BiDi Serialization in CDP

This Document is Public

Authors: sadym@chromium.org
Short Link: go/bidi-serialization

UPD 2022-05-09

The suggested changes were landed in https://crrev.com/c/3472077, https://crrev.com/c/3596173, https://crrev.com/c/3472491

Signed off by

Name	**Write LGTM (or not) in this column**

Alex Rudenko <alexrudenko@chromium.org>	LGTM 4.1
Benedikt Meurer <bmeurer@chromium.org>	LGTM
Andrey Kosyakov <caseq@chromium.org>	LGTM 4.1, though we need better names, but this can be left to code review phase.
Danil Somsikov <danilsomsikov@chromium.org>	LGTM
Philip Jägenstedt <foolip@chromium.org>	LGTM with extra enthusiasm for 4.1
Mathias Bynens <mathias@chromium.org>	LGTM 4.1

One-page overview

Why?

To implement WebDriver BiDi protocol (aka BiDi) using CDP, value serialization should be implemented. Currently, there is no way to provide serialization for non JSON-serializable objects in CDP. As WebDriver BiDi is going to be implemented using CDP, there is a need for such a serialization.

What?

We propose to add a new method to the V8InspectorClient called bidiSerialize. This would allow a V8 embedder to implement a custom serialization logic for embedder-specific value (e.g. Node or Element in case of Chrome).

V8 would implement default BiDi-specific serialization logic for the native V8 objects (e.g. primitive, object, maps, sets). It would also consult the embedder via the V8InspectorClient interface to see if the embedder wants to provide a custom serialization. If the embedder returns value, it would be used, otherwise a default V8 serialization takes place.

See prototype CLs at https://crrev.com/c/3472077 + https://crrev.com/c/3472491

These do the following:

1. Adds an additional param `addSerializedValue` + `serializationMaxDepth` to `Runtime.callFunctionOn`, Runtime.evaluate etc, and an additional field `serializedValue` to the `Runtime.RemoteObject` serializing V8, DOM and any other required domain's objects.
2. Adds a virtual bidiSerialize method to the V8 inspector client, which can be implemented by the embedder (Blink, NodeJS, etc).

Serialization process

When a serialized value for the given Object is requested, the following steps are done by the V8 Inspector:

1. Call the Object's bidiSerialize. If custom serialization is implemented, consider its result as a serialized value.
2. Otherwise, the Object is considered to be a V8 object, and is serialized by V8 Inspector recursively, respecting the `serializationMaxDepth` (according to BiDi serialization specification).

Performance	**Good**	1 extra call in case of `console.log`
Trustworthiness	**Good**	
BiDi compatibility	**Good**	
CDP compatibility	**Good**	

Pros:

- Addresses all the main concerns
- Provides a trust-worthy way to serialize objects in CDP.

Cons:

- (weak, negotiable) Additional call is needed in case of the object received from the console.log event.
- (conceptual), the `Runtime` method returns DOM specific data, which can be confusing/misleading.

Scenarios

The scenarios below are BiDi scenarios where serialization might be needed:

1. Node assertion.

In a UI test, the user receives a node and asserts its state (some specific attributes, having specific children etc).

2. `console.log` events.

User listens to the console.log events and verifies there are no errors.
User wants to get a log message preview for debugging purposes.

3. Custom script.

User runs a custom script which returns some custom data (e.g. page's state).

Context

RemoteObject

Currently, there is a concept ot Runtime.RemoteObject, which can contain either objectId or, in case of primitive values or JSON values, value.
The RemoteObject can be a result of `Runtime.evaluate`, `console.log(..args)` etc.
The RemoteObject can be:

- An EcmaScript primitive (number, string);
- Custom (not necessarily JSON-serializable) instance (object, function etc);
- DOM V8 (1), (2) representation;
- Something else?

Requirements

(weak) Performance	The implementation should not create too much performance overhead
Trustworthiness	The serialization results should be reliable and even malicious pages should not be able to simulate unexpected results, e.g. by manipulations with constructors.
BiDi compatibility	● The BiDi serialization should be implementable. ● The serialization should share the same `objectId`s in BiDi and CDP.
CDP compatibility	The implementation should not break CDP concepts and should be implementable.

Alternatives considered

1. Serialization in a target context

Currently implemented in the Mapper.

Performance	**Good**	
Trustworthiness	**Poor**	Object’s constructor can be faked by the page.
BiDi compatibility	**Poor**	No way to provide the real CDP objectId.

CDP compatibility	**Good**	

Pros:
- No changes in CDP needed.

Cons:
- No way to make it trustworthy.
- No way to share `objectId`s.

2. Step-by-step using existing `` `Runtime.callFunctionOn(returnByValue=true)` ``

Extension of the previous Serialization in a target context approach, but solving the issue with ObjectId.

Performance	**Poor**	Each nested item requires 2 calls: to get an `` `objectId` `` and to get a serialized value.
Trustworthiness	**Poor**	Object's constructor can be faked by the page.
BiDi compatibility	**Moderate**	Race conditions between consecutive CDP calls.
CDP compatibility	**Good**	

Pros:
- No changes in CDP needed.

Cons:
- No way to make it trustworthy.
- Need for 2 CDP calls for each nested instance creates significant performance overhead.
- Race condition between consecutive CDP calls,

3. Implement `[DOM/DOMDebugger].describeObject(objectId)`

Proof of concept: https://crrev.com/c/3440218
Add a new method to some domain aware of DOM and V8, like DOM or DOMDebugger. E.g. `` `DOMDebugger.describeObject(objectId)` `` which serializes the provided object according to the BiDi serialization spec. Serialization requires an additional CDP call.

Performance	**Good**	Can be done in 1 additional CDP call

Trustworthiness	**Good**	
BiDi compatibility	**Good/Moderate**	Potential change between the 2 CDP calls, where the object returned from Runtime.evaluate (or log event) could change before being serialized.
CDP compatibility	**Moderate**	Conceptually mixing V8 and Blink.

Pros:
- Fully compatible with BiDi.

Cons:
- (weak) Having the logic implemented in DOMDebugger makes the approach not usable when debugging NodeJS.
- (weak) Additional serialization call is needed (compared to the next option).
- Inherits the BiDi serialization specific:
    - Mixing V8 and Blink instances.
    - BiDi supports only a subset of the Blink (1), (2) objects.

4. Add V8-only `addSerializedValue` param to `Runtime.callFunctionOn`, `evaluate` etc

Add an additional param `addSerializedValue` to `Runtime.callFunctionOn`, Runtime.evaluate etc, and an additional field `serializedValue` to the `Runtime.RemoteObject` serializing V8 objects and marking DOM and other API objects as `apiObject`.

Performance	**Good**	1 call to evaluate and serialize
Trustworthiness	**Good**	
BiDi compatibility	**Poor**	The approach doesn't support DOM serialization.
CDP compatibility	**Good**	

Pros:
- Can be reused in NodeJS.

Cons:
- Not fully BiDi compatible. DOM objects are not supported. Trick with `isolatedWorld` can be used for node serialization, but not for other DOM instances (eg Window).
- (weak, as it doesn't seem negotiable) Additional call is needed in case of the object received from the console.log event.

4.1 (recommended) Add `addSerializedValue` param to `Runtime.callFunctionOn` based on embedder-implemented serializer

Described in "What?" section.

Open questions

1. ~~In approach 4, can Runtime provide DOM trustworthiness without having knowledge about Blink? E.g. is there a way to recognize if the instance's constructor is a native or user-defined during serialization? Maybe somewhere in objectToProtocolValue?~~
- ~~Reply from Yang Guo :~~
  ~~Within V8 you can detect whether an object was created to back a DOM object by checking its instance type. Something like here: https://source.chromium.org/chromium/chromium/src/+/main:v8/src/objects/value-serializer.cc;l=567;drc=91318efd060e4164810264f47cd31b3fc7159d68 we use WriteHostObject here to call out to Blink to serialize such an object in the value-serializer, which implements message passing and structured clone~~
- ~~Comments by Benedikt Meurer :~~
  ~~https://source.chromium.org/chromium/chromium/src/+/main:v8/include/v8-inspector.h;bpv=1;bpt=1?q=v8inspectorclient and make smth like `descriptionForValueSubtype`~~
  https://crrev.com/c/3472077 + https://crrev.com/c/3472491
2. Should the serialized `objectId` still be a referenceId, or should a consistent ObjectId be provided?