ɔbservable

1 The Observable Constructor

The Observable constructor is the %Observable% intrinsic object and the initial value of the Observable property of the global object. When called as a constructor it creates and initializes a new Observable object. Observable is not intended to be called as a function and will throw an exception when called in that manner.

The Observable constructor is designed to be subclassable. It may be used as the value in an extends clause of a class definition. Subclass constructors that intend to inherit the specified Observable behaviour must include a super call to the Observable constructor to create and initialize the subclass instance with the internal state necessary to support the Observable and Observable.prototype built-in methods.

1.1 Observable (*subscriber*)

The **`Observable`** constructor initializes a new Observable object. It is not intended to be called as a function and will throw an exception when called in that manner.

The *subscriber* argument must be a function object. It is called each time the **`subscribe`** method of the Observable object is invoked. The *subscriber* function is called with a wrapped observer object and may optionally return a function which will cancel the subscription.

The **`Observable`** constructor performs the following steps:

1. If NewTarget is **undefined**, throw a **TypeError** exception.
2. If IsCallable(*subscriber*) is **false**, throw a **TypeError** exception.
3. Let *observable* be ? OrdinaryCreateFromConstructor(NewTarget, %ObservablePrototype%, « [[Subscriber]] »).
4. Set *observable's* [[Subscriber]] internal slot to *subscriber*.
5. Return *observable*.

2 Properties of the Observable Constructor

lue of the [[Prototype]] internal slot of the **`Observable`** constructor is the intrinsic object
tionPrototype%.

Besides the **`length`** property (whose value is 1), the **`Observable`** constructor has the following properties:

2.1 Observable.prototype

The initial value of **`Observable.prototype`** is the intrinsic object %ObservablePrototype%.

This property has the attributes { [[Writable]]: **false**, [[Enumerable]]: **false**, [[Configurable]]: **false** }.

2.2 Observable.from (*x*)

When the **`from`** method is called, the following steps are taken:

```
1. Let C be the this value.
2. If IsConstructor(C) is false, let C be %Observable%.
3. Let observableMethod be ? GetMethod(x, @@observable).
4. If observableMethod is not undefined, then
    a. Let observable be ? Call(observableMethod, x, « »).
    b. If Type(observable) is not Object, throw a TypeError exception.
    c. Let constructor be ? Get(observable, "constructor").
    d. If SameValue(constructor, C) is true, return observable.
    e. Let subscriber be a new built-in function object as defined in Observable.from Delegating Functions.
    f. Set subscriber's [[Observable]] internal slot to observable.
5. Else,
    a. Let iteratorMethod be ? GetMethod(x, @@observable).
    b. If iteratorMethod is undefined, throw a TypeError exception.
    c. Let subscriber be a new built-in function object as defined in Observable.from Iteration Functions.
    d. Set subscriber's [[Iterable]] internal slot to x.
    e. Set subscriber's [[IteratorMethod]] internal slot to iteratorMethod.
6. Return Construct(C, « subscriber »).
```

2.2.1 Observable.from Delegating Functions

servable.from delegating function is an anonymous built-in function that has an [[Observable]] internal

When an Observable.from delegating function is called with argument *observer*, the following steps are taken:

```
1. Let observable be the value of the [[Observable]] internal slot of F.
2. Return Invoke(observable, "subscribe", « observer »).
```

The `length` property of an Observable.from delegating function is `1`.

2.2.2 Observable.from Iteration Functions

An Observable.from iteration function is an anonymous built-in function that has [[Iterable]] and [[IteratorFunction]] internal slots.

When an Observable.from iteration function is called with argument *observer*, the following steps are taken:

```
1. Let iterable be the value of the [[Iterable]] internal slot of F.
2. Let iteratorMethod be the value of the [[IteratorMethod]] internal slot of F.
3. Let iterator be ? GetIterator(items, iteratorMethod).
4. Let subscription be the value of observer's [[Subscription]] internal slot.
5. Repeat
    a. Let next be ? IteratorStep(iterator).
    b. If next is false, then
        i. Perform ! Invoke(observer, "complete", « »).
        ii. Return undefined.
    c. Let nextValue be ? IteratorValue(next).
    d. Perform ! Invoke(observer, "next", « nextValue »).
    e. If SubscriptionClosed(subscription) is true, then
        i. Return ? IteratorClose(iterator, undefined).
```

The `length` property of an Observable.from iteration function is `1`.

2.3 Observable.of (...*items*)

```
1. Let C be the this value.
2. If IsConstructor(C) is false, let C be %Observable%.
3. Let subscriber be a new built-in function object as defined in Observable.of Subscriber Functions.
4. Set subscriber's [[Items]] internal slot to items.
```

```
   Return Construct(C, « subscriber »).
```

2.3.1 Observable.of Subscriber Functions

An Observable.of subscriber function is an anonymous built-in function that has an [[Items]] internal slot.

When an Observable.of subscriber function is called with argument *observer*, the following steps are taken:

```
1. Let items be the value of the [[Items]] internal slot of F.
2. Let subscription be the value of observer's [[Subscription]] internal slot.
3. For each element value of items
    a. Perform ! Invoke(observer, "next", « value »).
    b. If SubscriptionClosed(subscription) is true, then
        i. Return undefined.
4. Perform ! Invoke(observer, "complete", « »).
5. Return undefined.
```

The `length` property of an Observable.of subscriber function is `1`.

3 Properties of the Observable Prototype Object

The `Observable` prototype object is the intrinsic object %ObservablePrototype%. The value of the [[Prototype]] internal slot of the `Observable` prototype object is the intrinsic object %ObjectPrototype%. The `Observable` prototype object is an ordinary object.

3.1 Observable.prototype.subscribe (*observer*)

The `subscribe` function begins sending values to the supplied *observer* object by executing the Observable object's subscriber function. It returns a `Subscription` object which may be used to cancel the subscription.

The `subscribe` function performs the following steps:

```
1. Let O be the this value.
2. If Type(O) is not Object, throw a TypeError exception.
3. If O does not have an [[Subscriber]] internal slot, throw a TypeError exception.
4. If IsCallable(observer) is true, then
```

```
        a. Let len be the actual number of arguments passed to this function.
        b. Let args be the List of arguments passed to this function.
        c. Let nextCallback be observer.
        d. If len > 1, let errorCallback be args[1].
        e. Else, let errorCallback be undefined.
        f. If len > 2, let completeCallback be args[2].
        g. Else, let completeCallback be undefined.
        h. Let observer be ObjectCreate(%ObjectPrototype%).
        i. Perform ! CreateDataProperty(observer, "next", nextCallback).
        j. Perform ! CreateDataProperty(observer, "error", errorCallback).
        k. Perform ! CreateDataProperty(observer, "complete", completeCallback).
 5. Else if Type(observer) is not Object, Let observer be ObjectCreate(%ObjectPrototype%).
 6. Let subscription be ? CreateSubscription(observer).
 7. Let startMethodResult be GetMethod(observer, "start").
 8. If startMethodResult.[[Type]] is normal, then
        a. Let start be startMethodResult.[[Value]].
        b. If start is not undefined, then
            i. Let result be Call(start, observer, « subscription »).
            ii. If result is an abrupt completion, perform HostReportErrors(« result.[[Value]] »).
            iii. If SubscriptionClosed(subscription) is true, then
                1. Return subscription.
 9. Else if startMethodResult.[[Type]] is throw, then perform HostReportErrors(« startMethodResult.
    [[Value]] »).
10. If result is an abrupt completion, perform HostReportErrors(« result.[[Value]] »).
11. Let subscriptionObserver be ? CreateSubscriptionObserver(subscription).
12. Let subscriber be the value of O's [[Subscriber]] internal slot.
13. Assert: IsCallable(subscriber) is true.
14. Let subscriberResult be ExecuteSubscriber(subscriber, subscriptionObserver).
15. If subscriberResult is an abrupt completion, then
        a. Perform ! Invoke(subscriptionObserver, "error", « subscriberResult.[[value]] »).
16. Else,
        a. Set the [[Cleanup]] internal slot of observer to subscriberResult.[[value]].
17. If SubscriptionClosed(subscription) is true, then
        a. Perform ! CleanupSubscription(subscription).
18. Return subscription.
```

3.1.1 ExecuteSubscriber (*subscriber*, *observer*)

stract operation ExecuteSubscriber with arguments *subscriber* and *observer* performs the following

1. Assert: IsCallable(*subscriber*) is **true**.
2. Assert: Type(*observer*) is Object.
3. Let *subscriberResult* be ? Call(*subscriber*, **undefined**, *observer*).
4. If *subscriberResult* is **null** or **undefined**, return **undefined**.
5. If IsCallable(*subscriberResult*) is **true**, return *subscriberResult*.
6. Let *result* be ? GetMethod(*subscriberResult*, `"unsubscribe"`).
7. If *result* is **undefined**, throw a **TypeError** exception.
8. Let *cleanupFunction* be a new built-in function object as defined in Subscription Cleanup Functions.
9. Set *cleanupFunction*'s [[Subscription]] internal slot to *subscriberResult*.
10. Return *cleanupFunction*.

3.1.2 Subscription Cleanup Functions

A subscription cleanup function is an anonymous built-in function that has a [[Subscription]] internal slot.

When a subscription cleanup function *F* is called the following steps are taken:

1. Assert: *F* as a [[Subscription]] internal slot whose value is an Object.
2. Let *subscription* be the value of *F*'s [[Subscription]] internal slot.
3. Return Invoke(*subscription*, `"unsubscribe"`, « »).

The `length` property of a subscription cleanup function is `0`.

3.2 Observable.prototype.constructor

The initial value of `Observable.prototype.constructor` is the intrinsic object %Observable%.

3.3 Observable.prototype [@@observable] ( )

The following steps are taken:

1. Return the **this** value.

The value of the `name` property of this function is `"[Symbol.observable]"`.

ubscription Objects

A Subscription is an object which represents a channel through which an Observable may send data to an Observer.

4.1 Subscription Abstract Operations

4.1.1 CreateSubscription (*observer*)

The abstract operation CreateSubscription with argument *observer* is used to create a Subscription object. It performs the following steps:

1. Assert: Type(*observer*) is Object.
2. Let *subscription* be ObjectCreate(%SubscriptionPrototype%, « [[Observer]], [[Cleanup]] »).
3. Set *subscription*'s [[Observer]] internal slot to *observer*.
4. Set *subscription*'s [[Cleanup]] internal slot to **undefined**.
5. Return *subscription*.

4.1.2 CleanupSubscription (*subscription*)

The abstract operation CleanupSubscription with argument *subscription* performs the following steps:

1. Assert: *subscription* is a Subscription object.
2. Let *cleanup* be the value of *subscription*'s [[Cleanup]] internal slot.
3. If *cleanup* is **undefined**, return **undefined**.
4. Assert: IsCallable(*cleanup*) is **true**.
5. Set *subscription*'s [[Cleanup]] internal slot to **undefined**.
6. Let *result* be Call(*cleanup*, **undefined**, « »).
7. If *result* is an abrupt completion, perform HostReportErrors(« *result*.[[Value]] »).
8. Return **undefined**.

4.1.3 SubscriptionClosed (*subscription*)

The abstract operation SubscriptionClosed with argument *subscription* performs the following steps:

1. Assert: *subscription* is a Subscription object.
2. If the value of *subscription*'s [[Observer]] internal slot is **undefined**, return **true**.

lse, return **false**.

4.2 The %SubscriptionPrototype% Object

All Subscription objects inherit properties from the %SubscriptionPrototype% intrinsic object. The %SubscriptionPrototype% object is an ordinary object and its [[Prototype]] internal slot is the %ObjectPrototype% intrinsic object. In addition, %SubscriptionPrototype% has the following properties:

4.2.1 get %SubscriptionPrototype%.closed

1. Let *subscription* be the **this** value.
2. If Type(*subscription*) is not Object, throw a **TypeError** exception.
3. If *subscription* does not have all of the internal slots of a Subscription instance, throw a **TypeError** exception.
4. Return ! SubscriptionClosed(*subscription*).

4.2.2 %SubscriptionPrototype%.unsubscribe ( )

1. Let *subscription* be the **this** value.
2. If Type(*subscription*) is not Object, throw a **TypeError** exception.
3. If *subscription* does not have all of the internal slots of a Subscription instance, throw a **TypeError** exception.
4. If SubscriptionClosed(*subscription*) is **true**, return **undefined**.
5. Set *subscription*'s [[Observer]] internal slot to **undefined**.
6. Return CleanupSubscription(*subscription*).

5 Subscription Observer Objects

A Subscription Observer is an object which wraps the *observer* argument supplied to the **`subscribe`** method of Observable objects. Subscription Observer objects are passed as the single parameter to an observable's *subscriber* function. They enforce the following guarantees:

- If the observer's **`error`** method is called, the observer will not be invoked again and the observable's cleanup function will be called.

f the observer's **`complete`** method is called, the observer will not be invoked again and the observable's leanup function will be called.
- When the subscription is canceled, the observer will not be invoked again.

In addition, Subscription Observer objects provide default behaviors when the observer does not implement **`next`**, **`error`** or **`complete`**.

5.1 Subscription Observer Abstract Operations

5.1.1 CreateSubscriptionObserver (*subscription*)

The abstract operation CreateSubscriptionObserver with argument *observer* is used to create a normalized observer which can be supplied to an observable's subscriber function. It performs the following steps:

1. Assert: Type(*subscription*) is Object.
2. Let *subscriptionObserver* be ObjectCreate(%SubscriptionObserverPrototype%, « [[Subscription]] »).
3. Set *subscriptionObserver*'s [[Subscription]] internal slot to *subscription*.
4. Return *subscriptionObserver*.

5.2 The %SubscriptionObserverPrototype% Object

All Subscription Observer objects inherit properties from the %SubscriptionObserverPrototype% intrinsic object. The %SubscriptionObserverPrototype% object is an ordinary object and its [[Prototype]] internal slot is the %ObjectPrototype% intrinsic object. In addition, %SubscriptionObserverPrototype% has the following properties:

5.2.1 get %SubscriptionObserverPrototype%.closed

1. Let *O* be the **this** value.
2. If Type(*O*) is not Object, throw a **TypeError** exception.
3. If *O* does not have all of the internal slots of a Subscription Observer instance, throw a **TypeError** exception.
4. Let *subscription* be the value of *O*'s [[Subscription]] internal.
5. Return ! SubscriptionClosed(*subscription*).

5.2.2 %SubscriptionObserverPrototype%.next (*value*)

```
   et O be the this value.
   Type(O) is not Object, throw a TypeError exception.
3. If O does not have all of the internal slots of a Subscription Observer instance, throw a TypeError
   exception.
4. Let subscription be the value of O's [[Subscription]] internal slot.
5. If SubscriptionClosed(subscription) is true, return undefined.
6. Let observer be the value of subscription's [[Observer]] internal slot.
7. Assert: Type(observer) is Object.
8. Let nextMethodResult be GetMethod(observer, "next").
9. If nextMethodResult.[[Type]] is normal, then
    a. Let nextMethod be nextMethodResult.[[Value]].
    b. If nextMethod is not undefined, then
        i. Let result be Call(nextMethod, observer, « value »).
        ii. If result is an abrupt completion, perform HostReportErrors(« result.[[Value]] »).
10. Else if nextMethodResult.[[Type]] is throw, then perform HostReportErrors(« nextMethodResult.[[Value]]
   »).
11. Return undefined.
```

5.2.3 %SubscriptionObserverPrototype%.error (*exception*)

```
1. Let O be the this value.
2. If Type(O) is not Object, throw a TypeError exception.
3. If O does not have all of the internal slots of a Subscription Observer instance, throw a TypeError
   exception.
4. Let subscription be the value of O's [[Subscription]] internal slot.
5. If SubscriptionClosed(subscription) is true, return undefined.
6. Let observer be the value of subscription's [[Observer]] internal slot.
7. Assert: Type(observer) is Object.
8. Let errorMethodResult be GetMethod(observer, "error").
9. If errorMethodResult.[[Type]] is normal, then
    a. Let errorMethod be errorMethodResult.[[Value]].
    b. If errorMethod is not undefined, then
        i. Let result be Call(errorMethod, observer, « exception »).
        ii. If result is an abrupt completion, perform HostReportErrors(« result.[[Value]] »).
10. Else if errorMethodResult.[[Type]] is throw, then perform HostReportErrors(« errorMethodResult.
   [[Value]] »).
11. Perform ! CleanupSubscription(subscription).
12. Return undefined.
```

%SubscriptionObserverPrototype%.complete ( )

```
1. Let O be the this value.
2. If Type(O) is not Object, throw a TypeError exception.
3. If O does not have all of the internal slots of a Subscription Observer instance, throw a TypeError exception.
4. Let subscription be the value of O's [[Subscription]] internal slot.
5. If SubscriptionClosed(subscription) is true, return undefined.
6. Let observer be the value of subscription's [[Observer]] internal slot.
7. Assert: Type(observer) is Object.
8. Let completeMethodResult be GetMethod(observer, "complete").
9. If completeMethodResult.[[Type]] is normal, then
    a. Let completeMethod be completeMethodResult.[[Value]].
    b. If completeMethod is not undefined, then
        i. Let result be Call(completeMethod, observer).
        ii. If result is an abrupt completion, perform HostReportErrors(« result.[[Value]] »).
10. Else if completeMethodResult.[[Type]] is throw, then perform HostReportErrors(« completeMethodResult.[[Value]] »).
11. Perform ! CleanupSubscription(subscription).
12. Return undefined.
```