



Prof. Dr. Leandro Luque
Faculdade de Tecnologia de Mogi das Cruzes

JAVASCRIPT

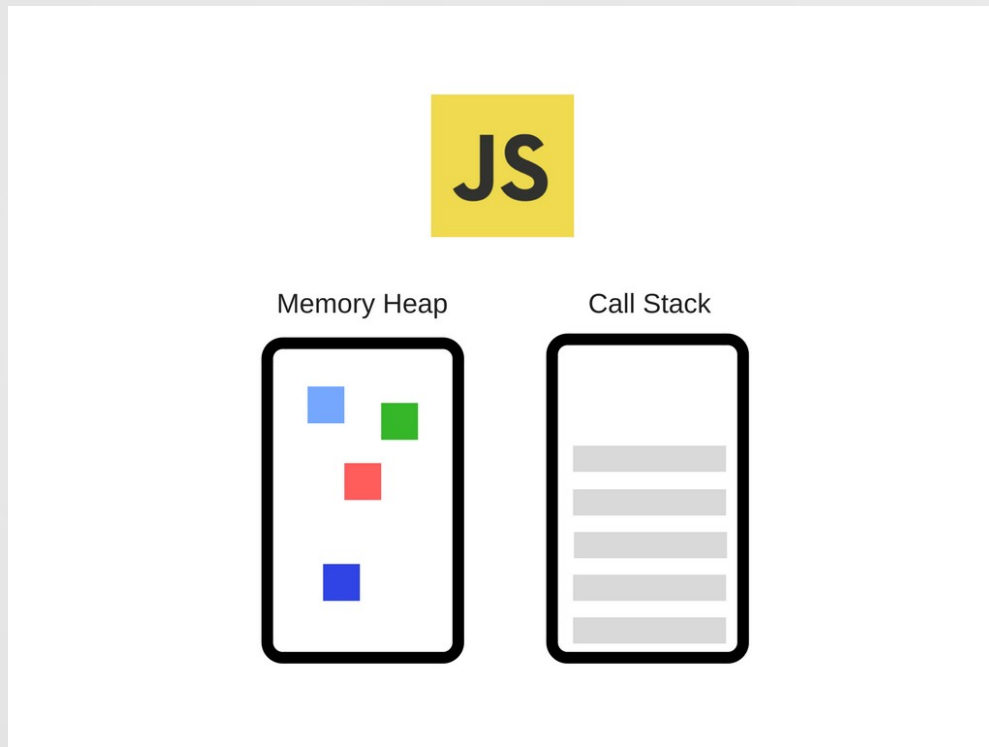
Loop event, Assíncrono e Geradores



O QUE PRECISAMOS GUARDAR NA MEMÓRIA PARA EXECUTAR UM PROGRAMA?

- Dados gerados pelo nosso programa (variáveis, constantes etc.);
- Onde nosso programa está? O que está sendo executado e o que vem depois?

ORGANIZAÇÃO DA MEMÓRIA EM JAVASCRIPT

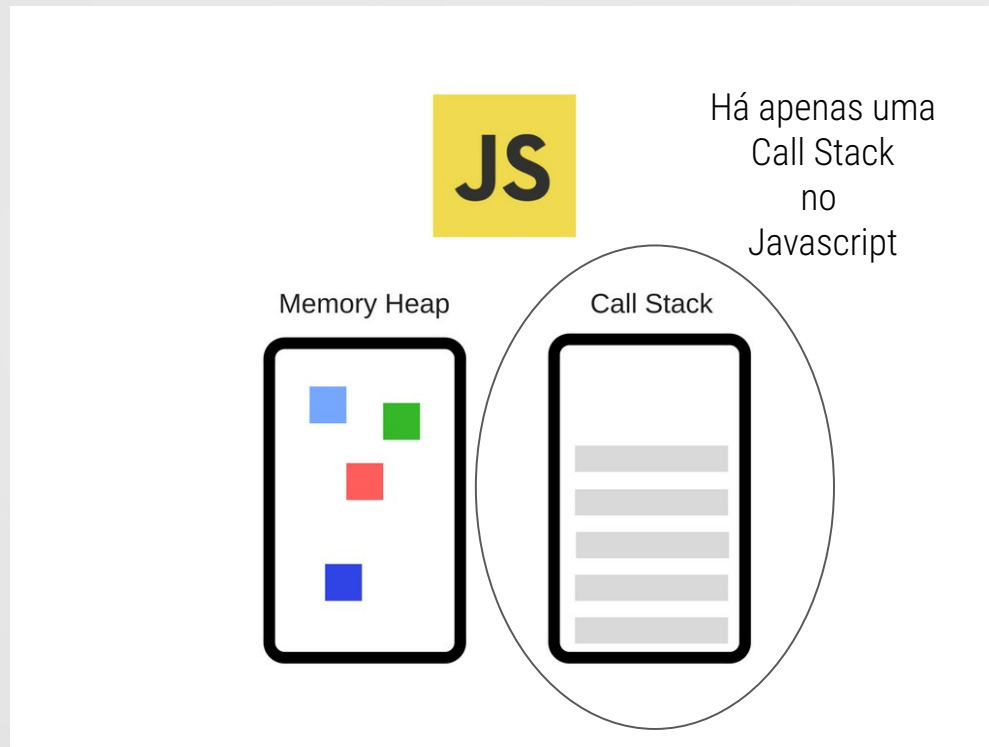


ORGANIZAÇÃO DA MEMÓRIA EM JAVASCRIPT

Exemplo de Call Stack

```
function multiplica(x, y) {  
  return x * y;  
}  
  
function imprimeQuadrado(x) {  
  var resultado = multiplica(x, x);  
  console.log(resultado);  
}  
  
imprimeQuadrado(5);
```

ORGANIZAÇÃO DA MEMÓRIA EM JAVASCRIPT





SINGLE-THREADED VS MULTI-THREADED LANGUAGE



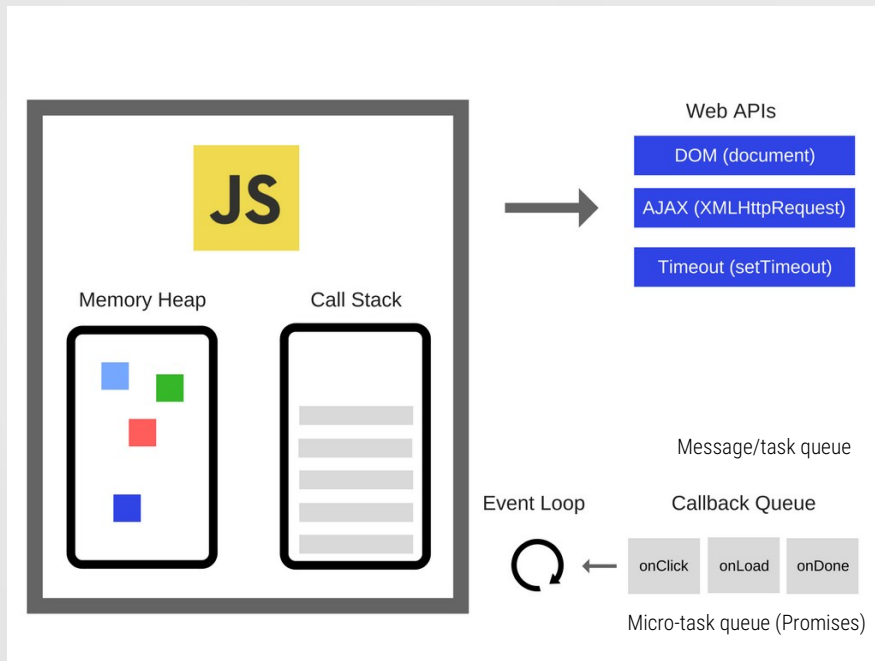
● Ver desenho do professor.

ORGANIZAÇÃO DA MEMÓRIA EM JAVASCRIPT

- Cada entrada da Call Stack é chamada de **Stack Frame**;
- É o conteúdo da Call Stack que é exibido no caso de uma exceção:

```
function terceira() {  
  throw new Error("Veja só o stacktrace");  
}  
function segunda() {  
  terceira();  
}  
function primeira() {  
  segunda();  
}  
primeira();
```

ALGUNS COMPONENTES DA EXECUÇÃO JAVASCRIPT



EVENT LOOP

```
while (true) {  
  if(callStack.estaVazia() && callbackQueue.temEvento()) {  
    callStack.adicionar(callbackQueue.primeiro());  
  }  
}
```



EXEMPLO DE EXECUÇÃO COM SETIMEOUT

EXEMPLOS DE USO DE API WEB COM CALLBACK

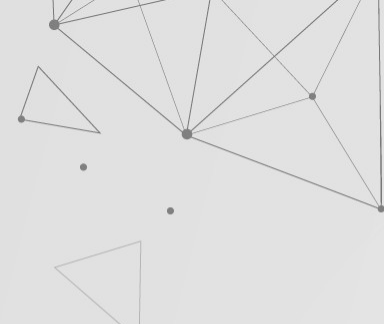
```
var xhr = new XMLHttpRequest();

xhr.onreadystatechange = function() {
  if (this.readyState == XMLHttpRequest.DONE) {
    if(this.status == 200) {
      alert(xhr.responseText);
    }
  }
}


xhr.open('GET', 'http://example.com', true);
xhr.send();
```

Fontes:

<https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/open>
<https://developer.mozilla.org/pt-BR/docs/Web/API/XMLHttpRequest/readyState>



CALLBACKS PODEM TORNAR A LEITURA UM POUCO DIFÍCIL



```
doSomething(function(result) {  
  doSomethingElse(result, function(newResult) {  
    doThirdThing(newResult, function(finalResult) {  
      console.log('Got the final result: ' + finalResult);  
    }, failureCallback);  
  }, failureCallback);  
}, failureCallback);
```

UMA ALTERNATIVA: PROMESSAS

- Um proxy para um valor que não é necessariamente conhecido quando a promessa é criada;
- Tem três estados:
 - pending (pendente): Estado inicial, que não foi realizada nem rejeitada.
 - fulfilled (realizada): sucesso na operação.
 - rejected (rejeitado): falha na operação.
- Quando se realizar, seu método then é chamado;
- Quando falhar, seu método catch é chamado;

```
let p = new Promise (function (resolve, reject) {  
    // resolva ou rejeite  
})  
.then(resultado => fazAlgo)  
.then(resultado => fazAlgo)  
.then(resultado => fazAlgo)  
.catch(resultado => fazAlgo);
```

USANDO PROMESSAS NA API FETCH

```
fetch('https://viacep.com.br/ws/01001000/json/')  
  .then((resultado) => {  
    return resultado.json();  
  })  
  .then(resultado => console.log(resultado))  
  .catch(erro => console.log(erro));
```

LIDANDO COM ASSÍNCRONO DE MODO SÍNCRONO.

```
async function fetchMovies() {  
  const response = await fetch('/movies');  
  // waits until the request completes...  
  console.log(response);  
}
```

```
/*  
  fetchMovies()  
  .then  
  .catch  
*/
```