星云链Nebulas——4.智能合约存储区

前面一篇《星云链Nebulas——3.编译和部署智能合约》介绍了如何编写智能合约以及如何在星云链部署和调用智能合约。

本篇我们来详细的介绍有关星云链智能合约的存储。星云链智能合约(Smart Contract)提供了链上数据存储功能，类似于传统的Key-Value存储系统（如NoSQL、Mongodb、Redis等），可以付费（即消耗Gas）将数据存储到星云链上。

内置存储对象

星云链的智能合约运行环境内置了存储对象`LocalContractStorage`，可以存储数字，字符串，JavaScript对象，存储数据只能在智能合约内使用，其他合约不能读取存储的内容。

1、基础用法

`LocalContractStorage`的简单接口包括`set`，`get`，`del`接口，实现了存储，读取，删除数据功能。存储可以是数字，字符串，对象。

- 存储数据

```
// 存储数据，数据会被json序列化成字符串保存
LocalContractStorage.put(key, value);
// 或者
LocalContractStorage.set(key, value);
```

- 读取数据

```
// 获取数据
LocalContractStorage.get(key);
```

- 删除数据

```
// 删除数据，数据删除后无法读取
LocalContractStorage.del(key);
// 或者
LocalContractStorage.delete(key);
```

- 合约实例

以下是一个具体在合约中使用LocalContractStorage的例子：

```
'use strict';

var SampleContract = function () {
};

SampleContract.prototype = {
    init: function () {
    },
    set: function (name, value) {
        // 存储字符串
        LocalContractStorage.set("name", name);
        // 存储数字
        LocalContractStorage.set("value", value);
        // 存储对象
        LocalContractStorage.set("obj", {name: name, value: value});
    },
    get: function () {
        var name = LocalContractStorage.get("name");
        console.log("name:" + name)
        var value = LocalContractStorage.get("value");
        console.log("value:" + value)
        var obj = LocalContractStorage.get("obj");
        console.log("obj:" + JSON.stringify(obj))
    },
    del: function () {
        var result = LocalContractStorage.del("name");
        console.log("del result:"+result)
    }
};

module.exports = SampleContract;
```

2、高级用法

`LocalContractStorage` 除了基本的 `set` , `get` , `del` 方法，还提供方法来绑定合约属性。对绑定过的合约属性的读写将直接在 `LocalContractStorage` 上读写，而无需调用 `get` 和 `set` 方法。

2.1 绑定属性

在绑定一个合约属性时，需要提供对象实例，属性名和序列化方法。

- 绑定接口

```
// define a object property named `fieldname` to `obj` with descriptor.
// default descriptor is JSON.parse/JSON.stringify descriptor.
// return this.
defineProperty(obj, fieldName, descriptor);

// define object properties to `obj` from `props`.
// default descriptor is JSON.parse/JSON.stringify descriptor.
// return this.
defineProperties(obj, descriptorMap);
```

- 绑定属性实例

  以下是一个在合约中使用`LocalContractStorage`绑定属性的例子：

```
'use strict';

var SampleContract = function () {
    // SampleContract的`size`属性为存储属性，对`size`的读写会存储到链上，
    // 此处的`descriptor`设置为null，将使用默认的JSON.stringify()和JSON.parse()
    LocalContractStorage.defineProperty(this, "size", null);

    // SampleContract的`value`属性为存储属性，对`value`的读写会存储到链上，
    // 此处的`descriptor`自定义实现，存储时直接转为字符串，读取时获得Bignumber对象
    LocalContractStorage.defineProperty(this, "value", {
        stringify: function (obj) {
            return obj.toString();
        },
        parse: function (str) {
            return new BigNumber(str);
        }
    });
    // SampleContract的多个属性批量设置为存储属性，对应的descriptor默认使用JSON序列化
    LocalContractStorage.defineProperties(this, {
        name: null,
        count: null
    });
};
module.exports = SampleContract;
```

- 读写绑定属性合约实例

  然后，我们可以如下在合约里直接读写这些属性。

```
SampleContract.prototype = {
    // 合约部署时调用，部署后无法二次调用
    init: function (name, count, size, value) {
        // 在部署合约时将数据存储到链上
        this.name = name;
        this.count = count;
        this.size = size;
        this.value = value;
    },
    testStorage: function (balance) {
        // 使用value时会从存储中读取链上数据，并根据descriptor设置自动转换为Bignumber
        var amount = this.value.plus(new BigNumber(2));
        if (amount.lessThan(new BigNumber(balance))) {
            return 0
        }
    }
};
```

2.2 绑定Map属性

`LocalContractStorage` 还提供了对合约中map属性的绑定方法。

- 绑定Map属性实例

  以下是绑定Map属性的例子：

```
'use strict';

var SampleContract = function () {
    // 为`SampleContract`定义`userMap`的属性集合，数据可以通过`userMap`存储到链上
    LocalContractStorage.defineMapProperty(this, "userMap");

    // 为`SampleContract`定义`userBalanceMap`的属性集合，并且存储和读取序列化方法自定义
    LocalContractStorage.defineMapProperty(this, "userBalanceMap", {
        stringify: function (obj) {
            return obj.toString();
        },
        parse: function (str) {
            return new BigNumber(str);
        }
    });

    // 为`SampleContract`定义多个集合
```

```
    LocalContractStorage.defineMapProperties(this,{
        key1Map: null,
        key2Map: null
    });
};

SampleContract.prototype = {
    init: function () {
    },
    testStorage: function () {
        // 将数据存储到userMap中，并序列化到链上
        this.userMap.set("robin","1");
        // 将数据存储到userBalanceMap中，使用自定义序列化函数，保存到链上
        this.userBalanceMap.set("robin",new BigNumber(1));
    },
    testRead: function () {
        //读取存储数据
        var balance = this.userBalanceMap.get("robin");
        this.key1Map.set("robin", balance.toString());
        this.key2Map.set("robin", balance.toString());
    }
};

module.exports = SampleContract;
```

2.3 Map数据遍历

在智能合约中如果需要遍历Map集合，可以采用如下方式：
定义两个map，分别是arrayMap，dataMap，arrayMap采用严格递增的计数器作为key，dataMap采用data的key作为key，详细参见set方法。
遍历实现参见forEach，先遍历arrayMap，得到dataKey，再对dataMap遍历。

> 由于Map遍历性能开销比较大，不建议对大数据量map进行遍历，建议按照limit，offset形式进行遍历，否者可能会由于数据过多，导致调用超时。

以下是遍历Map的例子：

```
"use strict";

var SampleContract = function () {
   LocalContractStorage.defineMapProperty(this, "arrayMap");
   LocalContractStorage.defineMapProperty(this, "dataMap");
   LocalContractStorage.defineProperty(this, "size");
};

SampleContract.prototype = {
```

```
    init: function () {
        this.size = 0;
    },

    set: function (key, value) {
        var index = this.size;
        this.arrayMap.set(index, key);
        this.dataMap.set(key, value);
        this.size +=1;
    },

    get: function (key) {
        return this.dataMap.get(key);
    },

    len:function(){
      return this.size;
    },

    forEach: function(limit, offset){
        limit = parseInt(limit);
        offset = parseInt(offset);
        if(offset>this.size){
           throw new Error("offset is not valid");
        }
        var number = offset+limit;
        if(number > this.size){
          number = this.size;
        }
        var result  = "";
        for(var i=offset;i<number;i++){
            var key = this.arrayMap.get(i);
            var object = this.dataMap.get(key);
            result += "index:"+i+" key:"+ key + " value:" +object+"_";
        }
        return result;
    }
};

module.exports = SampleContract;
```

下一篇预告

星云链Nebulas——5.通过RPC API和星云链交互