LintCode 参考程序

Lionx

2016 年 4 月 19 日

目录

第一章　入门（Naive）

1.1 Problem ID: 228 Middle of Linked List

1.1.1 Description

Find the middle node of a linked list.

1.1.2 Example

Given 1->2->3, return the node with value 2.

Given 1->2, return the node with value 1.

1.1.3 Code

C++

```cpp
/**
 * Definition of ListNode
 * class ListNode {
 * public:
 *     int val;
 *     ListNode *next;
 *     ListNode(int val) {
 *         this->val = val;
 *         this->next = NULL;
 *     }
 * }
 */
class Solution{
public:
    /**
     * @param head: the head of linked list.
     * @return: a middle node of the linked list
     */
    ListNode *middleNode(ListNode *head) {
        // Write your code here
        if(head == NULL){
            return NULL;
        }
        ListNode *fast = head;
        ListNode *slow = head;
        while(fast->next != NULL && fast->next->next != NULL){
            slow = slow->next;
            fast = fast->next->next;
        }
        return slow;
    }
};
```

Python

```python
"""
Definition of ListNode
class ListNode(object):

    def __init__(self, val, next=None):
        self.val = val
        self.next = next
"""

class Solution:
    # @param head: the head of linked list.
    # @return: a middle node of the linked list
    def middleNode(self, head):
        # Write your code here
        if head is None:
            return None
        slow = head;
        fast = head;
        while fast.next is not None and fast.next.next is not None:
            slow = slow.next
            fast = fast.next.next
        return slow
```

第二章　容易（Easy）

2.1 Problem ID: 496 Toy Factory

2.1.1 Description

Factory is a design pattern in common usage. Please implement a ToyFactory which can generate proper toy based on the given type.

2.1.2 Example

```
ToyFactory tf = ToyFactory();
Toy toy = tf.getToy('Dog');
toy.talk();
>> Wow

toy = tf.getToy('Cat');
toy.talk();
>> Meow
```

2.1.3 Code

C++

```cpp
/**
 * Your object will be instantiated and called as such:
 * ToyFactory* tf = new ToyFactory();
 * Toy* toy = tf->getToy(type);
 * toy->talk();
 */
class Toy {
public:
    virtual void talk() const=0;
};

class Dog: public Toy {
    // Write your code here
    void talk() const{
        cout << "Wow" << endl;
    }
};

class Cat: public Toy {
    // Write your code here
    void talk() const{
        cout << "Meow" << endl;
    }
};

class ToyFactory {
```

```cpp
public:
    /**
     * @param type a string
     * @return Get object of the type
     */
    Toy* getToy(string& type) {
        // Write your code here
        if(type == "Dog"){
            return new Dog();
        }
        if(type == "Cat"){
            return new Cat();
        }
        return NULL;
    }
};
```

Python

```python
"""
Your object will be instantiated and called as such:
ty = ToyFactory()
toy = ty.getToy(type)
toy.talk()
"""
class Toy:
    def talk(self):
        raise NotImplementedError('This method should have implemented.')

class Dog(Toy):
    # Write your code here
    def talk(self):
        print "Wow"

class Cat(Toy):
    # Write your code here
    def talk(self):
        print "Meow"


class ToyFactory:
    # @param {string} shapeType a string
    # @return {Toy} Get object of the type
    def getToy(self, type):
        # Write your code here
        if type == "Dog":
            return Dog()
        if type == "Cat":
            return Cat()
        return None
```

2.2 Problem ID: 497 Shape Factory

2.2.1 Description

Factory is design pattern in common usage. Implement a ShapeFactory that can generate correct shape.

2.2.2 Example

```
ShapeFactory sf = new ShapeFactory();
Shape shape = sf.getShape("Square");
shape.draw();
>>  ----
>> |    |
>> |    |
>>  ----

shape = sf.getShape("Triangle");
shape.draw();
>>   /\
>>  /  \
>> /____\

shape = sf.getShape("Rectangle");
shape.draw();
>>  ----
>> |    |
>>  ----
```

2.2.3 Code

C++

```
/**
 * Your object will be instantiated and called as such:
 * ShapeFactory* sf = new ShapeFactory();
 * Shape* shape = sf->getShape(shapeType);
 * shape->draw();
 */
class Shape {
public:
    virtual void draw() const=0;
};

class Rectangle: public Shape {
    // Write your code here
    void draw() const{
        cout << "␣----" << endl << "|␣␣␣␣|" << endl << "␣----" << endl;
```

```cpp
    }
};

class Square: public Shape {
    // Write your code here
     void draw() const{
        cout << " ----" << endl << "|    |" << endl
       << "|    |" << endl << " ----" << endl;
     }
};

class Triangle: public Shape {
    // Write your code here
     void draw() const{
        cout << "  /\\" << endl << " /  \\" << endl << "/____\\" << endl;
    }
};

class ShapeFactory {
public:
    /**
     * @param shapeType a string
     * @return Get object of type Shape
     */
    Shape* getShape(string& shapeType) {
        // Write your code here
        if(shapeType == "Square"){
            return new Square();
        }
        if(shapeType == "Rectangle"){
            return new Rectangle();
        }
        if(shapeType == "Triangle"){
            return new Triangle();
        }
        return NULL;
    }
};
```

Python

```python
"""
Your object will be instantiated and called as such:
sf = ShapeFactory()
shape = sf.getShape(shapeType)
shape.draw()
"""
class Shape:
    def draw(self):
        raise NotImplementedError('This method should have implemented.')

class Triangle(Shape):
    # Write your code here.
    def draw(self):
        print "␣␣/\\"
        print "␣/␣␣\\"
        print "/____\\"

class Rectangle(Shape):
    # Write your code here
    def draw(self):
        print "␣----"
        print "|␣␣␣␣|"
        print "␣----"

class Square(Shape):
    # Write your code here
    def draw(self):
        print "␣----"
        print "|␣␣␣␣|"
        print "|␣␣␣␣|"
        print "␣----"

class ShapeFactory:
    # @param {string} shapeType a string
    # @return {Shape} Get object of type Shape
    def getShape(self, shapeType):
        # Write your code here
        if shapeType == "Square":
            return Square()
        if shapeType == "Triangle":
            return Triangle()
        if shapeType == "Rectangle":
            return Rectangle()
        return None
```

第三章　中等（Medium）

第四章　困难（Hard）

第五章　超难（Super）