Feature of lib: duck-type

Prepare in nodejs:

```
var duck = require('../duck-type').namespace();
```

Part One:

Let us get start with examples:

-- Example 1

Support we have to implement a function foo(param1) {...}, and we want to make sure that param1 should be a String

You can verify the type of param1 like this:

```
function foo(param1) {
    duck(param1).is(String);
    ...
}
```

You also can verify many parameters at once, like:

```
duck(param1, param2).are(String, Number);
```

-- example 2

How about complex object like: {name:'hello', age: 12345}

You can verify it like:

```
duck(param1).is({name:String, age:Number});
```

Please note:

duc({name:'hello', age: 12345, something:'foo'}).is({name:String, age:Number}); //also can be passed, means the object is compatible with the type

-- example

what if more complicated object, like

```
{
    name: {
        first: 'Yu',
        last: 'Shen'
    },
    age: 123,
    sayHello: function() {
        //TODO
    }
}
```

you can verify the type like this:

```
duck(param1).is({
    name : {first:String, last:String},
```

```
        age: Number,
        sayHello: Function
    });
```

-- example 3

for array, duck-type can support different pattern:

```
    duck(param1).is([]); //means param1 must be a array
    duck(param1).is([Number]); //means param1 must be a array, and each element of the array must be a Number
    duck(param1).is([Number,String,Date]); //means param1 must be a array, and the first element must be a Number, the second element must be a String....
```

of cause, you can combine defination of array and defination of object, like;

```
duck(param1).is({
    title: String,
    description: String,
    resourceDemand: [{
        resourceTypeId: Number,
        year: Number,
        month: Number,
        quantity: Number
    }]
})
```

Part Two:

Yes, duck-type can be used to verify the type of your parameters of function, BUT, do not stop with verify. Declare the type of your system and re-use them are better choice.

-- Example 4

How to define type? Do it like:

```
duck.type('ResourceDemand',{   //now, we defined a type ResourceDemand
        resourceTypeId: Number,
        year: Number,
        month: Number,
        quantity: Number
});
```

How to use it?

```
duck(param1).is(duck.ResourceDemand);
```

-- Example 5

You can define some basic type, like Integer, Long

```
duck.type('Integer',function(value){
        return duck(value).is(Number) && value % 1 === 0 && value >= -2147483648 && value
<= 2147483647;
});
```

You also can define like:

```
duck.type('Email', function(value) {        //the callback function will be as a validator when test target, and the target will be pass to callback by parameter 'value'.
        ...
});

duck.type('IpAddress', function(value) {
        ...
})
```

-- Example 6

you can use type to defined your new type, I mean:

```
duck.type('Proposal',{
        id: duck.Integer
        title: String,
        description: String,
        resourceDemands: [duck.ResourceDemand]
});
```

-- Example 6

you can partially verify like this:

```
duck(param1).is(duck.Proposal.resourceDemands);
```

-- Example 7

Different type can be organize to different namespace:

```
duck.type('User',{                    // here, system, lang are different namespaces.
        id: system.Id,
        name: lang.Word
        age: lang.Integer,
});
```

Part Three:

type define first is encouraged, it is practice of 'Convention First', especially, when you have to teamwork with other people, or use some functions which are still developing. Type define can be a defense of your code.

If you have defined type already. 'mock' is another benefit of duck-type.

-- Example 8

duck.mock(duck.Proposal); //it will return an object, which must compatible with type Proposal.

I mean,

```
{
        id: 112,
        title: 'sdfasf adsf',
        description: 'sdfsdf sdf 234s sd',
        resourceDemands: [{
                resourceTypeId: 123,
                year: 2343,
                month: 234,
                quantity: 444
```

```
            }]
        }
```

the object like above might be return, of cause, most of value will be changed randomly.

End

Type define first. Then start your work with mock data, then verify parameter at product runtime.