Private brand checks implementation notes

marja@ - January 2021 - Public & non-confidential

Proposal: https://tc39.es/proposal-private-fields-in-in/

Building blocks

This proposal builds on top of the “Class fields” proposal (spec) and “Private methods” proposal (spec) which are both Stage 3 and not merged to the main spec.

Class fields proposal (background info)

Adds the Private Name specification type. It has one slot, [[Description]].

Adds syntax for PrivateIdentifier: # IdentifierName and productions like MemberExpression: MemberExpression . PrivateIdentifier.

Adds MakePrivateReference which creates a Reference for a private name reference (note: always strict). The runtime semantics for the productions such as “MemberExpression . PrivateIdentifier” use it.

Adds PrivateEnvironment: “The Lexical Environment for Private Names that the function was closed over.” MakePrivateReference creates a reference which binds to it.

Adds static semantics AllPrivateIdentifiersValid. Takes “names” as an argument; for the productions which have PrivateIdentifiers, return false if the identifier is not in “names”.

Adds static semantics PrivateBoundIdentifiers. Contains the private identifiers which are defined inside a class.

Defines PrivateFieldFind, PrivateFieldAdd, PrivateFieldGet, PrivateFieldSet. These work based on a list of entries stored in the [[PrivateFieldValues]] internal slot. Each entry has [[PrivateName]] (pointer to the Private Name object) and [[PrivateFieldValue]].

Every object now has the [[PrivateFieldValues]] slot.

Extends [[Construct]] internal method on function objects: It calls InitializeInstanceFields which reads [[Fields]] and calls DefineField for each field. DefineField: If the fieldName is a private name, calls PrivateFieldAdd.

Extends Runtime semantics for ClassDefinitionEvaluation to set the constructor's [[Fields]].

Extends GetValue: If the reference is a private reference, calls PrivateFieldGet.

Extends SetValue: If the reference is a private reference, calls PrivateFieldSet.

Private methods proposal (background info)

Adds the syntax for methods.

Extends Private Name

- For private methods, [[Value]] contains the private method.
- For accessors, [[Get]] and [[Set]] contain the access or functions.
- For private methods and accessors, [[Brand]] contains the "original class of the private method or accessor".

Extend PrivateFieldGet: if the field kind is "method" or "accessor", do PrivateBrandCheck and return the [[Value]] of the private name or [[Get]] of the Private Name, respectively.

Similarly for PrivateFieldSet.

DefineOrdinaryMethod sets the [[Value]] and [[Brand]] of the Private Name.

Private brand checks proposal (this feature)

Runtime semantics for "PrivateIdentifier in ShiftExpression":

- Fields -> do PrivateFieldFind
- Methods and accessors: Do PrivateBrandCheck (defined in the "Private methods" proposal)

PrivateFieldFind doesn't walk the prototype chain up. So we're only asking if that particular object has the field.

For private methods, it's only a brand check. This makes sense, since an object always has all the private methods of a class, or none. Private methods cannot be deleted, and an object cannot be partially initialized in a way that would affect methods.

The implementation

Parsing

The proposal plugs the syntax in ShiftExpression so that it gets the right priority among the BinaryExpression. It allows a specific kind of ShiftExpression, namely, “PrivateName in ShiftExpression”. Note that PrivateName is not a valid PrimaryExpression and not allowed as a component of any other BinaryExpression.

V8 doesn’t implement the binary expression parsing like the spec defines it, instead it parses all binary expressions in ParserBase::ParseBinaryExpression and sorts out the operator priority there. So we’ll need to semi-awkwardly plug the private brand check expressions there.

Bytecode generator

Private fields
-> Use the bytecode TestIn with the private name.

Private non-static methods
-> Use the bytecode TestIn, but instead of the private name, test for the class brand.

Private static methods
-> Implemented fully in the BytecodeGenerator: we just check if the right hand side is the class where the method is defined. Special casing is needed for the case where the right hand side is not an object (we need to throw an error).

Runtime / IC

Handling of the TestIn bytecode (functions which need changes are in bold):

TestIn
-> KeyedHasIC
 -> KeyedLoadIC w/ LoadAccessMode::kHas
  -> KeyedHasIC_Megamorphic
   -> **CodeStubAssembler::HasProperty**
     -> Prototype chain walk right here
     -> Runtime_HasProperty
  -> Runtime_KeyedHasIC_Miss
   -> KeyedLoadIC::Load
     -> **LoadIC::Load**
     -> KeyedLoadIC::RuntimeLoad -> Runtime_HasProperty

For property lookup, we already have the logic for “if it’s a private name, don’t walk up the prototype chain”.

- C++: LookupIterator::ComputeConfiguration. It’s used by Runtime_HasProperty via JSReceiver::HasProperty.
- CSA: AccessorAssembler::GenericPropertyLoad.

Changes:

- Add the “if it’s a private name, don’t walk up the prototype chain” logic in Try
- LoadIC::Load shouldn’t throw an error if we use private names in the IsAnyHas mode.