Como o Javascript Funciona?

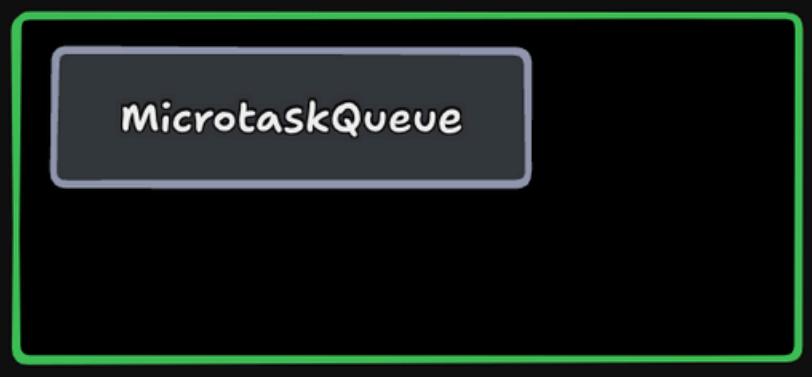
CODE

```
setTimeout(function TaskQueue() {
     console.log("finaizou - Task Queue")
}, 1000);

Promise.resolve()
  .then(function MicrotaskQueue() {
       console.log("finaizou - Microtask Queue")
       });

function fourth() {
  console.log("finaizou - sync")
}
function third() { fourth() }
function second() { third() }
function first() { second() }
first();
```

CALL STACK

Task Queue

Microtask Queue



Event Loop, Microtasks, Task queue, Call Stack e Single Threaded

Isaac Gomes

JS

como nosso código **JS** é executado?

```
setTimeout(function TaskQueue() {
      console.log("finaizou - Task Queue")
}, 0);
Promise.resolve()
  .then(function MicrotaskQueue() {
    console.log("finaizou - Microtask Queue")
});
function fourth() {
    console.log("finaizou - sync")
}
function third() { fourth()
function second() { third() }
function first() { second() }
first();
```

qual a nossa ordem de **execução** e pq? para isso vamos entender as partes da nossa execução

Isaac Gomes

JS

Single Threaded

JavaScript é uma linguagem de thread única, o que significa que só pode executar uma tarefa por vez. No entanto, o Event Loop permite que JavaScript lide com tarefas assíncronas sem bloquear a execução do código principal.

Isaac Gomes

Call Stack

O Call Stack é uma estrutura de dados que armazena informações sobre a execução das funções. Quando uma função é chamada, ela é empilhada no Call Stack, e quando a execução da função termina, ela é removida do Call Stack.

Isaac Gomes

JS

Microtasks Queue

Microtasks Queue** é usada para processar promessas (Promises) e outras tarefas de microtarefas. A Microtasks Queue tem prioridade sobre a **Task Queue,** o que significa que todas as microtarefas devem ser processadas antes que o **Event Loop** passe para o próximo evento na **Task Queue

Isaac Gomes

JS

Task Queue

A Task Queue armazena as mensagens (ou eventos) que precisam ser processadas. Quando uma operação assíncrona, como uma requisição HTTP ou um temporizador, é concluída, uma mensagem é colocada na Task Queue para que o Event Loop a processe.

Isaac Gomes

Event Loop

O Event Loop é um mecanismo que permite que o JavaScript execute código assíncrono e reaja a eventos, mesmo sendo uma linguagem single-threaded. Ele monitora a Call Stack e a Task Queue, garantindo que as tarefas na fila sejam processadas quando a pilha de chamadas está vazia. Assim, o Event Loop mantém o JavaScript não bloqueante e eficiente.

Isaac Gomes

JS

agora vamos acompanhar a execução do nosso código

CODE

```
setTimeout(function TaskQueue() {
     console.log("finaizou - Task Queue")
}, 0);
Promise.resolve()
 .then(function MicrotaskQueue() {
  console.log("finaizou - Microtask Queue")
});
function fourth() {
   console.log("finaizou - sync")
}
function third() { fourth()
function second() { third() }
function first() { second() }
first();
```

CALL STACK

Task Queue

Microtask Queue

RUN

Isaac Gomes

JS

agora vamos acompanhar a execução do nosso código

CODE

```
setTimeout(function TaskQueue() {
     console.log("finaizou - Task Queue")
}, 0);
Promise.resolve()
 .then(function MicrotaskQueue() {
  console.log("finaizou - Microtask Queue")
});
function fourth() {
   console.log("finaizou - sync")
}
function third() { fourth()
function second() { third() }
function first() { second() }
first();
```

CALL STACK

Task Queue

TaskQueue

Microtask Queue

*o primeiro bloco vai para **Task Queue,** pois, trata-se de um temporizador*

Isaac Gomes

JS

agora vamos acompanhar a execução do nosso código

CODE

```
setTimeout(function TaskQueue() {
     console.log("finaizou - Task Queue")
}, 0);
Promise.resolve()
 .then(function MicrotaskQueue() {
  console.log("finaizou - Microtask Queue")
});
function fourth() {
   console.log("finaizou - sync")
}
function third() { fourth()
function second() { third() }
function first() { second() }
first();
```

CALL STACK

Task Queue

TaskQueue

Microtask Queue

MicrotaskQueue

como é uma promise vai para Microtask Queue

Isaac Gomes

JS

agora vamos acompanhar a execução do nosso código

CODE

```
setTimeout(function TaskQueue() {
     console.log("finaizou - Task Queue")
}, 0);
Promise.resolve()
 .then(function MicrotaskQueue() {
  console.log("finaizou - Microtask Queue")
});
function fourth() {
   console.log("finaizou - sync")
}
function third() { fourth()
function second() { third() }
function first() { second() }
first();
```

CALL STACK

first

Task Queue

TaskQueue

Microtask Queue

MicrotaskQueue

como nosso código é sync vai para Call Stack

Isaac Gomes

JS

agora vamos acompanhar a execução do nosso código

CODE

```
setTimeout(function TaskQueue() {
     console.log("finaizou - Task Queue")
}, 1000);

Promise.resolve()
  .then(function MicrotaskQueue() {
       console.log("finaizou - Microtask Queue")
       });

function fourth() {
  console.log("finaizou - sync")
}
function third() { fourth() }
function second() { third() }
function first() { second() }
first();
```

CALL STACK

fourth

third

second

first

Task Queue

TaskQueue

Microtask Queue

MicrotaskQueue

como uma função sync chama a outra elas vão ser espilhadas até o console.log

Isaac Gomes

JS

agora vamos acompanhar a execução do nosso código

CODE

```
setTimeout(function TaskQueue() {
     console.log("finaizou - Task Queue")
}, 0);
Promise.resolve()
 .then(function MicrotaskQueue() {
  console.log("finaizou - Microtask Queue")
});
function fourth() {
   console.log("finaizou - sync")
}
function third() { fourth()
function second() { third() }
function first() { second() }
first();
```

CALL STACK

Task Queue

TaskQueue

Microtask Queue

MicrotaskQueue

uma vez executado o console as funções vão sendo removidas da call stack

Isaac Gomes

JS

agora vamos acompanhar a execução do nosso código

CODE

```
setTimeout(function TaskQueue() {
     console.log("finaizou - Task Queue")
}, 0);
Promise.resolve()
 .then(function MicrotaskQueue() {
  console.log("finaizou - Microtask Queue")
});
function fourth() {
   console.log("finaizou - sync")
}
function third() { fourth()
function second() { third() }
function first() { second() }
first();
```

CALL STACK

MicrotaskQueue

Task Queue

TaskQueue

Microtask Queue

uma vez com a call stack limpa o event loop vai puxar do Microtasks uma vez que o Microtasks tem prioridade sobre o Task Queue

Isaac Gomes

JS

agora vamos acompanhar a execução do nosso código

CODE

```
setTimeout(function TaskQueue() {
     console.log("finaizou - Task Queue")
}, 0);
Promise.resolve()
 .then(function MicrotaskQueue() {
  console.log("finaizou - Microtask Queue")
});
function fourth() {
   console.log("finaizou - sync")
}
function third() { fourth()
function second() { third() }
function first() { second() }
first();
```

CALL STACK

Task Queue

TaskQueue

Microtask Queue

segundo console.log executado com isso a função é removida da call stack

Isaac Gomes

JS

agora vamos acompanhar a execução do nosso código

CODE

```
setTimeout(function TaskQueue() {
    console.log("finaizou - Task Queue")
}, 0);
Promise.resolve()
 .then(function MicrotaskQueue() {
  console.log("finaizou - Microtask Queue")
});
function fourth() {
   console.log("finaizou - sync")
}
function third() { fourth()
function second() { third() }
function first() { second() }
first();
```

CALL STACK

TaskQueue

Task Queue

Microtask Queue

agora que ja passamos pelas sync
e Microtasks puxa as task queue
executando o ultimo console.log
e limpando a call stack

Isaac Gomes

JS

qual foi nossa ordem?

```
setTimeout(function TaskQueue() {
      console.log("finaizou - Task Queue")
}, 0);
Promise.resolve()
  .then(function MicrotaskQueue() {
    console.log("finaizou - Microtask Queue")
});
function fourth() {
    console.log("finaizou - sync")
}
function third() { fourth()
function second() { third() }
function first() { second() }
first();
```

"finaizou - sync"

"finaizou - Microtask Queue"

"finaizou - Task Queue"

Isaac Gomes

JS

Como o Javascript Funciona?

CODE

```
setTimeout(function TaskQueue() {
     console.log("finaizou - Task Queue")
}, 1000);

Promise.resolve()
  .then(function MicrotaskQueue() {
       console.log("finaizou - Microtask Queue")
       });

function fourth() {
  console.log("finaizou - sync")
}
function third() { fourth() }
function second() { third() }
function first() { second() }
first();
```

CALL STACK

fourth

third

second

first

Task Queue

TaskQueue

Microtask Queue

MicrotaskQueue

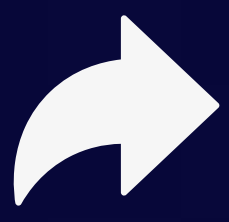
agora nossa imagem começa a fazer sentido

Isaac Gomes

JS

Gostou?

Curta



Compartilhe

Salve

Isaac Gomes