Quota Overriding In DevTools

Attention: Externally visible, non-confidential
Author: jarrydg@chromium.org
Status: Inception | Draft | Accepted | Done
Created: 2020-05-05 / **Last Updated**: 2020-11-05

One-page overview

As of Q3 2019, web apps now have the ability to write to up to 60% of total disk space. This gives a lot of power to the web platform in that apps can now start making use of non-trivial amounts of storage to drive offline applications. However, 60% of disk space is wildly different on a low-end mobile device and a workstation with a multiple TB drive. **This makes web app behavior a bit unpredictable for the developer, who would be unsure how their app would behave on a device with much different storage capacity.**

The proposed feature is an addition to the Clear Storage section of the Application tab in DevTools, in which developers could specify an override value, in MB, for the storage quota. This would allow developers to temporarily simulate behavior of another device, or simulate storage pressure (the state of having low available disk space), helping them to provide a smooth experience for all their users.

Summary

Quota is the system which sets and enforces limits on disk usage by the browser and web apps. This feature will give developers the ability to simulate different devices and test the behavior of their apps in low disk availability scenarios.

Platforms

All

Team

storage-dev@chromium.org

Tracking issue

https://crbug.com/945786

Value proposition

Give developers the ability to simulate being on devices with different storage capacities for storage-based testing.

Code affected

DevTools front-end and back-end

Signed off by

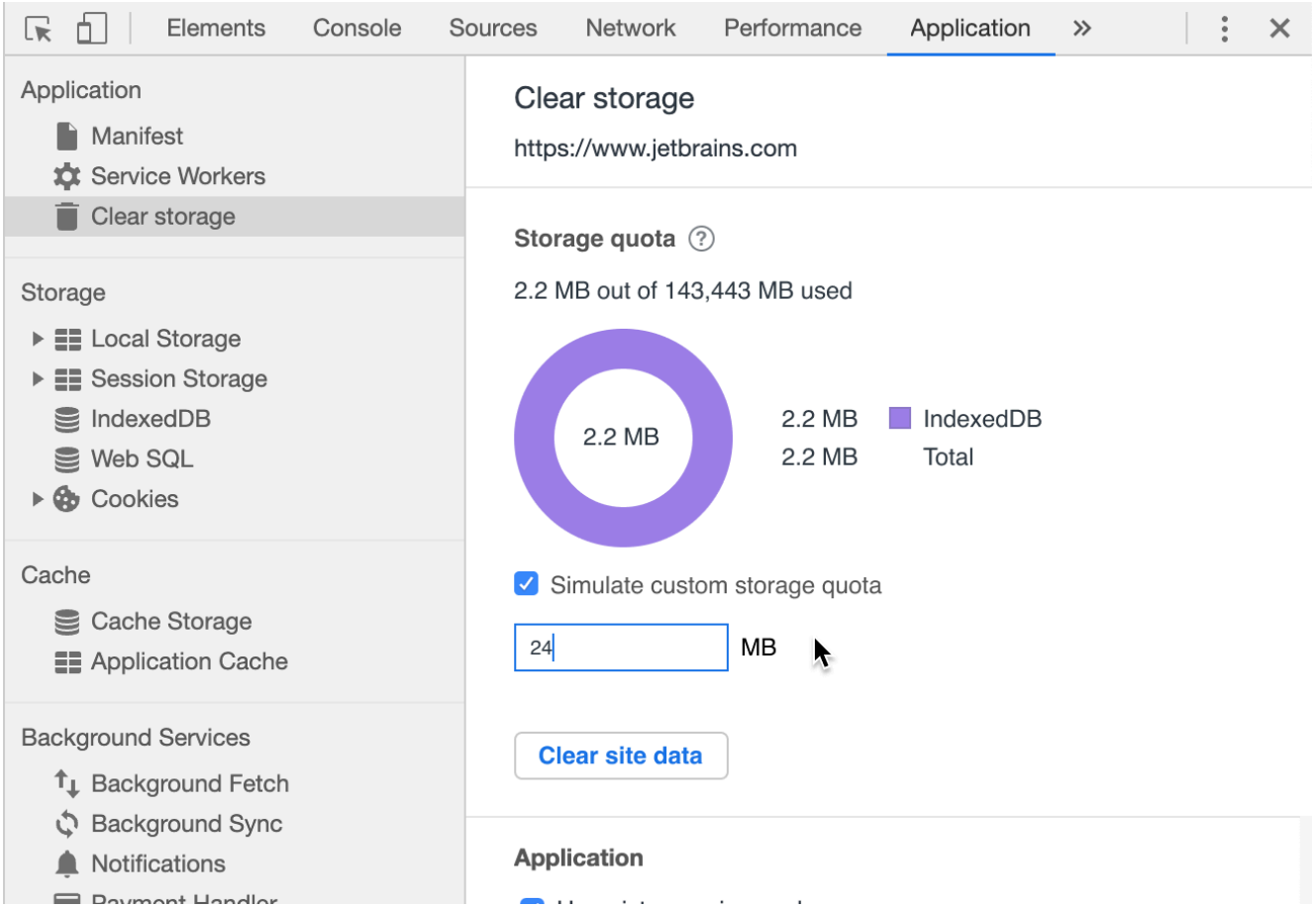
Name	Write (not) LGTM in this row
bmeurer@chromium.org	LGTM
rob.paveza@microsoft.com	LGTM
petermarshall@chromium.org	LGTM
petermueller@chromium.org	LGTM
caseq@chromium.org	LGTM*
sigurds@chromium.org	LGTM if caseq@ can resolve the remaining issues

Core user stories

- 2 CDP sessions (DevTools instances) set a quota for the same origin. Session A sets a quota of 1, session B sets a quota of 10 afterwards. The current quota is now 10 because it is the last override that arrived.
    - If either session disconnects, the quota remains 10.
    - If either session disables the quota, the quota is removed and goes back to the default quota
- Multiple sessions for a single origin will stay in sync. Session A is started and sets a quota of 10, before Session B starts. Session B starts and it polls for quota, and is informed that there is an active quota override, but will not hold onto this override.
    - If session A disconnects, quota should be reset to the default quota
- A CDP session sets quotas for several origins and then disconnects. The quotas are removed for all overridden origins.
- A CDP session sets the quota for the same origin first to 10 then to 100, without waiting for a reply in between. After both messages are processed the quota is 100.

A developer of a web app that uses non-trivial amounts of storage would like to test what their app behaves like under storage pressure currently has to fill up their working storage partition. The easiest way to do this is probably using RAMDisk to set up a small (in-memory) storage partition, and launch Chrome with a blank profile directory located in the aforementioned partition. With a quota override in DevTools, developers can simply input the max quota they'd like to test. The same applies for developers that want to test the behavior on devices of varying storage capacities. This is similar to the network throttling feature, which allows developers to test their app while simulating network conditions that might match that of their users.

Design



A new checkbox will allow users to set a custom storage quota.

UX Deck

Assets

UI Specs

The feature will be added to the `Application` panel, under the `Clear storage` section. It will be hidden by default, but a checkbox labeled “Simulate custom storage quota” will reveal an input field for users to enter in a non-negative number (Bytes) to set as the quota override.

Web platform storage is origin scoped and quota is issued on a per-origin basis. Due to this, a storage quota override will need to be enacted on all open tabs and service workers for a given origin, as long as at least one DevTools session has an active override for that origin. Without this, storage activity on one tab could leave another tab in an anomalous state (i.e. put an origin above the overridden quota value in another tab). A storage quota override will remain active until it is explicitly disabled by the user (uncheck the box) or the last DevTools instance is disconnected.

The `Clear storage` section will be renamed to `Storage`, so the menu title and headline will both be updated. The icon in the navigation pane (trash can) will also be replaced.

- `StorageHandler` will be refactored such that all access to `QuotaManager` will be routed through `QuotaManagerProxy`. Access to `QuotaManager` will need to be asynchronous, and it is possible for there to be a race. The reason QuotaManager access will need to be async is because the Quota system is being split out into the Storage Service running on its own process, and `QuotaManager` will become a mojo interface. QuotaManagerProxy can be called from any thread, and routes calls to QuotaManager on the IO thread. This, in combination with the `SequenceChecker` on `QuotaManager` will ensure that we do not have any race conditions. For simplicity, this document will refer only to `QuotaManager`, but the reader can safely assume that all access will be routed through `QuotaManagerProxy`.
- Every storage handler should have a unique identifier with which all quota override requests to the quota manager are keyed. If the storage handler goes away (as result of the CDP session being disconnected), it signals to the quota manager that it withdraws and only provides this unique identifier.
- With this design, the storage handler doesn't need to know whether it currently holds an override: **The quota manager is the single source of truth for that**. All the storage handler needs to know is whether it ever (tried to) set an override during its lifetime. In this case the storage handler must send the withdrawal signal to the quota manager once the corresponding session disconnects.
- The interface on `QuotaManager` for overriding quota must require a unique id. The `QuotaOverrideHandle` will wrap an integer that is obtained from an integer member of `QuotaManager`. This number is used to uniquely identify a client that wants to override a quota. The `QuotaOverrideHandle` would then represent the connection between the `StorageHandler` and the `QuotaManager`, and not a single override as it currently does. If the `QuotaOverrideHandle` is destroyed, it posts a withdrawal task to `QuotaManager`.
- Every `StorageHandler` has a `QuotaOverrideHandle` member. This handle will withdraw any overrides associated with its owning `StorageHandler` when it is destructed. Attempts to call `OverrideQuotaForOrigin()` before a StorageHandler has a QuotaOverrideHandle will be put into a callback queue. This is to make sure that StorageHandler has a unique id to call `OverrideQuotaForOrigin()` with.
- `QuotaManager` tracks for each origin:
  - the current quota override value
  - a set of ids holding on to the quota. These are the ids that are wrapped by `QuotaOverrideHandle` and have a 1:1 mapping with CDP sessions that have attempted to set an override.
- If `OverrideQuotaForOrigin()` is called with kDisable, then origin is removed from the overrides, no matter how many clients are holding on to the override.
- `QuotaManager` provides a `WithdrawOverrides()` method, which removes the unique id in the handle from all origins, and drops overrides if the set of override holders becomes empty. This message must be posted if the storage handler goes away due to a disconnect.

Classes

QuotaManager

This is the top level class in the quota system. This is the class that is queried for a quota. In this change, `QuotaManager` will be modified to keep track of quota overrides.

```
struct QuotaOverride {
  QuotaOverride(int64_t size) :
    quota_size(size) {}


  int64_t quota_size;
  url::Origin origin;
  std::set<int> active_override_session_ids;
}

enum class QuotaOverrideAction {
  kEnable,
  kDisable
}

// Map from origin to quota size and the count of
// DevTools clients currently holding a quota override.
std::map<url::Origin, QuotaOverride> devtools_overrides_
int next_override_handle_id_ = 0;

std::unique_ptr<QuotaOverrideHandle> QuotaManager::GetOverrideHandle() {
  return std::make_unique<QuotaOverrideHandle>(++next_override_handle_id_,
                                               quota_manager_proxy_);
}

void QuotaManager::OverrideQuotaForOrigin(
    const int handle_id, const url::Origin& origin, int64_t quota_size,
    QuotaOverrideAction action_mode) {
  if (action_mode == kEnable) {
    DCHECK_GE(next_override_handle_id_, handle_id);
    if (!base::Contains(devtools_overrides_, origin)) {
        devtools_overrides_[origin] = QuotaOverride(quota_size, origin);
    }
    devtools_overrides_[origin].active_override_session_ids.insert(handle_id);
    return;
  }
  else if (action_mode == kDisable) {
    devtools_overrides_.erase(origin)
  }
}

void QuotaManager::WithdrawOverrides(int handle_id) {
  for (auto& quota_override : devtools_overrides_) {
    quota_override.active_override_session_ids.erase(handle_id);
    if (!quota_override.active_override_session_ids.size())
      devtools_overrides_.erase(quota_override.origin);
  }
}
```

QuotaManagerProxy

```
void GetOverrideHandle(base::SequencedTaskRunner* original_task_runner,
    base::OnceCallback<void(std::unique_ptr<QuotaOverrideHandle>)> cb) {
```

```
  if (!io_thread->BelongsToCurrentThread()) {
    io_thread->PostTask(FROM_HERE,
                        base::BindOnce(&QuotaManagerProxy::GetOverrideHandle, this,
                        base::RetainedRef(original_task_runner), std::move(cb)));
    return;
  }
  DidGetOverrideHandle(manager_->GetOverrideHandle(),
                       base::RetainedRef(original_task_runner), std::move(cb))
}

void DidGetOverrideHandle(std::unique_ptr<QuotaOverrideHandle> handle,
    base::SequencedTaskRunner* original_task_runner,
    base::OnceCallback<void(std::unique_ptr<QuotaOverrideHandle>)> cb) {
  if (!original_task_runner->RunsTasksInCurrentSequence()) {
    original_task_runner->PostTask(FROM_HERE, base::BindOnce(
        &DidGetOverrideHandle, base::RetainedRef(original_task_runner), std::move(cb)));
    return;
  }

  std::move(cb).Run(std::move(handle));
}

void OverrideQuotaForOrigin(const int handle_id, const url::Origin& origin,
                            int64_t quota_size, QuotaOverrideAction action_mode,
                            base::SequencedTaskRunner* original_task_runner,
                            base::OnceClosure callback) {
  if (!io_thread->BelongsToCurrentThread()) {
    io_thread->PostTask(FROM_HERE,
                        base::BindOnce(
                            &QuotaManagerProxy::OverrideQuotaForOrigin, this,
                            handle_id, origin, quota_size, action_mode,
                            base::RetainedRef(original_task_runner),
                            std::move(callback)));
    return;
  }
  manager_->OverrideQuotaForOrigin(handle_id, origin, quota_size, action_mode)
  DidOverrideQuotaForOrigin(base::RetainedRef(original_task_runner),
                            std::move(callback));
}

void DidOverrideQuotaForOrigin(SequencedTaskRunner* original_task_runner,
                               base::OnceClosure callback) {
  if (!original_task_runner->RunsTasksInCurrentSequence()) {
    original_task_runner->PostTask(FROM_HERE,
        base::BindOnce(&DidOverrideQuotaForOrigin,
                       base::RetainedRef(original_task_runner), std::move(cb)));
    return;
  }

  std::move(cb).Run();
}

void WithdrawOverrides(const int handle_id) {
  if (!io_thread->BelongsToCurrentThread()) {
```

```
    io_thread->PostTask(FROM_HERE,
                        base::BindOnce(
                            &QuotaManagerProxy::WithdrawOverrides, this, handle_id));
    return;
  }
  manager_->WithdrawOverrides(handle_id)
}
```

QuotaOverrideHandle

`QuotaManager::GetOverrideHandle()` will return a `QuotaOverrideHandle` to `StorageHandler` so that `StorageHandler` has a unique id to identify it's override requests. `QuotaOverrideHandle` also serves to release all of its associated quota overrides from `QuotaManager` in its destructor, freeing the DevTools backend from the burden of managing that state.

```
class QuotaOverrideHandle {
 public:
  QuotaOverrideHandle(scoped_refptr<QuotaManagerProxy> proxy) :
      quota_manager_(proxy) {};
  ~QuotaOverrideHandle() {
    WithdrawOverrides();
  }

  void OverrideQuotaForOrigin(
      const String& origin_string,
      double quota_size, const QuotaOverrideAction action_mode,
      std::unique_ptr<OverrideQuotaForOriginCallback> callback) {
    quota_manager_->OverrideQuotaForOrigin(id_, origin_string, quota_size,
                                           action_mode, std::move(callback));
  }


  void WithdrawOverrides() {
    quota_manager_->WithdrawOverrides(this.id_);
  }

 private:
  scoped_refptr<QuotaManagerProxy> quota_manager_;
  int id_;
}
```

StorageHandler

This class is responsible for communicating storage related information between DevTools and storage systems. `StorageHandler` runs on the Browser process, and there is one instance of this class per DevTools client.

```
 private:
  std::unique_ptr<QuotaOverrideHandle> quota_override_handle_;
  std::vector<base::OnceClosure> override_callback_queue_;
  bool is_fetching_handle_ = false;

// Take action mode string from frontend and convert to backend type
```

```
storage::QuotaOverrideAction QuotaOverridingActionModeEnumToMode();


void StorageHandler::GetOverrideHandle() {
  DCHECK(!quota_override_handle_);
  if (is_fetching_handle_)
    return;

  is_fetching_handle_ = true;
  scoped_refptr<QuotaManagerProxy> manager_proxy =
      storage_partition_->GetQuotaManager()->proxy_();
  manager_proxy->GetOverrideHandle(base::ThreadTaskRunnerHandle::Get().get(),
      base::BindOnce(&StorageHandler::DidGetOverrideHandle, this);
}

void StorageHandler::DidGetOverrideHandle(std::unique_ptr<QuotaOverrideHandle> handle) {
  quota_override_handle_.reset(handle);
  is_fetching_handle_ = false;

  for (auto& callback : override_callback_queue_) {
    callback.Run();
  }
}

void StorageHandler::OverrideQuotaForOrigin(
    const String& origin_string,
    double quota_size, const String& action_mode,
    std::unique_ptr<OverrideQuotaForOriginCallback> callback) {
  // input validation
  // massage params from DevTools frontend
  url::Origin origin = url::Origin::Create(GURL(origin_string));

  if (!quota_override_handle_) {
    GetOverrideHandle();
    override_callback_queue_.push_back(base::BindOnce(
        &OverrideQuotaForOrigin, this, origin_string, quota_size, action_mode,
        std::move(callback));
    return;
  }

  quota_override_handle_->OverrideQuotaForOrigin(
      origin, quota_size, action_mode,
      base::ThreadTaskRunnerHandle::Get().get(),
      base::BindOnce(&StorageHandler::DidOverrideQuotaForOrigin, this,
                     std::move(callback);
}

void StorageHandler::DidOverrideQuotaForOrigin(
    std::unique_ptr<OverrideQuotaForOriginCallback> callback) {
  callback.sendSuccess();
}

Response StorageHandler::Disable() {
 ...
```

```
  quota_override_handle_.reset();
}
```

Protocol Functions

`overrideStorageQuotaForOrigin()` - Enable or disable quota override from a DevTools session for an origin. This call will be plumbed through to the corresponding `QuotaManager` function and pass the 3 parameters along. To add this, I will follow the *Adding a Protocol Method* section in the *Contributing to Chrome DevTools Protocol* document.

Subsequent calls to this function (from the same or different protocol clients) will override each other. This means that protocol clients will be responsible for making sure that the storage quota override value displayed is up-to-date.

- (string) `origin`
- (number) `quotaSize` - the override size, in Bytes. This value must be a non-negative number. If the value is smaller than the the amount of storage currently used by the app, the quota system will treat this the same as if the app just reached its quota limit -- no data will be deleted but any further attempts to write will be met with quota exceeded errors
- (enum) `actionMode`
    - enable - enables the quota override
    - disable - resets the quota to the default value

`getUsageAndQuota()` - The return value will need to be updated to pass along a boolean that represents whether or not an override is currently active. This will tell the DevTools client about the quota override state for this origin, which could have been manipulated by another client. Since this function is called in `update()` (which is fired every second while the Storage section is open), it will take at most 1 second for a DevTools client to sync with `QuotaManager`.

Caveats and Alternatives

Content Setting

The possibility of adding controls to chrome://settings/content was considered. The following is a possible design for said controls:

A Storage label would be added to the top of SiteDetails page (ie chrome://settings/content/siteDetails?site=https%3A%2F%2Fwww.google.com%2F) that would contain two sub-labels: Usage (which already exists on the page) and Quota. Under Quota, we could have two radio buttons, Automatic and Manual -- if Manual was selected, the user would have a slider or text box to control the quota override.

The benefit of having a user-exposed control is that end-users would have an easier time reducing quota for misbehaving sites in settings pages than from DevTools.

One of the downsides of this alternative is that this feature would not live next to similar features, which are in DevTools, such as network throttling to simulate various network conditions, or the device toolbar which allows developers to simulate different device screen sizes. Another downside is that this would require more engineering effort. It would require effort to come up with a solution that clears the bar for user-exposed UI. This solution would also require some degree of confidence

that the feature provides enough value to users to justify the added space taken up in content settings.

Reference Count Active Overrides in QuotaManager

In the original design, `QuotaManager.devtools_overrides_` was of type `std::map<url::Origin, std::pair<int64_t, int>>` (the difference being the std::pair, storing an `int` as the value instead of a set of `int`). Originally, `devtools_overrides_` would just keep track of the number of DevTools clients that had an active override which would need to be incremented and decremented by DevTools backends, but this was switched to keep track of the IDs for devtool sessions that have an active override. It was decided that the original implementation was too brittle, and would be difficult to debug. There are several classes of bugs that can arise out of `QuotaManager` keeping track of quota overrides:

- double acquisition
- double release
- forgetting to acquire
- forgetting to release

A switch away from reference counting to using explicit IDs eliminated the possibility of double acquisition or double release bugs, since we won't be storing a single ID more than once. In order to address the other two bugs, `StorageHandler::ScopedQuotaOverride` would register with `QuotaManager` in it's constructor, and unregister the override in it's destructor.

The problem with this was that callers of `QuotaManager::OverrideQuotaForOrigin()` would need to remember to acquire and release handles, and in order to know that, the reader/maintainer needs context on both the DevTools side and the quota side. It was decided that we should move to the current design, in which we reverted to using reference counting in `QuotaManager`, but have the reference counts automatically managed by a handle class, `QuotaOverrideHandle`. The constructor and destructor of this class will handle updates to `QuotaManager`'s state, so that any future authors of code that calls into the quota override code mustn't remember to do this themselves.

Multiple Overrides for a Single Origin

Consider consecutive calls to override quota for a single origin, each from distinct DevTools sessions. In the current design, if the DevTools session that made the first call to override the quota, the origin's quota will be reset to the default quota value assigned by `QuotaManager`. Originally, we had considered an alternative where the overridden quota value would be held onto by the other active DevTools sessions, but decided on the current approach because it significantly reduces the implementation complexity.

This behaviour could potentially lead to developer confusion. We can modify the behavior later on if this user story turns out to require a different behavior. Note that even if several clients are connected, it is easily possible to remove an override by disabling it.

AtomicSequenceNumber

An earlier design had considered using an `AtomicSequenceNumber` member of `QuotaManager` to ensure unique ids could be passed out in a thread-safe manner. In the comments for AtomicSequenceNumber, it says "AtomicSequenceNumber is a thread safe increasing sequence

number generator." However, QuotaManager is currently not thread-safe and has sequence checkers. We concluded that AtomicSequenceNumbers is incompatible with the direction QuotaManager is moving. The future direction of QuotaManager imposed by the Storage Service means that communication with QuotaManager will need to become a process hop eventually. In said future, an `AtomicSequenceNumber` in `QuotaManager` would not work.

One proposal suggested using AtomicSequenceNumber in the meantime, before QuotaManager moves out-of-process. The position of the quota owners was that they do not want to commit `QuotaManager` to implementing thread-safe methods. As described in the `QuotaManager` class comment, "The constructor and proxy() methods can be called on any thread. All other methods must be called on the IO thread." QuotaManager is already too complex and there have been bugs in it, so the team was not willing to modify this policy because it would add a non-trivial amount of complexity for an optimization that may not be necessary.

The alternative proposed was to have requests go through QuotaManagerProxy, which calls into `QuotaManager` exclusively on the IO thread. With all calls on the IO thread, `QuotaManager` can use a private int member to keep track of the next ID and dole out IDs on a single thread, which would mean there wouldn't be any race.

Rollout plan

Waterfall

Core principle considerations

Speed

This feature is not expected to have an impact on performance for end-users. Most end-users will not use this feature, however it is a major improvement to speed of the feedback cycle for developers testing under different storage conditions. For those users, there is the possibility that the override has some performance effects on the storage systems, for which there are monitored performance metrics.

Security

This does not introduce any new attack vectors. The feature can only be interacted with by the end-user, and the Chrome threat model assumes no hostile physical access to the device.

Simplicity

The feature will, by default, be hidden behind a checkbox. This is to reduce clutter on the `Clear storage` section and keep a simple UI. QuotaManager will enforce that any quota related work will be done on the IO thread, and thus the UI will never be unresponsive.

Accessibility

I plan to work with the DevTools team and reuse UI components that are already in line with our accessibility guidelines and best practices.

Testing plan

I plan on working with the DevTools team to pair program tests.

Followup work

Sessions without a QuotaOverrideHandle

Consider a user story described above:

- Multiple sessions for a single origin will stay in sync. Session A is started and sets a quota of 10, before Session B starts. Session B starts and it polls for quota, and is informed that there is an active quota override, and will hold onto this override.
    - If session A disconnects, quota should be reset to the default quota

It is possible that this may lead to a confusing experience for developers. If the team receives feedback in this vein, consider modifying this such that the following behavior would replace that of the current design:

- If session A disconnects, quota will not be reset to the default quota, as Session B still is holding onto this override.

Appendix

- Contributing to Chrome DevTools Protocol
- Chrome DevTools Contribution Guide