Замыкания

№ урока:	14	**Курс:**	JavaScript Базовый

Средства обучения:	Visual Studio Code Web Browser

Обзор, цель и назначение урока

Научиться использовать замыкания, понять внутренний механизм работы замыканий, основанный на лексическом окружении.

Изучив материал данного занятия, учащийся сможет:

- Работать с globalthis.
- Понимать, что такое контекст выполнения и стек вызова.
- Понимать, что такое лексическое окружение и как оно используется при разрешении идентификаторов.
- Использовать замыкания и понимать их внутренний механизм работы.

Содержание урока

1. Глобальный объект
2. Контекст выполнения (Execution Context)
3. Лексическое окружение (Lexical Environment)
4. Замыкания (Closures)

Резюме

- Код, выполняемый на JavaScript, находится в контексте выполнения глобального объекта, который в браузере представлен объектом windows, а в Node.JS - объектом global.
  При написании сценариев, которые полагаются на глобальный объект и будут работать в разных средах выполнения, для ссылки на глобальный объект можно использовать переменную globalThis. globalThis – стандартизированное имя глобального объекта.

- Переменные, созданные с помощью var и функции определенные в script, становятся свойствами глобального объекта. Если в сценарии необходимо создать идентификатор, доступный в любой части текущего сценария или других сценариях, такой идентификатор нужно сделать свойством глобального объекта.

- Глобальные переменные и функции стоит избегать, так как могут возникнуть конфликты, связанные с тем, что другие сценарии начнут определять глобальные переменные или функции с таким же именем, как созданные ранее, что может повлечь за собой непредсказуемое поведение.

- Контекст выполнения – специальный механизм, который используется для описания окружения, в котором выполняется JavaScript код (для сохранения объявленных переменных, this и других состояний важных для работы кода). Любой выполняемый код выполняется в контексте выполнения.

CyberBionic systematics

CyberBionic Systematics ®
19 Evgenia Sverstiuka Str., 5 floor
Kyiv, Ukraine

t. +380 (44) 361-8937
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

- **Стек выполнения (execution stack) –** сохраняет все контексты выполнения, которые были созданы в процессе работы кода. Когда запускается функция, ее контекст выполнения добавляется в стек, когда функция завершает свою работу – ее контекст удаляется из стека.

- Можно выделить три типа контекстов выполнения
  - Global Execution Context – создается один раз при запуске сценария
  - Functional Execution Context – создается при каждом запуске функции
  - Eval Function Execution Context – код, выполняемый через eval

- Контекст выполнения создается в два этапа:
  - 1) Creation Phase – создаются и инициализируются компоненты контекста.
  - 2) Execution Phase – выполняется код.

- Контекст выполнения содержит:
  - Лексическое окружение (LexicalEnvironment) – для определения какой идентификатор с каким значением связан.
  - Окружение переменных (VariableEnvironment) – для переменных var.

  Лексическое окружение содержит:
  - EnvironemtRecord – содержит определения переменных и функций
  - OuterEnv – внешнее окружение на основе стека вызова
  - ThisBinding – контекст (this) выполняемого кода, связанного с этим лексическим окружением

- **Замыкание (closure)** — это функция и лексическое окружение, в котором эта функция была создана.

```
function makeCounter() {
      let counter = 0;

            function increment() {
                  return counter += 1;
                  }
       return increment;
      }
```

При запуске функции makeCounter будет создано лексическое окружение, в котором будет находится переменная counter. Каждая функция содержит внутреннюю ссылку на лексическое окружение, в котором она была создана. Когда функция increment возвращается из функции makeCounter, лексическое окружение makeCounter не может быть удалено из оперативной памяти, потому что на него ссылается функция increment.
Если повторно запустить функцию makeCounter - будет создано новое лексическое окружение и новая функция increment будет содержать в себе на него ссылку. Функция и связанные с ней лексические окружения представляют замыкание.

Замыкания могут быть причиной большего расхода памяти если в замыкании хранится достаточно большая цепочка лексических окружений.

CyberBionic systematics

CyberBionic Systematics ®
19 Evgenia Sverstiuka Str., 5 floor
Kyiv, Ukraine

t. +380 (44) 361-8937
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

- Замыкания часто используются при реализации различных шаблонов кодирования в JavaScript. Например, для реализации шаблона модуль – для создания объекта с открытыми и закрытыми свойствами и методами.

- **IIFE** (immidiatly invoked function expression) - функция, запускаемая сразу после объявления. Прием, который позволяет определить блок кода, в котором созданные идентификаторы будут локальными, а не глобальными, так как создавая переменную в функции она находится в лексическом окружении данной функции. IIFE широко использовались в предыдущих версиях JavaScript до введения модулей и let и позволяли легко контролировать количество создаваемых глобальных переменных.

Закрепление материала

- Что такое globalThis
- Что такое контекст выполнения, чем он отличается от контекста функции?
- Что такое лексическое окружение?
- Что такое IIFE, зачем используется данная конструкция?
- Что такое замыкание, опишите принцип работы замыканий.

Самостоятельная деятельность учащегося

Выполните задания в директории Exercises\Tasks\014 Closures. Текст задач находится в комментариях в тегах script

Рекомендуемые ресурсы

ECMAScript 2022 Language Specification. Executable Code and Execution Context
https://tc39.es/ecma262/#sec-executable-code-and-execution-contexts

Замыкания
https://developer.mozilla.org/ru/docs/Web/JavaScript/Closures

IIFE
https://developer.mozilla.org/ru/docs/Glossary/IIFE

CyberBionic Systematics ®
19 Evgenia Sverstiuka Str., 5 floor
Kyiv, Ukraine

t. +380 (44) 361-8937
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com