The purpose of this doc is to brainstorm and create a strawman proposal for parameter descriptors that could be used to attach metadata to function parameters that could be used by function parameter decorators.

goals:

* define a parameter descriptor
  + consider properties like: name, position, default, optional, rest, type
* define an ability to retrieve parameter descriptors for a function
* define an ability to read and write metadata from/to a descriptor

unclear/non-goals?

* define an ability to create new or remove existing parameters for a function
* define an ability to change non-metadata properties
* define descriptors for function arguments ( someFn(@someDecorator arg1) )

# Parameter Descriptor Type

All parameter descriptor objects have the following fields:

* *name*: string, readonly name of the parameter
* *position*: number, readonly 0-based index of the parameter
* *default*: any, readonly default value of the parameter
* *optional*: boolean, readonly flag for marking optionality of a parameter. The value is true only if default value was defined.
* *rest*: boolean, readonly flag for rest parameters

# Retrieving Parameter Descriptors

There are several options to consider, including top level api and key used for lookup.

Retrieval api:

* adding Function.getParameterDescriptor
* reusing Object.getOwnPropertyDescriptor (confusing name and target)
* the Reflect namespace: Reflect.getParameters(func)

Key used for lookup:

* 0-based index
* parameter name (would require adding something like Function.getParameterNames, which would be handy anyway);
* no key-based lookup; just an API that provides an array of descriptors (which have an inherent position and explicit name information)

# Retrieving and Modifying Metadata Associated with Parameters

Could Reflect.metadata be expanded to accommodate parameter metadata? Should it be key-ed off of the parameter descriptor?

Would keying off of descriptor couldn't be pooled and reused for function with the same signature?

Or should metadata be part of the descriptor?

# Using Param Descriptors with Decorators

function someFunction(@dec1 foo, @dec2() bar, @dec3('someVal') baz) {

...

}

function dec1(targetFn, targetName, targetParamDescriptor) {

// define metadata for the

Reflect.defineMetadata('someMetaKey','someMetaValue', targetfn, targetParamDescriptor);

}

// reading metadata - index based

let paramDescriptor = Function.getParameterDescriptor(0);

let metadata = Reflect.getMetadata(paramDescriptor, someFunction);

// reading metadata - name based

let firstParamName = Function.getParameterNames(somefunction)[0];

let paramDescriptor = Function.getParameterDescriptor(firstParamName);

let metadata = Reflect.getMetadata(paramDescriptor, someFunction);

// deleting metadata

let paramDescriptor = ...;

let result = Reflect.deleteMetadata('someMetaKey',someFunction, paramDescriptor);

# Reflect.getParameters Proposal

function es3(a, a, b) {}

Reflect.getParameters(es3)

//=> [

// { type: 'simple', name: 'a' },

// { type: 'simple', name: 'a' },

// { type: 'simple', name: 'b' }

// ]

function es6(a, b, c=1, ...rest) {}

Reflect.getParameters(es6)

//=> [

// { type: 'simple', name: 'a' },

// { type: 'simple', name: 'b' },

// { type: 'default', name: 'c' },

// { type: 'rest', name: 'rest' }

// ]

# 

# What about inheritance?

Do we need to care? Functions can't inherit unless there is some prototype sorcery going on.