**Strawman for adding a [[MethodCall]] MOP operation and corresponding Proxy support.**

Deltas to the Rev 14 ES6 draft.

## 8.3 Ordinary Object Internal Methods and Internal Data Properties

### 8.3.14+ [[MethodCall]] (P, ArgumentsList, Receiver)

When the [[MethodCall]] internal method of *O* is called with property key *P*, List *ArgumentsList*, and ECMAScipt language value *Receiver* the following steps are taken:

1. Assert: IsPropertyKey(*P*) is **true**.
2. Assert: *argumentsList* is a List.
3. Let *method* be the result of calling the [[Get]] internal method of *O* with arguments *P*, and *Receiver*.
4. ReturnIfAbrupt(*method*).
5. If Type(*method*) is not Object, throw a **TypeError** exception.
6. If IsCallable(*method*) is **false**, throw a **TypeError** exception.
7. Return the result of calling the [[Call]] internal method of *method* with *Receiver* as the *thisArgument* and *ArgumentsList* as *argumentsList*.

**Comment [AWB151]:** [[Method]] call is a new derive trap that is emitted for all FunctionCall expressions wh the function value is derived from a property access.

### 8.4.4 Symbol Exotic Objects

### 8.4.4.9 [[Get]] (P, Receiver)

When the [[Get]] internal method of an exotic Symbol object *O* is called with property key *P* and ECMAScipt language value *Receiver* the following steps are taken:

1. Assert: IsPropertyKey(*P*) is **true**.
2. Return **undefined**.

**Comment [AWB152]:** In the current spec. [[Get]] for Symbol objects exposes the current realms default toString method.

### 8.4.14 [[MethodCall]] (P, ArgumentsList, Receiver)

When the [[MethodCall]] internal method ofan exotic Symbol object *O* is called with property key *P*, List *ArgumentsList*, and ECMAScipt language value *Receiver* the following steps are taken:

3. Assert: IsPropertyKey(*P*) is **true**.
4. If *P* is **"toString"**, then
    a. Let *ctx* be the running execution context.
    b. Let *ctxRealm* be *ctx*'s Realm component.
    c. Let *toString* be *ctxRealm*.[[intrinsics]].% ObjProto_toString %.
    d. Return the result of calling the [[Call]] internal method of *toString* with *Receiver* as the *thisArgument* and *ArgumentsList* as *argumentsList*.
5. Throw a **TypeError** exception.

**Comment [AWB153]:** [[MethodCall]] disallows all method calls on symbols except "toString" and support without exposing a function object.

## 8.5 Proxy Object Internal Methods and Internal Data Properties

### 8.5.9 [[Get]] (P, Receiver)

When the [[Get]] internal method of an exotic Proxy object *O* is called with property key *P* and ECMAScipt language value *Receiver* the following steps are taken:

1. Assert: IsPropertyKey(*P*) is **true**..

**Comment [AWB154]:** In this design, all MOP operations that don't have traps are transparently forwarded to the target object with and this referecnes the proxy translated to this-references to the target.

2. Let *handler* be the value of the [[ProxyHandler]] internal data property of *O*.
3. Let *target* be the value of the [[ProxyTarget]] internal data property of *O*.
4. Let *trap* be the result of GetMethod(*handler*, "**get**").
5. ReturnIfAbrupt(*trap*).
6. If *trap* is **undefined**, then
    a. If SameValue(*O*, *Receiver*) is **true**, then
        i. Let *forwardedReceiver* be *target*.
    b. Else,
        i. Let *forwardedReceiver* be *Receiver*.
    c. Return the result of calling the [[Get]] internal method of *target* with arguments *P* and *forwardedReceiver*.
7. Let *trapResult* be the result of calling the [[Call]] internal method of *trap* with *handler* as the **this** value and a new List containing *target*, *P*, and *Receiver*.
8. ReturnIfAbrupt(*trapResult*).
9. Let *targetDesc* be the result of calling the [[GetOwnProperty]] internal method of *target* with argument *P*.
10. ReturnIfAbrupt(*targetDesc*).
11. If *targetDesc* is not **undefined**, then
    a. If IsDataDescriptor(*targetDesc*) and *targetDesc*.[[Configurable]] is **false** and *targetDesc*.[[Writable]] is **false**, then
        i. If SameValue(*trapResult*, *targetDesc*.[[Value]]) is **false**, then throw a **TypeError** exception.
    b. If IsAccessorDescriptor(*targetDesc*) and *targetDesc*.[[Configurable]] is **false** and *targetDesc*.[[Get]] is **undefined**, then
        i. If *trapResult* is not **undefined**, then throw a **TypeError** exception.
12. Return *trapResult*.

NOTE [[Get] for proxy objects enforces the following invariants:

- The value reported for a property must be the same as the value of the corresponding target object property if the target object property is a non-writable, non-configurable data property.
- The value reported for a property must be **undefined** if the corresponding corresponding target object property is non-configurable accessor property that has **undefined** as its [[Get]] attribute.

**Comment [AWB155]:** If the receiver is the same as the proxy, the target is used as the receiver.

Get accessors run with the target as their this value.

### 8.5.10 [[Set]] ( P, V, Receiver)

When the [[Set]] internal method of an exotic Proxy object *O* is called with property key *P*, value *V*, and ECMAScipt language value *Receiver*, the following steps are taken:

1. Assert: IsPropertyKey(*P*) is **true**.
2. Let *handler* be the value of the [[ProxyHandler]] internal data property of *O*.
3. Let *target* be the value of the [[ProxyTarget]] internal data property of *O*.
4. Let *trap* be the result of GetMethod(*handler*, "**set**").
5. ReturnIfAbrupt(*trap*).
6. If *trap* is **undefined**, then
    a. If SameValue(*O*, *Receiver*) is **true**, then
        i. Let *forwardedReceiver* be *target*.
    b. Else,
        i. Let *forwardedReceiver* be *Receiver*.
    c. Return the result of calling the [[Set]] internal method of *target* with arguments *P*, *V*, and *forwardedReceiver*.
7. Let *trapResult* be the result of calling the [[Call]] internal method of *trap* with *handler* as the **this** value and a new List containing *target*, *P*, *V*, and *Receiver*.
8. ReturnIfAbrupt(*trapResult*).
9. If ToBoolean(*trapResult*) is **false**, then return **false**.
10. Let *targetDesc* be the result of calling the [[GetOwnProperty]] internal method of *target* with argument *P*.
11. ReturnIfAbrupt(*targetDesc*).
12. If *targetDesc* is not **undefined**, then
    a. If IsDataDescriptor(*targetDesc*) and *targetDesc*.[[Configurable]] is **false** and *targetDesc*.[[Writable]] is **false**, then
        i. If SameValue(*V*, *targetDesc*.[[Value]]) is **false**, then throw a **TypeError** exception.
    b. If IsAccessorDescriptor(*targetDesc*) and *targetDesc*.[[Configurable]] is **false**, then
        i. If *targetDesc*.[[Set]] is **undefined**, then throw a **TypeError** exception.

**Comment [AWB156]:** If the receiver is the same as the proxy, the target is used as the receiver.

Set accessors run with the target as their this value.

13. Return **true**.

NOTE [[Set]] for proxy objects enforces the following invariants:

- Cannnot change the value of a property to be different from the value of the corresponding target object property if the corresponding target object property is a non-writable, non-configurable data property.
- Cannot set the value of a property if the corresponding corresponding target object property is a non-configurable accessor property that has **undefined** as its [[Set]] attribute.

### 8.5.13+ [[MethodCall]] (P, ArgumentsList, Receiver)

When the [[MethodCall]] internal method ofan exotic Proxy object *O* is called with property key *P*, List *ArgumentsList*, and ECMAScipt language value *Receiver* the following steps are taken:

13. Assert: IsPropertyKey(*P*) is **true**..
14. Let *handler* be the value of the [[ProxyHandler]] internal data property of *O*.
15. Let *target* be the value of the [[ProxyTarget]] internal data property of *O*.
16. Let *trap* be the result of GetMethod(*handler*, "`methodCall`").
17. ReturnIfAbrupt(*trap*).
18. If *trap* is **undefined**, then
    a. If SameValue(*O*, *Receiver*) is **true**, then
        i. Let *forwardedReceiver* be *target*.
    b. Else,
        i. Let *forwardedReceiver* be *Receiver*.
    c. Return the result of calling the [[MethodCall]] internal method of *target* with arguments *P*, *ArgumentsList*, and *forwardedReceiver*.
19. Let *argArray* be the result of CreateArrayFromList(*ArgumentsList*).
20. Return the result of calling the [[Call]] internal method of *trap* with *handler* as the **this** value and a new List containing *target*, *P*, *argArray*, and *Receiver*.

**Comment [AWB157]:** The name of the trap

**Comment [AWB158]:** If the receiver is the same as t proxy, the target is used as the receiver.

Methods run with the target as their this value, even if t method is inherited from a taget parent.

**Comment [AWB159]:** There are no invariants enforc upon methodCall.

### 11.2.3 Function Calls

#### Runtime Semantics: Evaluation

**Comment [AWB1510]:** No changes here, they are ju for reference.

*CallExpression* : *MemberExpression Arguments*

1. Let *ref* be the result of evaluating *MemberExpression*.
2. If this *CallExpression* is in a tail position (13.7) then let *tailCall* be **true**, otherwise let *tailCall* be **false**.
3. Return the result of the abstract operation EvaluateCall with arguments *ref*, *Arguments*, and *tailCall*.

*CallExpression* : *CallExpression Arguments*

1. Let *ref* be the result of evaluating *CallExpression*.
2. If this *CallExpression* is in a tail position (13.7) then let *tailCall* be **true**, otherwise let *tailCall* be **false**.
3. Return the result of the abstract operation EvaluateCall with arguments *ref*, *Arguments*, and *tailCall*.

#### Runtime Semantics: EvaluateCall Abstract Operation

The abstract operation EvaluateCall takes as arguments a value *ref*, and a syntactic grammar production *arguments*, and a Boolean argument *tailPosition*. It performs the following steps:

1. ReturnIfAbrupt(*ref*).
2. If Type(*ref*) is Reference, then

a. If IsPropertyReference(*ref*) is **true**, then
   i. Return the result of the abstract operation EvaluateMethodCall with arguments *ref*, *arguments*, and *tailPosition*.
b. Else, the base of *ref* is an Environment Record
   i. Let *thisValue* be the result of calling the WithBaseObject concrete method of GetBase(*ref*).
   ii. If *thisValue* is not **undefined**, then
      1. Let *newRef* be a value of type Reference whose base value is *thisValue* and whose referenced name is GetReferencedName(*ref*), and whose strict reference flag is IsStrictReference(ref).
      2. Return the result of the abstract operation EvaluateMethodCall with arguments *newRef*, *arguments*, and *tailPosition*.

3. Else Type(*ref*) is not Reference,
   a. Let *thisValue* be **undefined**.
4. Let *func* be GetValue(*ref*).
5. ReturnIfAbrupt(*func*).
6. Let *argList* be the result of performing ArgumentListEvaluation of *arguments*.
7. ReturnIfAbrupt(*argList*).
8. If Type(*func*) is not Object, throw a **TypeError** exception.
9. If IsCallable(*func*) is **false**, throw a **TypeError** exception.
10. If *tailPosition* is **true**, then
   a. Let *leafContext* be the running execution context.
   b. Suspend *leafContext*.
   c. Pop *leafContext* from the execution context context stack. The execution context now on the top of the stack becomes the running execution context, however it remains in its suspended state.
   d. Assert: *leafContext* has no further use. It will never be activated as the running execution context.
11. Let *result* be the result of calling the [[Call]] internal method on *func*, passing *thisValue* as the *thisArgument* and *argList* as the *argumentsList*.
12. Assert: If *tailPosition* is **true**, the above call will not return here, but instead evaluation will continue with the resumption of *leafCallerContext* as the running execution context.
13. Assert: Type(*result*) is an ECMAScript language type
14. Return *result*.

A tail position call must either release any transient internal resources associated with the currently executing function execution context before invoking the target function or reuse those resources in support of the target function.

NOTE 1 For example, a tail position call should only grow an implementation's activication record stack by the amount that the size of the target function's activation record exceeds the size of the calling function's activation record. If the target function's activation record is smaller, then the total size of the stack should decrease.

### Runtime Semantics: EvaluateMethodCall Abstract Operation

The abstract operation EvaluatePropertyCall takes as arguments a value *ref*, and a syntactic grammar production *arguments*, and a Boolean argument *tailPosition*. It performs the following steps:

1. Assert: Type(*ref*) is Reference and IsPropertyReference(*ref*) is **true**
2. If IsUnresolvableReference(*V*), throw a **ReferenceError** exception.
3. Let *argList* be the result of performing ArgumentListEvaluation of *arguments*.
4. ReturnIfAbrupt(*argList*).
5. Let *base* be the result of calling GetBase(*ref*).
6. If HasPrimitiveBase(*ref*) is **true**, then
   a. Assert: In this case, *base* will never be **null** or **undefined**.
   b. Let *base* be ToObject(*base*).
7. Let *thisValue* be GetThisValue(*ref*).
8. Let *key* be GetReferencedKey(*ref*).
9. If *tailPosition* is **true**, then
   a. Let *leafContext* be the running execution context.
   b. Suspend *leafContext*.
   c. Pop *leafContext* from the execution context context stack. The execution context now on the top of the stack becomes the running execution context, however it remains in its suspended state.

**Comment [AWB1511]:** Explict property references are handled as method calls.

**Comment [AWB1512]:** Implicit property references (via a with binding or global object binding) are also handled as method calls.

**Comment [AWB1513]:** Retriving the functi value is now only happens here if the ref is neither an implicit or explicit property referenc

**Comment [AWB1514]:** This is a new abstr operation that handles method calls.

**Comment [AWB1515]:** This takes cares of property access on primitive values.

d. Assert: *leafContext* has no further use. It will never be activated as the running execution context.

10. Let *result* be the result of calling the [[MetholdCall]] internal method on *base*, passing *key*, as the *argList* *argumentsList* and *thisValue*.
11. Assert: If *tailPosition* is **true**, the above call will not return here, but instead evaluation will continue with the resumption of *leafCallerContext* as the running execution context.
12. Assert: Type(*result*) is an ECMAScript language type
13. Return *result*.

Comment [AWB1516]: It translate them into calls on the [[MethodCall]] internal method of the the base objec

#### 11.2.4 The `super` Keyword

#### Runtime Semantics: Evaluation

*CallExpression* : `super` *Arguments*

Comment [AWB1517]: This is another form of functi call syntax that alwas is a method call

1. If the code matched by the syntactic production that is being evaluated is strict mode code, let *strict* be **true**, else let *strict* be **false**.
2. Let *ref* be the result of MakeSuperReference(**undefined**, *strict*).
3. ReturnIfAbrupt(*ref*).
4. If this *CallExpression* is in a tail position (13.7) then let *tailCall* be **true**, otherwise let *tailCall* be **false**.
5. Return the result of the abstract operation EvaluateMethodCall with arguments *ref*, *Arguments*, and *tailCall*.

### 15.17.1 Exported Function Properties Reflecting the Essentional Internal Methods

#### 15.17.1.8 Reflect.methodCall (target, propertyKey, argumentsList, receiver=target)

When the `get` function is called with arguments *target*, *propertyKey*, *argumentsList*, and *receiver* the following steps are taken:

1. Let *obj* be ToObject(*target*).
2. ReturnIfAbrupt(*obj*).
3. Let *key* be ToPropertyKey(*propertyKey*).
4. ReturnIfAbrupt(*key*).
5. If *receiver* is not present, then
    a. Let *receiver* be *target*.
6. Let *argList* be an empty List.
7. Let *index* be 0.
8. Repeat while *index* < *n*
    a. Let *indexName* be ToString(*index*).
    b. Let *nextArg* be the result of Get(*argumentsList*, *indexName*).
    c. ReturnIfAbrupt(*nextArg*).
    d. Append *nextArg* as the last element of *argList*.
    e. Set *index* to *index* + 1.
9. Return the result of calling the [[Get]] internal method of *obj* with arguments *key*, *argList*, and *receiver*.