『0014』 - Solidity Types - 动态大小字节数组(Dynamically-sized byte array)

> 孔壹学院：国内区块链职业教育领先品牌
>
> 作者：黎跃春，区块链、高可用架构工程师
> 微信：liyc1215 QQ群：348924182 博客：http://liyuechun.org

一、Dynamically-sized byte array

- `string` 是一个动态尺寸的 `UTF-8` 编码字符串，它其实是一个特殊的可变字节数组，`string` 是引用类型，而非值类型。
- `bytes` 动态字节数组，引用类型。

根据经验，在我们不确定字节数据大小的情况下，我们可以使用 `string` 或者 `bytes`，而如果我们清楚的知道或者能够将字节书控制在 `bytes1` ~ `bytes32`，那么我们就使用 `bytes1` ~ `bytes32`，这样的话能够降低存储成本。

二、常规字符串 sting 转换为 bytes

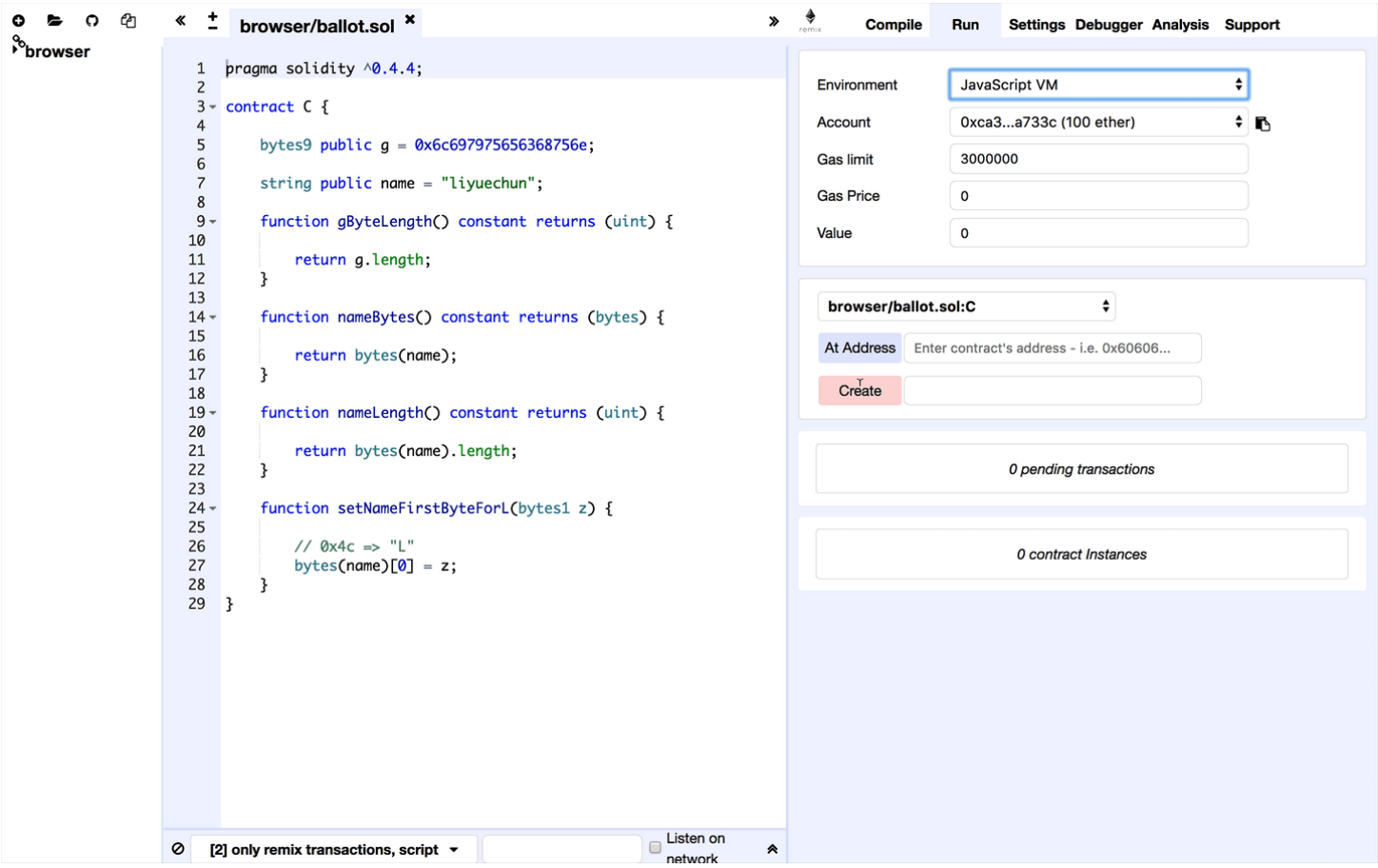
`string` 字符串中没有提供 `length` 方法获取字符串长度，也没有提供方法修改某个索引的字节码，不过我们可以将 `string` 转换为 `bytes`，再调用 `length` 方法获取字节长度，当然可以修改某个索引的字节码。

1、源码

```
pragma solidity ^0.4.4;

contract C {

    bytes9 public g = 0x6c697975656368756e;

    string public name = "liyuechun";

    function gByteLength() constant returns (uint) {

        return g.length;
```

```
    }

    function nameBytes() constant returns (bytes) {

        return bytes(name);
    }

    function nameLength() constant returns (uint) {

        return bytes(name).length;
    }

    function setNameFirstByteForL(bytes1 z) {

        // 0x4c => "L"
        bytes(name)[0] = z;
    }
}
```

2、效果图



3、说明

```
function nameBytes() constant returns (bytes) {

    return bytes(name);
}
```

`nameBytes` 这个函数的功能是将字符串 `name` 转换为 `bytes`，并且返回的结果为 `0x6c697975656368756e`。`0x6c697975656368756e` 一共为 `9字节`，也就是一个英文字母对应一个字节。

```
function nameLength() constant returns (uint) {

    return bytes(name).length;
}
```

我们之前讲过，`string` 字符串它并不提供 `length` 方法帮助我们返回字符串的长度，所以在 `nameLength` 方法中，我们将 `name` 转换为 `bytes`，然后再调用 `length` 方法来返回字节数，因为一个字节对应一个英文字母，所以返回的字节数量刚好等于字符串的长度。

```
function setNameFirstByteForL(bytes1 z) {

    // 0x4c => "L"
    bytes(name)[0] = z;
}
```

如果我们想将 `name` 字符串中的某个字母进行修改，那么我们直接通过 `x[k] = z` 的形式进行修改即可。`x` 是bytes类型的字节数组，`k` 是索引，`z` 是 `byte1` 类型的变量值。

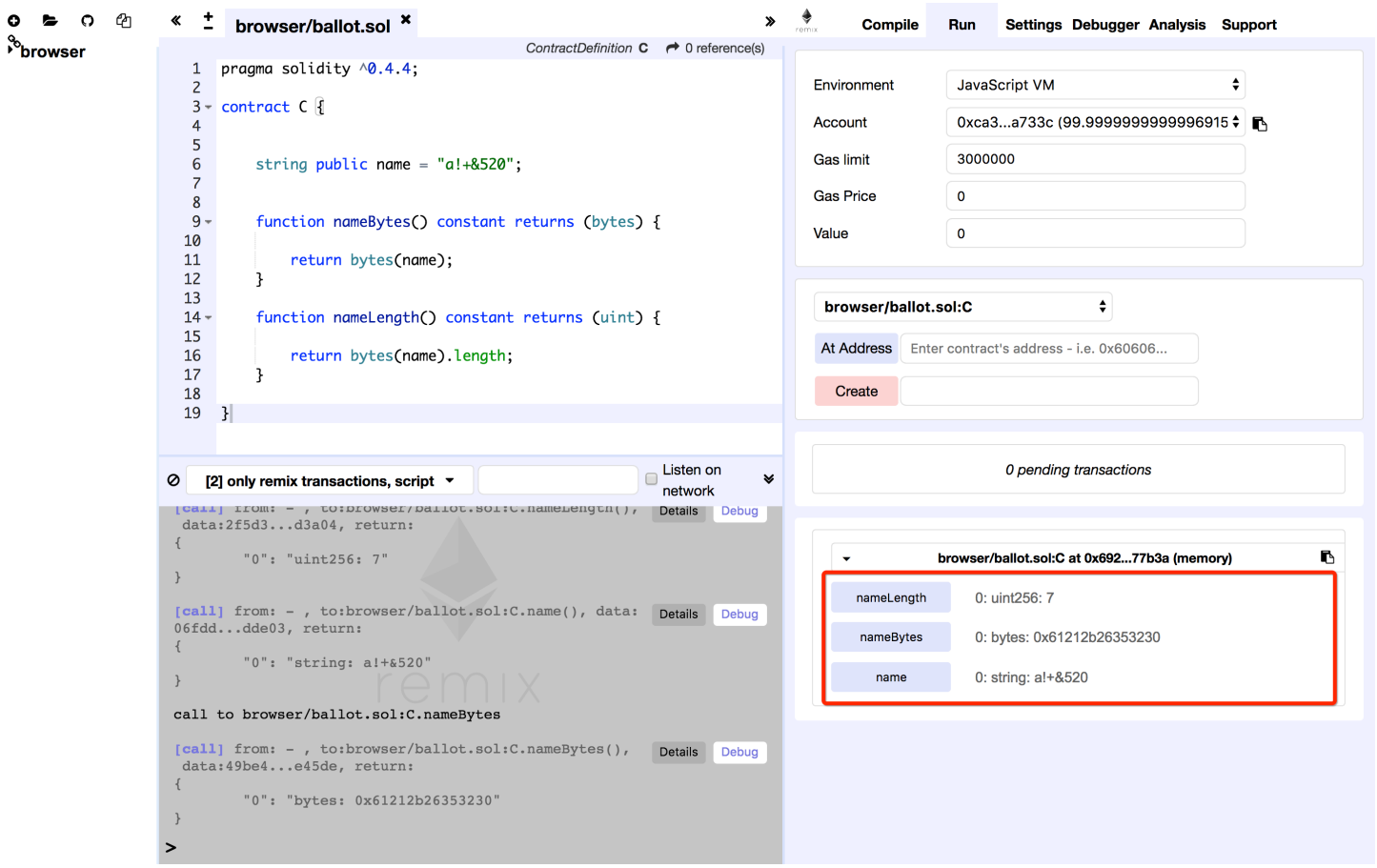
`setNameFirstByteForL` 方法中，我就将 `liyuechun` 中的首字母修改成 `L`，我传入的 `z` 的值为 `0x4c`，即大写的 `L`。

三、汉字字符串或特殊字符的字符串转换为bytes

1、特殊字符

```
pragma solidity ^0.4.4;

contract C {
```

```
    string public name = "a!+&520";


    function nameBytes() constant returns (bytes) {

        return bytes(name);
    }

    function nameLength() constant returns (uint) {

        return bytes(name).length;
    }

}
```

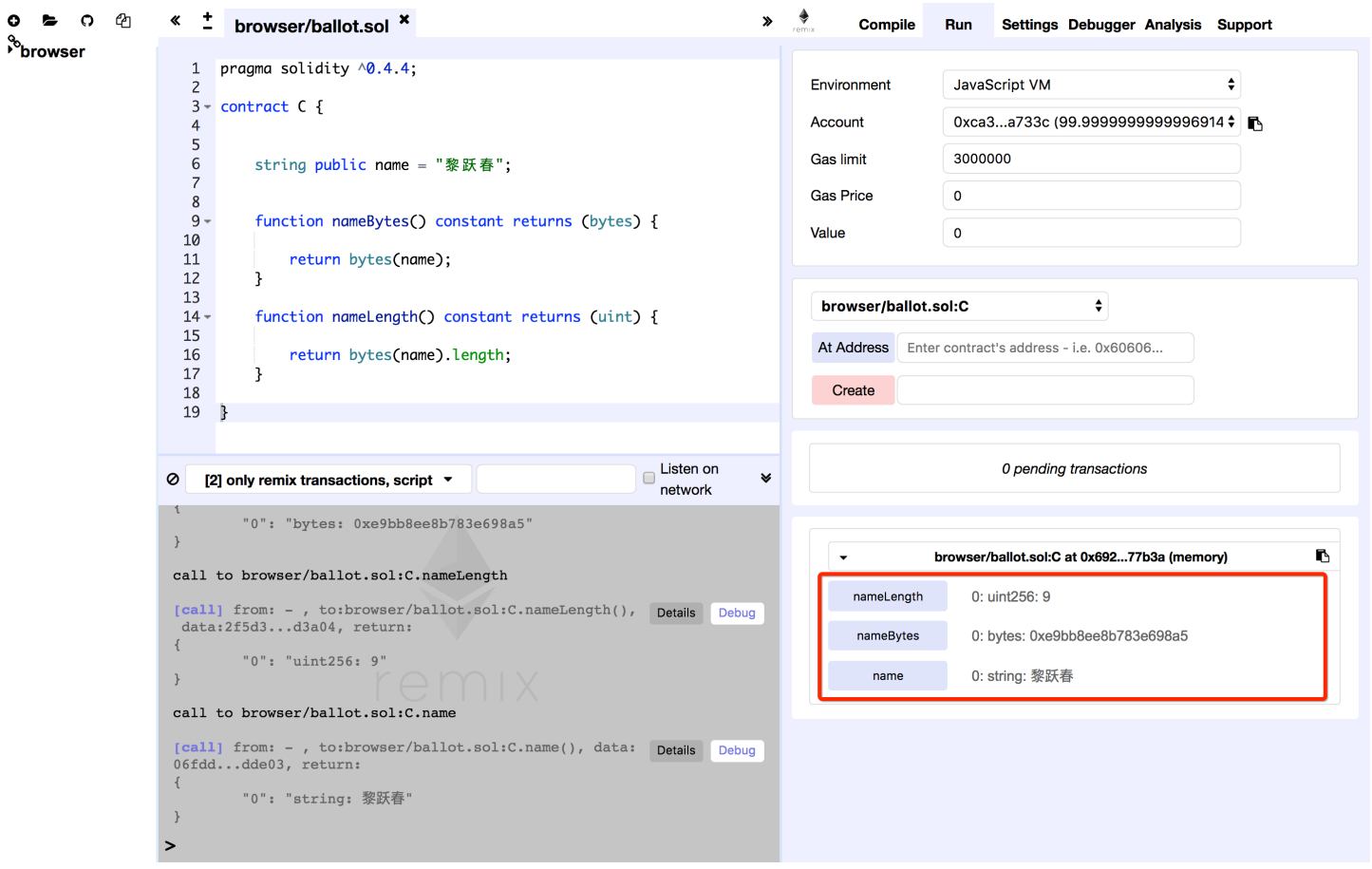


在这个案例中，我们声明了一个 `name` 字符串，值为 `a!+&520`，根据 `nameBytes` 和 `nameLength` 返回的结果中，我们不难看出，不管是 `字母`、`数字` 还是 `特殊符号`，每个字母对应一个 `byte（字节）`。

2、中文字符串

```
pragma solidity ^0.4.4;
```

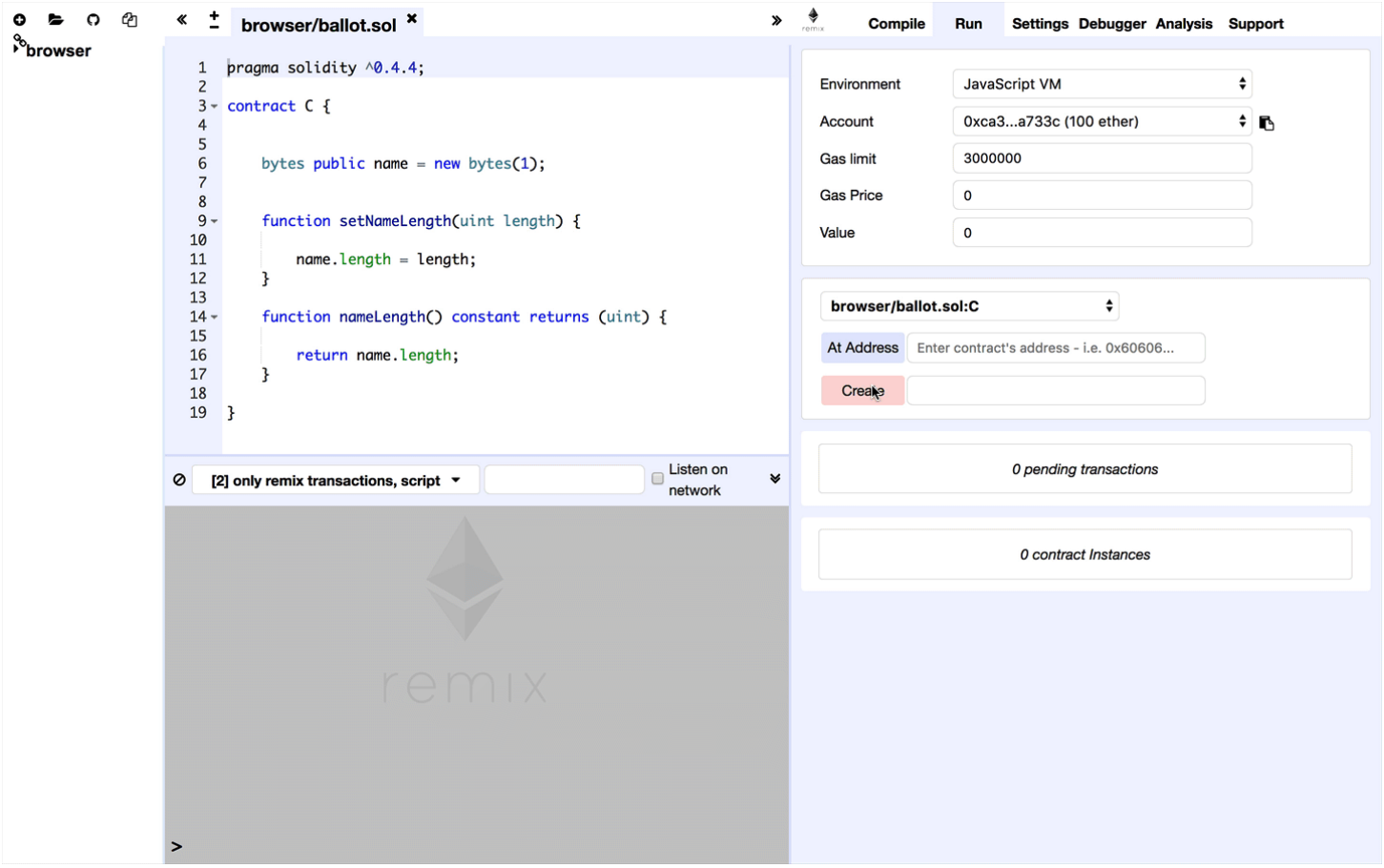
```
contract C {


    string public name = "黎跃春";


    function nameBytes() constant returns (bytes) {

        return bytes(name);
    }

    function nameLength() constant returns (uint) {

        return bytes(name).length;
    }

}
```



在上面的代码中，我们不难看出，`黎跃春`转换为`bytes`以后的内容为`0xe9bb8ee8b783e698a5`，一共`9个字节`。也就是一个汉字需要通过`3个字节`来进行存储。那么问题来了，以后我们取字符串时，字符串中最好不要带汉字，否则计算字符串长度时还得特殊处理。

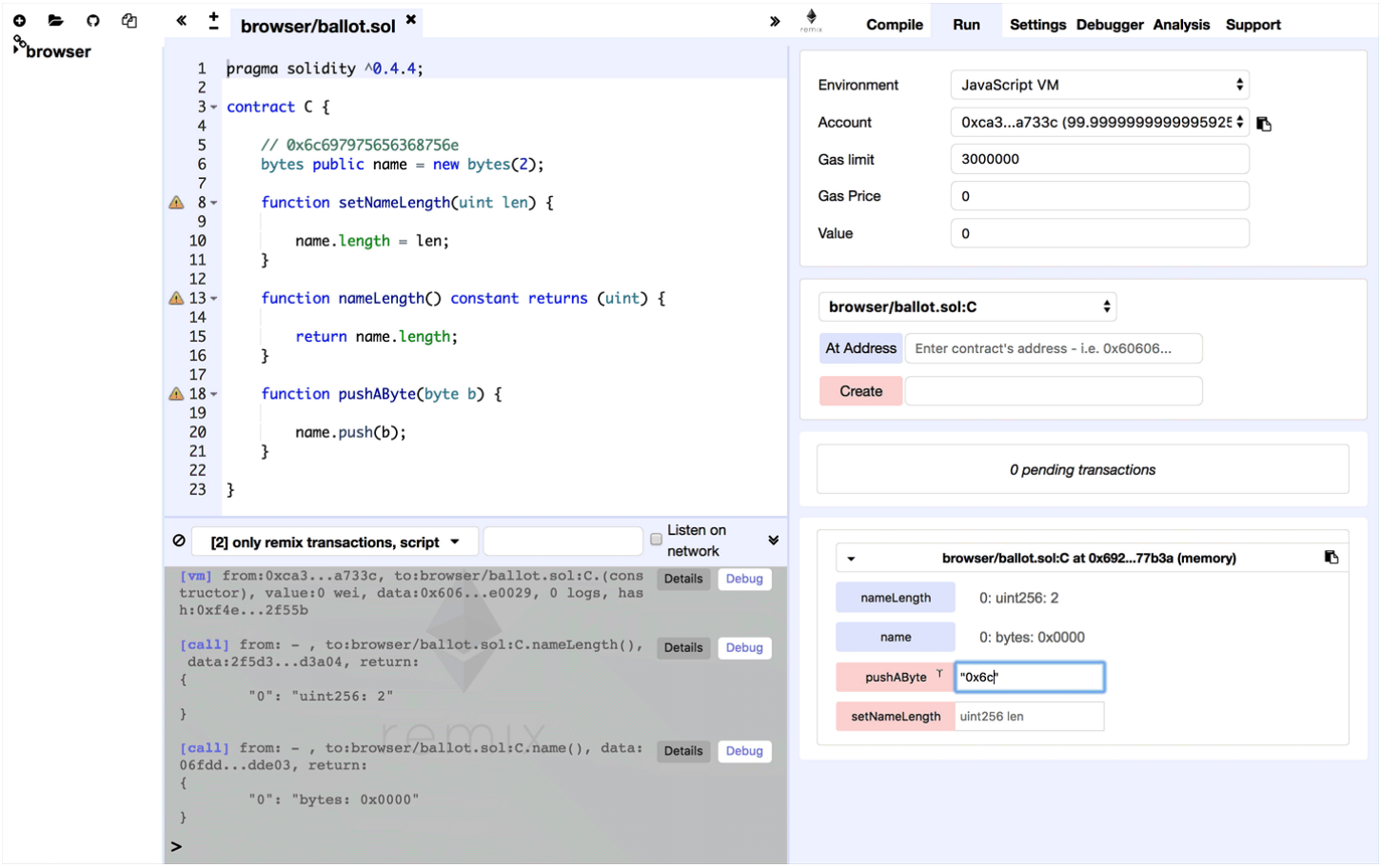
四、创建bytes字节数组

```
pragma solidity ^0.4.4;

contract C {


    bytes public name = new bytes(1);


    function setNameLength(uint length) {

        name.length = length;
    }

    function nameLength() constant returns (uint) {

        return name.length;
    }

}
```



五、bytes可变数组length和push两个函数的使用案例

```
pragma solidity ^0.4.4;

contract C {

    // 0x6c697975656368756e
    // 初始化一个两个字节空间的字节数组
    bytes public name = new bytes(2);

    // 设置字节数组的长度
    function setNameLength(uint len) {

        name.length = len;
    }

    // 返回字节数组的长度
    function nameLength() constant returns (uint) {

        return name.length;
    }

    // 往字节数组中添加字节
    function pushAByte(byte b) {

        name.push(b);
    }

}
```



说明：当字节数组的长度只有2时，如果你通过push往里面添加了一个字节，那么它的长度将变为3，当字节数组里面有3个字节，但是你通过length方法将其长度修改为2时，字节数组中最后一个字节将被从字节数组中移除。

五、总结

对比分析：

- 不可变字节数组

我们之前的文章中提到过如果我们清楚我们存储的字节大小，那么我们可以通过 `bytes1`，`bytes2`，`bytes3`，`bytes4`，……，`bytes32` 来声明字节数组变量，不过通过 `bytesI` 来声明的字节数组为不可变字节数组，字节不可修改，字节数组长度不可修改。

- 可变字节数组

我们可以通过 `bytes name = new bytes(length)` - `length` 为字节数组长度，来声明可变大小和可修改字节内容的可变字节数组。