

Dokumentacja Biblioteki insideJS

Robert Bendun

26 marca 2017

Spis treści

1	Wprowadzenie	2
2	Tworzenie podstawowych obiektów	2
3	Funkcje	3
4	Tablice	4

1 Wprowadzenie

Biblioteka insideJS umożliwia pisanie natywnego kodu zbliżonego do języka JavaScript w języku C++. Do swojego działania potrzebuje kompilatora obsługującego standard C++11. Aktualnie biblioteka **nie jest** prekompilowana. W celu używania biblioteki wystarczy dołączyć plik nagłówkowy "js/index.html". W dalszych przykładach będzie on pomijany. Większość elementów biblioteki umieszczona jest w przestrzeni nazw **js** dlatego w dalszych przykładach będzie ona pomijana. Wyjątkami są makra preprocesora. Wyszczególnione są one w rozdziale "Preprocesor".

2 Tworzenie podstawowych obiektów

Język JavaScript nie posiada jawnie zdefiniowanego terminu klasy. Podstawowym sposobem tworzenia obiektów jest przypisanie odpowiednich pól z ich wartościami. Poniżej przykład kodu:

Listing 1: Podstawowy obiekt

```
1 #include <iostream>
2
3 int main()
4 {
5     var object = new Object {
6         "name" is "Doge",
7         "sound" is "Wooooof",
8         "age" is 5
9     };
10
11     std::cout << object["name"] << '\n'; // returns "Doge"
12     std::cout << object["sound"] << '\n'; // returns "Wooooof"
13     std::cout << object["age"] << '\n'; // returns 5
14 }
```

Aby dostać się do pól obiektów należy użyć operatora [], gdzie w należy podać odpowiednią nazwę pola. Do wyświetlania pól obiektu można wykorzystywać standardowe strumienie.

Możliwe jest także stworzenie obiektu poprzez funkcję będącą odpowiednikiem konstruktora. Kluczowym elementem takiej funkcji jest zwracanie null na koniec jej działania.

Listing 2: Konstruktor obiektów

```
1 #include <iostream>
2
3 int main()
4 {
5     var Dog = new Function(INLINE {
6         that["name"] = "Doge";
7         that["sound"] = "Woof";
8         return null;
9     });
10
11     var dog = Dog();
12
13     std::cout << dog["name"] << " says " << dog["sound"];
14 }
```

Konstruktor może także przyjmować argumenty (należy zwrócić uwagę na zmianę z INLINE na INLINE_ARG):

Listing 3: Argument konstruktora

```
1 #include <iostream>
2
3 int main()
4 {
```

```

5     var Dog = new Function(INLINE_ARG {
6         that["name"] = *arguments;
7         that["sound"] = "Woof";
8         return null;
9     });
10
11     var dog = Dog("Spike");
12
13     std::cout << dog["name"] << " says " << dog["sound"];
14 }

```

W przypadku pojedynczego argumentu należy użyć operatora wyluskiwania *. W przypadku podania większej liczby argumentów należy użyć operatora indeksu [].

Listing 4: Argumenty konstruktora

```

1 #include <iostream>
2
3 int main()
4 {
5     var Pair = new Function(INLINE_ARG {
6         that["first"] = arguments[0];
7         that["second"] = arguments[1];
8         return null;
9     });
10
11     var dogs = Pair("Spike", "Doge");
12     std::cout << dogs << std::endl;
13 }

```

3 Funkcje

Aktualnie jedyne funkcje jakie mogą być tworzone są to funkcje lokalne (znajdujące się we wnętrzu klasycznych funkcji z języka C++). Funkcje mogą być tak samo elementem obiektów jak ciągi znaków bądź liczby.

Listing 5: Funkcja wewnątrz obiektu

```

1 #include <iostream>
2
3 int main()
4 {
5     var object = new Object {
6         "name" is "Doge",
7         "sound" is "Wooooof",
8         "say" is new Function(INLINE{
9             std::cout << that["name"] << " says " << that["sound"] << std::endl;
10         })
11     };
12
13     object["say"]();
14 }

```

Słowo kluczowe `that` jest to kontekst funkcji. Kiedy funkcja jest elementem obiektu jak w powyższym przykładzie `that` odwołuje się do danego obiektu. Kiedy używamy funkcji konstruktora wtedy `that` jest odwołaniem do nowego obiektu. Aby uruchomić funkcję z wymuszonym kontekstem należy zastosować metodę `call` obiektu funkcyjnego.

Listing 6: Ustawianie kontekstu

```

1 #include <iostream>
2

```

```

3 int main()
4 {
5     var object = new Object {
6         "name" is "Doge",
7         "sound" is "Wooooof"
8     };
9
10    var say = new Function(INLINE{
11        std::cout << that["name"] << " says " << that["sound"] << std::endl;
12    });
13
14    say.call(object);
15 }

```

Występujące wcześniej `INLINE` oraz `INLINE_ARG` są w rzeczywistości następującymi deklaracjami preprocesora.

Listing 7: Deklaracje preprocesora

```

1 #define INLINE [&](js::iterator&& that)->js::Atom*
2 #define INLINE_ARG [&](js::iterator&& that, js::iterator &&arguments)->js::Atom*

```

Klasa `iterator` jest to wrapper przechowujący wskaźnik do rzeczywistego obiektu/funkcji. Umożliwia takie operacje jak wyłuskanie, wywołanie czy indeksowanie. Więcej informacji znajduje się w rozdziale "Przechowywanie obiektów".

4 Tablice

Tablice tworzone są w sposób analogiczny do obiektów. Wyjątkiem jest sposób nadawania identyfikatorów. Występuje on niejawnie.

Listing 8: Tworzenie tablicy

```

1 #include <iostream>
2
3 int main()
4 {
5     var table = new Array{ "Hello", 10, "World", 20, 30 };
6     std::cout << table[0] << std::endl; // returns "Hello"
7     std::cout << table << std::endl; // returns ["Hello", 10, "World", 20, 30]
8 }

```

Jest wiele sposobów na dostanie się do elementów kontenera. Aby iterować po elementach tablicy można wykorzystać:

- pętlę zakresową `for` z C++11
- tradycyjną pętlę `for`
- metodę `for_each` (a dokładnie to makro ułatwiające korzystanie z niej)

Wszystkie z poniższych metod pokazano poniżej. Rezultatem każdej z nich jest ciąg znaków "Hello 10 World 20".

Listing 9: Iteracja po tablicy

```

1 #include "js/index.hpp"
2
3 int main()
4 {
5     using namespace js;
6
7     var table = new Array{ "Hello", 10, "World", 20 };
8     std::string result1, result2, result3;

```

```

9
10 // 1
11 for(auto &x : cast<Array>(table))
12     result1 += x->toString() + ' ';
13
14 // 2
15 cast<Array>(table).forEach({
16     result2 += element->toString() + ' ';
17 });
18
19 // 3
20 for(std::size_t i=0; i < cast<Array>(table).length(); i++)
21     result3 += table[i]->toString() + ' ';
22
23 }

```

Powyższy przykład można wykonać dużo prościej za pomocą metody join.

Listing 10: Wykorzystywanie metody join

```

1 int main()
2 {
3     var table = new Array{ "Hello", 10, "World", 20 };
4     std::string result = cast<Array>(table).join(" ");
5 }

```

Listings

1	Podstawowy obiekt	2
2	Konstruktor obiektów	2
3	Argument konstruktora	2
4	Argumenty konstruktora	3
5	Funkcja wewnątrz obiektu	3
6	Ustawianie kontekstu	3
7	Deklaracje preprocesora	4
8	Tworzenie tablicy	4
9	Iteracja po tablicy	4
10	Wykorzystywanie metody join	5