объект. Переменная **dataOrder** представляет массив из одного элемента типа объект. Создайте **два экземпляра** приложения Vue. **Выведите** данные в шаблоны приложений.

Первое приложение

В конце урока приведу пример простой программы, написанной с использованием фреймворка. В программе задействую все изученные ранее возможности по **обработке и выводу данных приложением**.

В программе реализован следующий алгоритм:

- **подключение** данных из внешних файлов. Данные сохранены в текстовом формате JSON. В реальных приложениях данные могут поставляться в асинхронном взаимодействии клиент-сервер;
- создание **экземпляра** приложения;
- создание хука **beforeCreate**. В обработчике хука преобразуем данные из текстового формата JSON в объект JavaScript. Для парсинга данных используем метод JSON.parse();
- создание хука **created**. В обработчике хука записываем в свойства data полученные объекты JS;
- вывод данных экземпляра в шаблон.

example_7. Первая программа

```
<div id="app">

    <h1>{{ author.surname }}  {{ author.name }} {{
    author.patronymic }}</h1>

    <p>

        Дата рождения:  <b>{{ author.dateBirth }}</b><br/>

        Город рождения: <b>{{ author.city }}</b><br/>

        Род деятельности: <b>{{ author.profession }}</b><br/>

        Язык произведений: <b>{{ author.language }}</b>

    </p>

    <hr>

    <h2>Книги автора</h2>

    <div>

        Название: <b>{{book[0].title}}</b><br>

        ISBN: <b>{{book[0].ISBN}}</b><br>

        Год выпуска: <b>{{book[0].year}}</b><br>

        Издательство: <b>{{book[0].publisher}}</b><br>

    </div>

    <div>
```

```
            Название: <b>{{book[1].title}}</b><br>
            ISBN: <b>{{book[1].ISBN}}</b><br>
            Год выпуска: <b>{{book[1].year}}</b><br>
            Издательство: <b>{{book[1].publisher}}</b><br>
      </div>
</div>
<script>
      // создание экземпляра
      var vm = Vue.createApp({
            // для реактивности данных объявляем свойства null
            data() {
            return {
                  book : null,
                  author : null
            }
            },
            // хук на событие beforeCreate
            // предварительная обработка принятых данных
            beforeCreate() {
                  // парсим формат JSON в объект
                  book = JSON.parse(bookJSON);
                  author = JSON.parse(authorJSON);
            },
            // хук на событие create
            created() {
                  // записываем в data обработанные объекты
                  this.book = book;
```

```
                this.author = author;

            }

        }).mount("#app");

</script>
```

Первая программа явно не идеальна. Явно не хватает циклической конструкции вывода данных, и хотелось бы добавить вывод изображений. Ну, так это только третий урок!

Все будет. Хорошего кодинга ;)