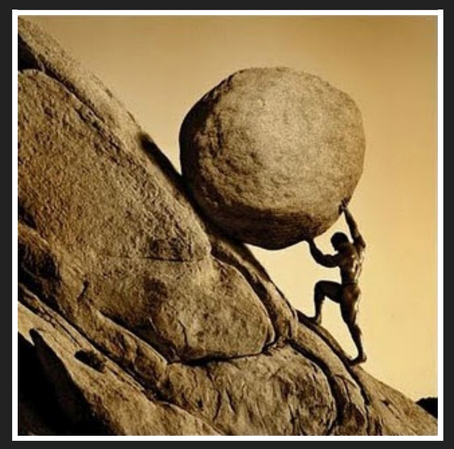
PERSISTENT NOTIFICATIONS



A persistent notification is a notification with an associated service worker registration.

NOTIFICATIONS

REQUESTPERMISSION

```
Notification.requestPermission()
  .then((result) => {

  });
```

PARTY

```
const options = {
    body: 'this is the body of the notification',
    icon: 'icon.png'
}
const n =
  new Notification('the title of the notification', options);
```

WEB WORKERS

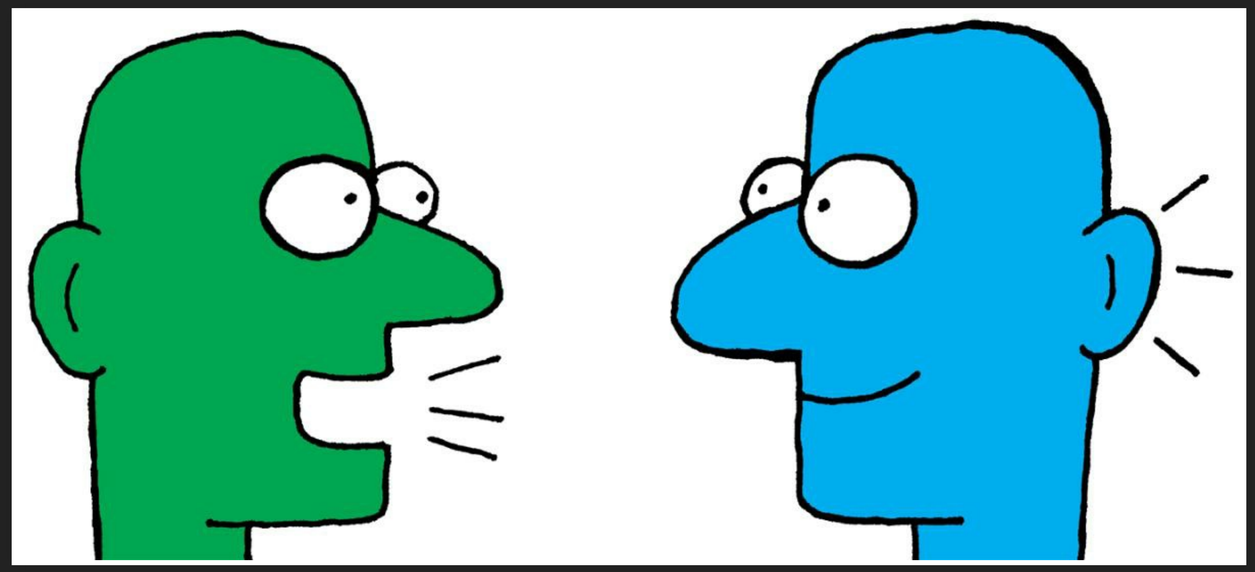
```
const myWorker = new Worker("worker.js");
```

WORKER SCOPE != WINDOW

workers run in another global context that is different from the current window.

Functions and classes available to Web Workers

COMMUNICATING



FROM CLIENT TO WORKER

```
// client
myWorker.postMessage();
```

```
// worker
onmessage = (e) => {

}
```

FROM WORKER TO CLIENT

```
// client
myWorker.onmessage = (e) => {

}
```

```
// worker
self.postMessage();
```

Data passed between the main page and workers is copied, not shared. Objects are serialized as they're handed to the worker, and subsequently, de-serialized on the other end. The page and worker do not share the same instance, so the end result is that a duplicate is created on each end. Most browsers implement this feature as structured cloning.

TERMINATE WORKER

FROM CLIENT

```
myWorker.terminate();
```

FROM WORKER

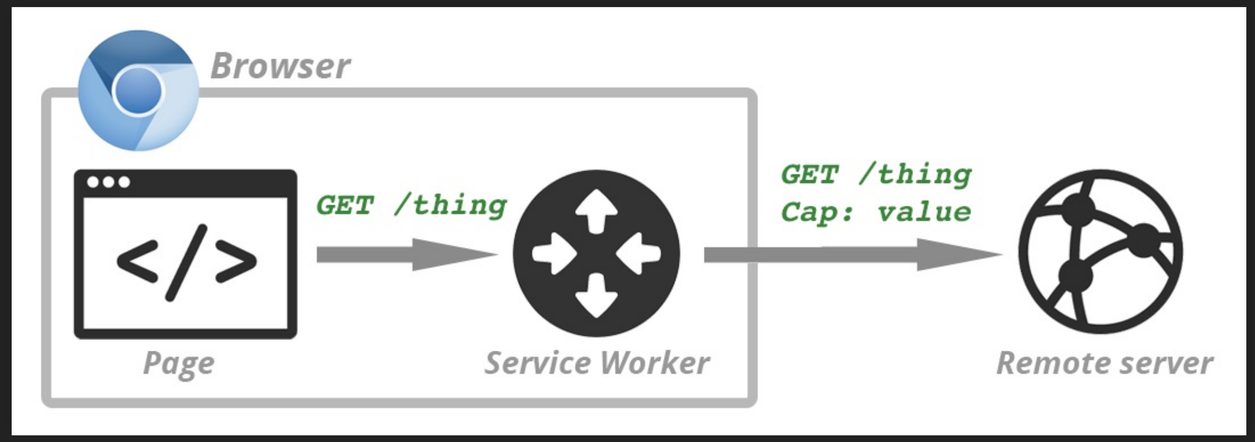
```
close();
```

TYPES OF WEB WORKERS

- Dedicated worker
- Shared worker
- Service worker
- Audio worker

SERVICE WORKERS

Service workers essentially act as proxy servers that sit between web applications, and the browser and network (when available). They are intended to (amongst other things) enable the creation of effective offline experiences, intercepting network requests and taking appropriate action based on whether the network is available and updated assets reside on the server. They will also allow access to push notifications and background sync APIs.


Browser
GET /thing
GET /thing
Cap: value
Page
Service Worker
Remote server

The user agent may terminate service workers at any time it has no event to handle or detects abnormal operation such as infinite loops and tasks exceeding imposed time limits, if any, while handling the events.

NOTIFICATIONS IN SERVICE WORKER

SETTIMEOUT

```
setTimeout(() => {
  self.registration.showNotification('notification title', {
    body: 'notification body'
  });

}, 1000);
```

WEBSOCKET

```
socket.onopen = (event) => {
  socket.onmessage = (event) => {
    if (event.data !== 'check') {
      self.registration.showNotification(event.data);
    } else {
      console.log(new Date());
    }
  }
};
```

PUSH NOTIFICATIONS