Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»
Филиал
«Минский радиотехнический колледж»

Учебная дисциплина «Программные средства создания Internet-приложений»

Инструкция

по выполнению лабораторной работы
«Создание приложения с использованием асинхронных событий»

Минск
2021

Лабораторная работа № 29

Тема работы: Создание приложения с использованием асинхронных событий

1. Цель работы

Формирование умений использования асинхронных событий в сценариях на языке JavaScript.

2. Задание

Выполнить задания в соответствии с порядком выполнения лабораторной работы.

3. Оснащение работы

ПК, редактор исходного кода, браузер.

4. Основные теоретические сведения

JavaScript – однопоточный язык программирования. Движок JS единовременно может обрабатывать только одно выражение – в одном потоке.

В каждом окне выполняется только один главный поток, который занимается выполнением JavaScript, отрисовкой и работой с DOM.

Он выполняет команды последовательно, может делать только одно дело одновременно и блокируется при выводе модальных окон, таких как alert.

Синхронный код внутри движка JS выполняется следующим образом:

```
const day = () => {
  console.log('Добрый день!');
};
const morning = () => {
  console.log('Доброе утро!');
  day();
  console.log('Добрый вечер!');
};
morning();
// Доброе утро!
// Добрый день!
// Добрый вечер!
```

Однако, такой подход не позволяет выполнять длительные операции, например, сетевой доступ, не блокируя основной поток. Асинхронность в таких случаях нужна, чтобы выполнять длительные сетевые запросы, не блокируя основной поток.

Есть два типа стиля асинхронного кода в JavaScript, старый метод – callbacks (обратные вызовы) и более новый – promise (промисы, обещания).

Асинхронные обратные вызовы – это функции, которые определяются как агрументы при вызове функции, которая начнет выполнение кода на заднем фоне. Когда код на заднем фоне завершает свою работу, он вызывает функцию обратного вызова, оповещающую, что работа сделана, либо оповещающую о трудностях в завершении работы.

```
const greating= ()=>{
    setTimeout(()=>{console.log('Good night!');}, 2000);
}
greating ();
console.log('Good morning!');
console.log('Good afternun!');
```

```
// Good morning!
// Good afternun!
// Good night!
```

Объект Promise (промис) используется для отложенных и асинхронных вычислений. Создать Promise можно с помощью ключевого слова new:

```
new Promise(function(resolve, reject) {
функция-исполнитель
});
```

Обычно функция-исполнитель описывает выполнение какой-то асинхронной работы, по завершении которой необходимо вызвать функцию resolve или reject.

В качестве первого аргумента передается та функция, которая вызывается в случае успешного выполнения, вторым аргументом указывается функция, которая будет вызвана в случае ошибки (имена можно заменить).

Аргументы resolve и reject – это колбэки, которые предоставляет сам JavaScript, поэтому не нужно их писать. Нужно лишь, чтобы функция-исполнитель вызвала одну из них по готовности.

При создании объекта Promise выполняется функция-исполнитель, после получения результата которой должен быть вызван один из этих колбэков:

resolve(value) – если работа завершилась успешно, с результатом value;

reject(error) – если произошла ошибка, error – объект ошибки.

У объекта Promise, возвращаемого конструктором new Promise, есть внутренние свойства:

- state («состояние») – вначале "pending" («ожидание»), потом меняется на "fulfilled" («выполнено успешно») при вызове resolve или на "rejected" («выполнено с ошибкой») при вызове reject;

- result («результат») – вначале undefined, далее изменяется на value при вызове resolve(value) или на error при вызове reject(error).

```
promise= new Promise(function (resolve,reject){
     a=+prompt('enter a');
     b=+prompt('enter b');
if (a>b) {
    resolve(console.log('Ура! Победа!'));
   }
   else
   {
    reject(new Error("Увы! Мы проиграли!"));
   }
    })
```

Пользовательская функция-исполнитель должна вызвать что-то одно: resolve или reject. Т.к. состояние промиса может быть изменено только один раз. Все последующие вызовы resolve и reject будут проигнорированы:

```
promise= new Promise(function (resolve,reject){
     a=+prompt('enter a');
     b=+prompt('enter b');
if (a>b) {
    resolve(console.log('Ура! Победа!'));
    reject(new Error("Увы! Мы проиграли!"));    //будет проигнорировано

}
   else
   {
```

```
    reject(new Error("Увы! Мы проиграли!"));
   }

})
```

Идея в том, что задача, выполняемая функцией-исполнителем, может иметь только один итог: результат или ошибку.

Функция resolve/reject ожидает только один аргумент (или ни одного). Все дополнительные аргументы будут проигнорированы.

Обычно функция-исполнитель делает что-то асинхронное и после этого вызывает resolve/reject, то есть через какое-то время. Однако, если задача функции-исполнителя не требует время (в отличие, например, при получении ответа от сервера или обращении к БД), resolve или reject могут быть вызваны сразу.

В случае, если необходимо, чтобы в зависимости от определенных условий выполнялся вызов либо resolve, либо reject, можно создать функцию, которая будет возвращать объект Promise.

```
function rezult (){
    return new Promise(function (resolve,reject){
     a=+prompt('enter a');
     b=+prompt('enter b');
if (a>b) {
    resolve();
   }
   else
   {
    reject();
   }
    })
};
```

Свойства state и result – это внутренние свойства объекта Promise и прямой доступ к ним отсутствует. Для обработки результата следует использовать методы .then/.catch/.finally.

Метод .then() возвращает Promise. Метод может принимать два аргумента. Первый аргумент метода .then – функция, которая выполняется, когда промис переходит в состояние «выполнен успешно», и получает результат (т.е. выполняется вызов resolve()). Второй аргумент .then – функция, которая выполняется, когда промис переходит в состояние «выполнен с ошибкой», и получает ошибку (т.е. выполняется вызов reject()).

```
function rezult (){
    return new Promise(function (resolve,reject){
     a=+prompt('enter a');
     b=+prompt('enter b');
if (a>b) {
    resolve();
   }
   else
   {
    reject();
   }
    })
};
const promise=rezult();
promise
   .then(()=>console.log('Мы выиграли!'),
        ()=> console.log('Мы Проиграли!')
            );
```

Если в .then() передать только одну функцию, то обработка ошибки выполняться в таком случае не будет.

Если необходимо обработать только ошибку, то можно использовать null или undefined в качестве первого аргумента then(null, onRejected). Или можно воспользоваться методом .catch.

Метод catch() возвращает Promise(обещание) и работает только в случае отклонения обещания.

```
const promise=rezult();
promise
    .then(()=>{console.log('Мы выиграли!');
            let  c=prompt('Что же дальше?');
              return c;
           }
             )
    .then((c)=>console.log(c))
    .catch(()=> {console.log('Увы! Мы проиграли!');});
```

Так как метод .then() возвращает Promise, можно объединить несколько вызовов then в цепочку. При необходимости, then может возвращать значение, которое можно передавать в следующий в цепочке промисов then.

```
var p2 = new Promise(function(resolve, reject) {
  resolve(1);
});

p2.then(function(value) {
  console.log(value); // 1
  return value + 1;
}).then(function(value) {
  console.log(value); // 2
});

p2.then(function(value) {
  console.log(value); // 1
});
```

В случае вызова .finally(f) функция f выполнится в любом случае, когда промис завершится: успешно или с ошибкой.

```
promise
    .finally(() => console.log ('Игра закончилась!'))
    .then(()=>{console.log('Мы выиграли!');
            let  c=prompt('Что же дальше?');
              return c;
           }
             )
    .then((c)=>console.log(c))
    .catch(()=> {console.log('Увы! Мы проиграли!');});
```

Если промис в состоянии ожидания (state="pending"), обработчики в .then/catch/finally будут ждать его. Однако, если Promise уже завершён, то обработчики выполнятся сразу.

5. Порядок выполнения работы

1. Добавьте на html-страницу текстовое поле ввода. Назначьте два обработчика событий для указанного поля на ‘click’ и ‘focus’, которые выводят информацию о типе

события. Добавьте обработчик, который изменяет содержимое поля ввода при получении фокуса. Создайте асинхронное событие для обработки формы, в котором сперва сработает событие onclick(), а потом onfocus().

2. Реализовать по клику на кнопку запуск слайдера, который с указанным интервалом будет загружать очередное изображение. В случае, если будет передано отрицательное число или 0 – должно выдаваться сообщение об ошибке. Очередное изображение должно загружаться после того, как отображено предыдущее.

3*. Реализуйте визуальный анимационный эффект средствами JavaScript таким образом, чтобы функция requestAnimationFrame возвращала промис, вместо того чтобы принимать в аргументы функцию-callback.

6. Форма отчета о работе

Лабораторная работа № ___

Номер учебной группы _____
Фамилия, инициалы учащегося _____
Дата выполнения работы _____
Тема работы: _____
Цель работы: _____
Оснащение работы: _____
Результат выполнения работы: _____
_____
_____

7. Контрольные вопросы и задания

1. Что представляет собой объект Promise?
2. Укажите внутренние свойства объекта Promise. Как они влияют на объект?
3. Для чего предназначены методы .then/.catch/.finally?
4. В чем суть использования «async/await» при работе с промисами?

8. Рекомендуемая литература

1. **JAVASCRIPT.RU** [Электронный ресурс] / Современный учебник JavaScript – 2007—2020 Илья Кантор. – Режим доступа: https://learn.javascript.ru. – Дата доступа: 04.03.2020.

2. **Никсон, Р.** Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 / Р. Никсон. 4-е изд. – СПб.: Питер, 2018.

3. **Симпсон, К.** ES6 и не только / К. Симпсон. – СПб.: Питер, 2017.

4. **Хавербеке, М.** Выразительный JavaScript. Современное веб-программирование / М. Хавербеке – СПб.: Питер, 2019.