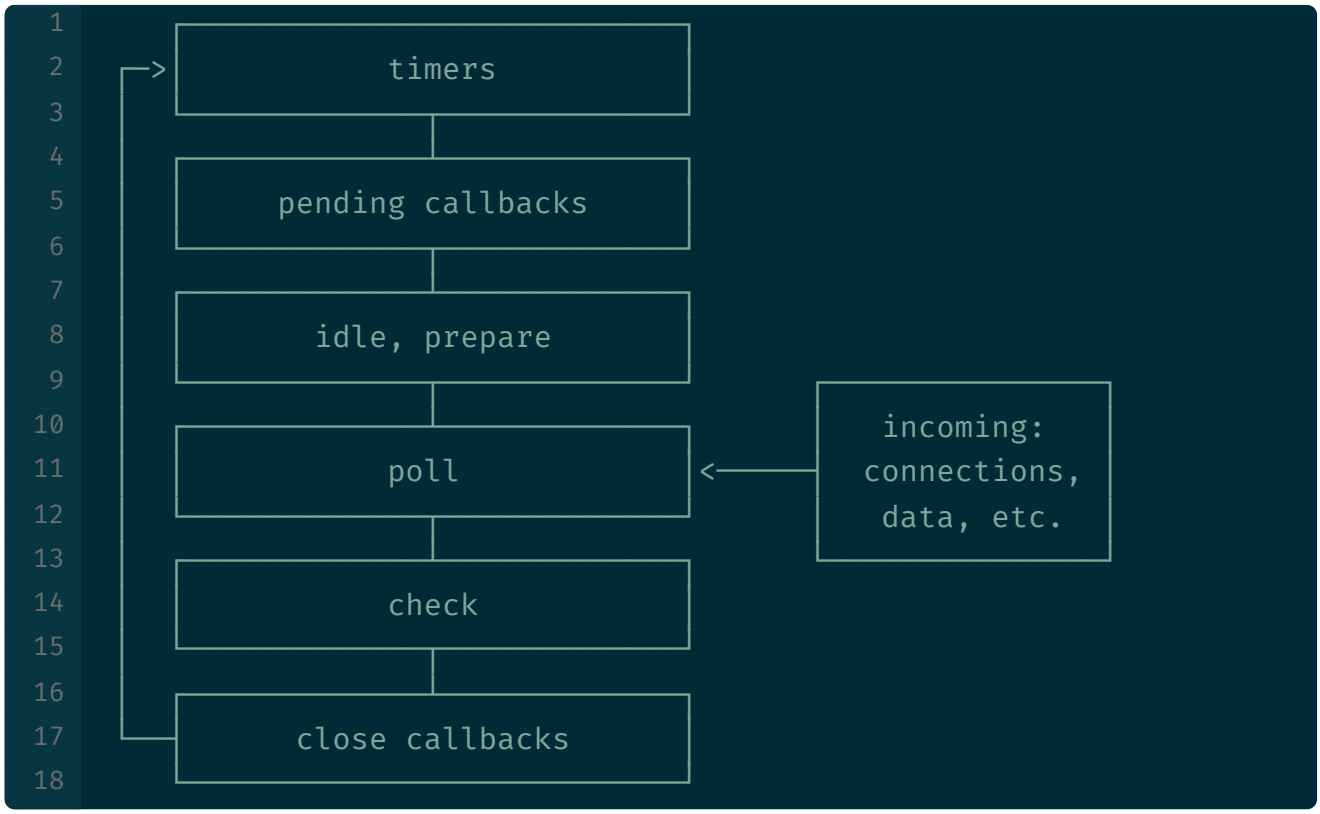
Event Loop en NodeJS

Event Loop

Cuando Node.js arranca, inicializa el *bucle de eventos*, procesa el script de entrada proporcionado (o cae en el REPL) que puede hacer llamadas asíncronas a la API, programar temporizadores, o llamar a *process.nextTick()*, y luego comienza a procesar el bucle de eventos.

El siguiente diagrama muestra una visión simplificada del orden de operaciones del bucle de eventos.



Cada casilla se denominará "fase" del bucle de eventos.

Cada fase tiene una cola FIFO de callbacks a ejecutar. Aunque cada fase es especial a su manera, por lo general, cuando el bucle de eventos entra en una fase determinada, realiza las operaciones específicas de esa fase y, a continuación, ejecuta los callbacks en la cola de esa fase hasta que la cola se haya agotado o se haya ejecutado el número máximo de callbacks. *Cuando la cola se haya agotado o se haya alcanzado el límite de callbacks, el bucle de eventos pasará a la siguiente fase, y así sucesivamente*.

Dado que cualquiera de estas operaciones puede programar más operaciones y que los nuevos eventos procesados en la "poll phase" son puestos en cola por el

núcleo, los eventos del poll pueden ponerse en cola mientras se procesan los eventos del poll. Como resultado, los callbacks de larga ejecución pueden permitir que la "poll phase" se ejecute mucho más tiempo que el umbral de un temporizador.

> Hay una ligera discrepancia entre la implementación de Windows y la de Unix/Linux, pero eso no es importante para esta demostración. Las partes más importantes están aquí. En realidad hay siete u ocho pasos, pero los que nos importan - los que Node.js realmente utiliza - son los de arriba.

Descripción general de las fases

- timers: esta fase ejecuta callbacks programados por setTimeout() y setInterval().
- callbacks pendientes: ejecuta callbacks de E/S diferidos a la siguiente iteración del bucle.
- idle, prepare: sólo se utilizan internamente.
- poll: recupera nuevos eventos de E/S; ejecuta callbacks relacionados con E/S (casi todos con la excepción de close callbacks, los programados por timers, y setImmediate()); el nodo se bloqueará aquí cuando sea apropiado.
- check: aquí se invocan las callbacks de setImmediate().
- close callbacks: algunos callbacks de cierre, por ejemplo socket.on('close', ...).

Entre cada ejecución del bucle de eventos, Node.js comprueba si está esperando alguna E/S asíncrona o temporizadores y se cierra limpiamente si no hay ninguno.

Fases en detalle

Timers

Un temporizador especifica el umbral tras el cual puede ejecutarse un callback proporcionado, en lugar de la hora exacta a la que una persona desea que se ejecute. Las callbacks de los temporizadores se ejecutarán tan pronto como puedan programarse una vez transcurrido el tiempo especificado; sin embargo, la programación del sistema operativo o la ejecución de otras callbacks pueden retrasarlas.

> Técnicamente, la poll phase controla cuándo se ejecutan los temporizadores.

Por ejemplo, digamos que usted programa un timeout para ejecutarse después de un umbral de 100 ms, entonces su script comienza a leer asincrónicamente un archivo que toma 95 ms:

```
1const fs = require('fs');23function someAsyncOperation(callback) {4
// Assume this takes 95ms to complete5  fs.readFile('/path/to/file',
callback);6}78const timeoutScheduled = Date.now();910setTimeout(() =>
{11  const delay = Date.now() - timeoutScheduled;1213
console.log(`${delay}ms have passed since I was scheduled`);14},
100);1516// do someAsyncOperation which takes 95 ms to
complete17someAsyncOperation(() => {18  const startCallback =
Date.now();1920  // do something that will take 10ms...21  while
(Date.now() - startCallback < 10) {22    // do nothing23  }24});
```

Cuando el bucle de eventos entra en la "poll phase", tiene la cola vacía (fs.readFile() no ha finalizado), por lo que esperará el número de ms que faltan para alcanzar el umbral del temporizador mas pronto. Mientras espera que pasen 95 ms, fs.readFile() termina de leer el fichero y su callback, que tarda 10 ms en completarse, se añade a la cola del poll y se ejecuta. Cuando el callback finaliza, no hay más callbacks en la cola, por lo que el bucle de eventos verá que el umbral del temporizador más cercano ha sido alcanzado y entonces volverá a la fase de timers para ejecutar el callback del temporizador. En este ejemplo, verás que el retardo total entre la programación del temporizador y la ejecución de su callback será de 105ms.

> .
>
> Para evitar que la "poll phase" acabe con el bucle de eventos, libuv (la biblioteca C que implementa el bucle de eventos de Node.js y todos los comportamientos asincrónicos de la plataforma) también tiene un máximo estricto (dependiente del sistema) antes de que deje de sondear en busca de más eventos.

Pending callbacks (Callbacks pendientes)

Esta fase ejecuta callbacks para algunas operaciones del sistema, como tipos de errores de TCP. Por ejemplo, si un socket TCP recibe ECONNREFUSED al intentar conectarse, algunos sistemas *nix* desean esperar para informar el error. Esto se pondrá en cola para ejecutarse en la fase de devoluciones de llamadas pendientes.

Poll (Sondeo)

La poll phase tiene dos funciones principales:

1. Calculando cuánto tiempo debe bloquear y sondear E/S, luego
2. Procesando eventos en la cola de sondeo.

Cuando el bucle de eventos entra en la fase de sondeo y NO hay temporizadores programados, sucederá una de dos cosas:

- Si la cola de sondeo no está vacía, el bucle de eventos iterará a través de su cola de *callbacks* y las ejecutará de forma síncrona hasta que se agote la cola o se alcance el límite estricto dependiente del sistema.
- Si la cola de sondeo está vacía, ocurrirá una de dos cosas más:
  - Si las secuencias de comandos han sido programadas por setImmediate(), el bucle de eventos *finalizará la fase de sondeo* y continuará con la *fase de verificación* para ejecutar esas secuencias de comandos programadas.
  - Si setImmediate() no programó scripts, el bucle de eventos *esperará a que se agreguen callbacks a la cola y luego las ejecutará de inmediato*.

Una vez que la cola de sondeo esté vacía, el bucle de eventos comprobará los temporizadores cuyos umbrales de tiempo se hayan alcanzado. Si uno o más temporizadores están listos, el bucle de eventos volverá a la fase de temporizadores para ejecutar callbacks de esos temporizadores.

Check (comprobación)

Esta fase permite que una persona ejecute callbacks inmediatamente después de que se haya completado la "poll phase". Si la "poll phase" queda inactiva y los scripts se han puesto en cola con setImmediate(), el bucle de eventos *puede continuar hasta la fase de comprobación en lugar de esperar*.

setImmediate() es en realidad un temporizador especial que se ejecuta en una fase separada del bucle de eventos. Utiliza una API libuv que programa los callbacks para que *se ejecuten después de que se haya completado la fase de encuesta*.

Por lo general, a medida que se ejecuta el código, el bucle de eventos finalmente llegará a la fase de sondeo, donde esperará una conexión entrante, una solicitud, etc. Sin embargo, si se ha programado un callback con setImmediate() y la fase de sondeo queda inactiva, terminará y continuará con la fase de comprobación en lugar de esperar los eventos de sondeo.

Close callbacks (Callbacks de cierre)

Si un socket o identificador se cierra abruptamente (por ejemplo, socket.destroy()), se emitirá el evento 'close' en esta fase. De lo contrario, se emitirá a través de process.nextTick().

Temas relacionados

Process.nextTick()