加密僵尸：ERC721标准和加密收藏品

ERC721 标准, 多重继承

让我们来看一看 ERC721 标准：

```
contract ERC721 {
  event Transfer(address indexed _from, address indexed _to, uint256 _tokenId);
  event Approval(address indexed _owner, address indexed _approved, uint256 _tokenId);

  function balanceOf(address _owner) public view returns (uint256 _balance);
  function ownerOf(uint256 _tokenId) public view returns (address _owner);
  function transfer(address _to, uint256 _tokenId) public;
  function approve(address _to, uint256 _tokenId) public;
  function takeOwnership(uint256 _tokenId) public;
}
```

```
contract SatoshiNakamoto is NickSzabo, HalFinney {
  // 喷喷喷，宇宙的奥秘泄露了
}
```

正如你所见，当使用多重继承的时候，你只需要用逗号，来隔开几个你想要继承的合约。在上面的例子中，我们的合约继承自 NickSzabo 和 HalFin

ERC721: 转移标准

注意 ERC721 规范有两种不同的方法来转移代币：

```
function transfer(address _to, uint256 _tokenId) public;

function approve(address _to, uint256 _tokenId) public;
function takeOwnership(uint256 _tokenId) public;
```

1. 第一种方法是代币的拥有者调用transfer 方法，传入他想转移到的 address 和他想转移的代币的 _tokenId。
2. 第二种方法是代币拥有者首先调用 approve，然后传入与以上相同的参数。接着，该合约会存储谁被允许提取代币，通常存储到一个 mapping => address) 里。然后，当有人调用 takeOwnership 时，合约会检查 msg.sender 是否得到拥有者的批准来提取代币，如果是，则将代币转移给

你注意到了吗，transfer 和 takeOwnership 都将包含相同的转移逻辑，只是以相反的顺序。（一种情况是代币的发送者调用函数；另一种情况是代调用它）。

所以我们把这个逻辑抽象成它自己的私有函数 _transfer，然后由这两个函数来调用它。这样我们就不用写重复的代码了。

ERC721: 批准

记住，使用 approve 或者 takeOwnership 的时候，转移有2个步骤：

1. 你，作为所有者，用新主人的 address 和你希望他获取的 _tokenId 来调用 approve
2. 新主人用 _tokenId 来调用 takeOwnership，合约会检查确保他获得了批准，然后把代币转移给他。

因为这发生在2个函数的调用中，所以在函数调用之间，我们需要一个数据结构来存储什么人被批准获取什么。

预防溢出

需要将safemath.sol引入

比如，使用 SafeMath 库的时候，我们将使用 using SafeMath for uint256 这样的语法。SafeMath 库有四个方法 — add, sub, mul, 以及 div。以这样来让 uint256 调用这些方法：

```
using SafeMath for uint256;

uint256 a = 5;
uint256 b = a.add(3); // 5 + 3 = 8
uint256 c = a.mul(2); // 5 * 2 = 1
```

还可以是SafeMath32,SafeMath16等

SafeMath

```
library SafeMath {

  function mul(uint256 a, uint256 b) internal pure returns (uint256) {
    if (a == 0) {
      return 0;
    }
    uint256 c = a * b;
    assert(c / a == b);
    return c;
  }

  function div(uint256 a, uint256 b) internal pure returns (uint256) {
    // assert(b > 0); // Solidity automatically throws when dividing by 0
    uint256 c = a / b;
    // assert(a == b * c + a % b); // There is no case in which this doesn't hold
    return c;
  }

  function sub(uint256 a, uint256 b) internal pure returns (uint256) {
    assert(b <= a);
    return a - b;
  }

  function add(uint256 a, uint256 b) internal pure returns (uint256) {
    uint256 c = a + b;
    assert(c >= a);
    return c;
  }
}
```

首先我们有了 `library` 关键字 — 库和 合约很相似，但是又有一些不同。 就我们的目的而言，库允许我们使用 `using` 关键字，它可以自动把库的所有

给一个数据类型：

```
using SafeMath for uint;
// 这下我们可以为任何 uint 调用这些方法了
uint test = 2;
test = test.mul(3); // test 等于 6 了
test = test.add(5); // test 等于 11 了
```

注意 `mul` 和 `add` 其实都需要两个参数。 在我们声明了 `using SafeMath for uint` 后，我们用来调用这些方法的 `uint` 就自动被作为第一个参数传递进

例中就是 `test`)

另外：

```
function add(uint256 a, uint256 b) internal pure returns (uint256) {
  uint256 c = a + b;
  assert(c >= a);
  return c;
}

// 如果我们在`uint8` 上调用 `.add`。它将会被转换成 `uint256`.
// 所以它不会在 2^8 时溢出，因为 256 是一个有效的 `uint256`.
```

LEVI_104