Traits基础

黄天羽、嵩天

www.python123.org

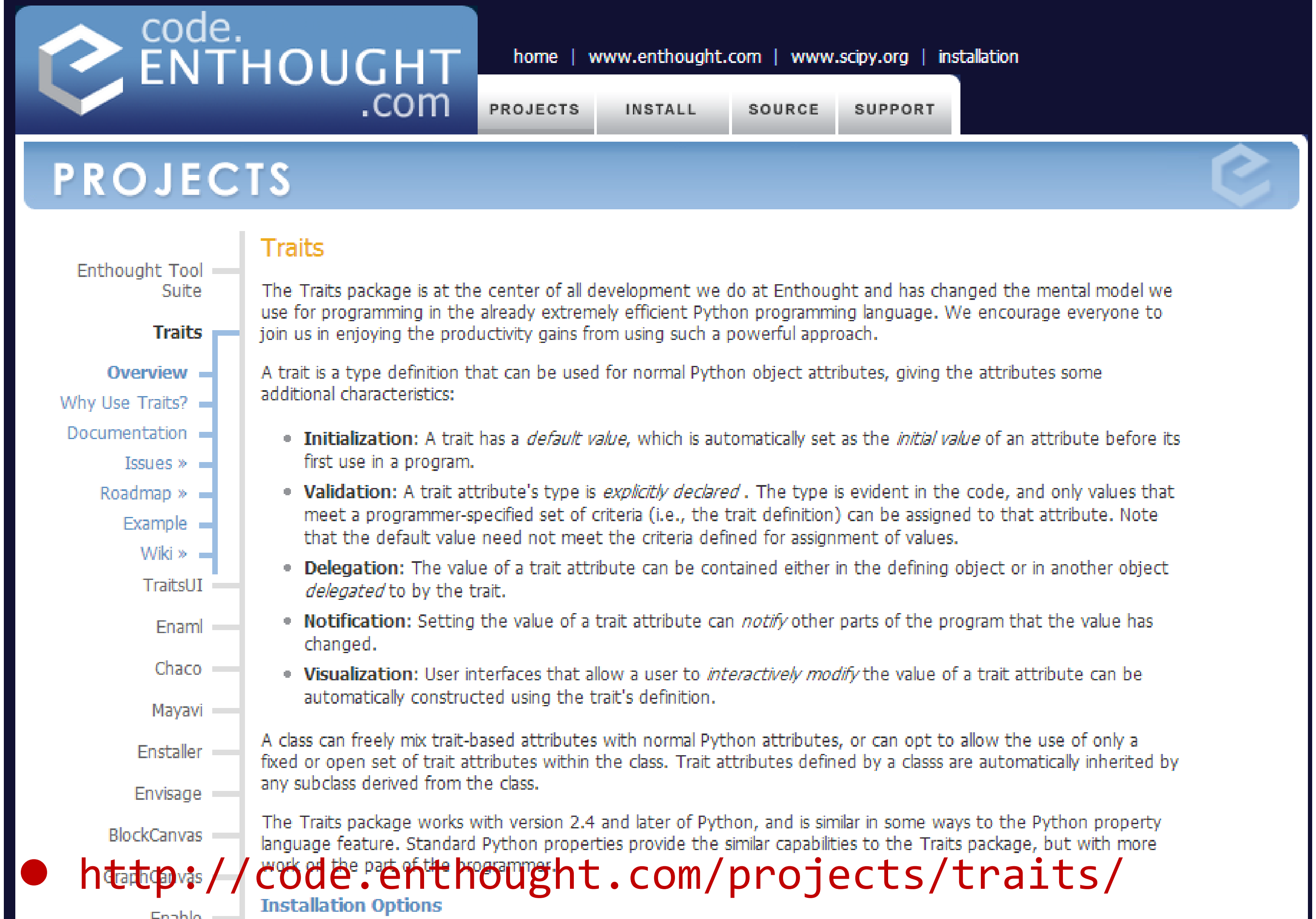
Traits介绍

Traits的背景

- Python作为一种动态编程语言，它的变量没有类型，这种灵活性给快速开发带来便利，不过也存在一定得缺点。
- 例如：颜色属性
  - ‘red’ 字符串 ‘abc’ 合法颜色值吗？
  - 0xff0000 整数
  - （255，0，0） 元组

Traits的背景

- Traits库可以为Python添加类型定义
- Traits属性解决color类型问题：
  - 接受能表示颜色的各种类型的值；
  - 赋值为不能表达颜色的值时，它能够立即捕捉到错误，提供一个错误报告，告诉用户能够接受什么值；
  - 它提供一个内部、标准的颜色表达方式。


code.ENTHOUGHT.com

home | www.enthought.com | www.scipy.org | installation

PROJECTS INSTALL SOURCE SUPPORT

PROJECTS

Enthought Tool Suite
Traits
Overview
Why Use Traits?
Documentation
Issues »
Roadmap »
Example
Wiki »
TraitsUI
Enaml
Chaco
Mayavi
Enstaller
Envisage
BlockCanvas
GraphCanvas
Enable

Traits

The Traits package is at the center of all development we do at Enthought and has changed the mental model we use for programming in the already extremely efficient Python programming language. We encourage everyone to join us in enjoying the productivity gains from using such a powerful approach.

A trait is a type definition that can be used for normal Python object attributes, giving the attributes some additional characteristics:

- **Initialization:** A trait has a *default value*, which is automatically set as the *initial value* of an attribute before its first use in a program.
- **Validation:** A trait attribute's type is *explicitly declared* . The type is evident in the code, and only values that meet a programmer-specified set of criteria (i.e., the trait definition) can be assigned to that attribute. Note that the default value need not meet the criteria defined for assignment of values.
- **Delegation:** The value of a trait attribute can be contained either in the defining object or in another object *delegated* to by the trait.
- **Notification:** Setting the value of a trait attribute can *notify* other parts of the program that the value has changed.
- **Visualization:** User interfaces that allow a user to *interactively modify* the value of a trait attribute can be automatically constructed using the trait's definition.

A class can freely mix trait-based attributes with normal Python attributes, or can opt to allow the use of only a fixed or open set of trait attributes within the class. Trait attributes defined by a classs are automatically inherited by any subclass derived from the class.

The Traits package works with version 2.4 and later of Python, and is similar in some ways to the Python property language feature. Standard Python properties provide the similar capabilities to the Traits package, but with more work on the part of the programmer.

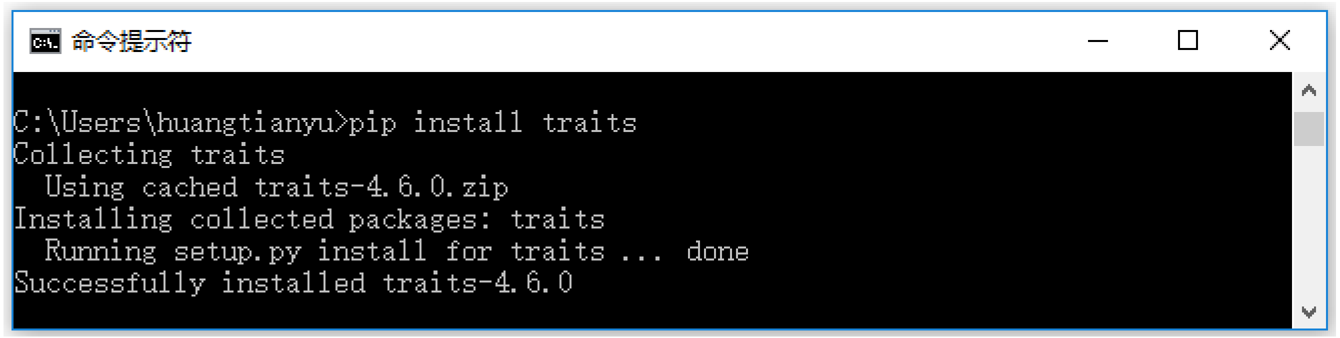
Installation Options

- http://code.enthought.com/projects/traits/

Traits库的安装

Win平台：“以管理员身份运行”cmd

在下载目录执行pip install traits



```
C:\Users\huangtianyu>pip install traits
Collecting traits
  Using cached traits-4.6.0.zip
Installing collected packages: traits
  Running setup.py install for traits ... done
Successfully installed traits-4.6.0
```

Successfully installed traits-4.6.0

Traits库的安装

测试traits是否安装成功？

所有拥有trairt属性的类都需要从HasTraits继承

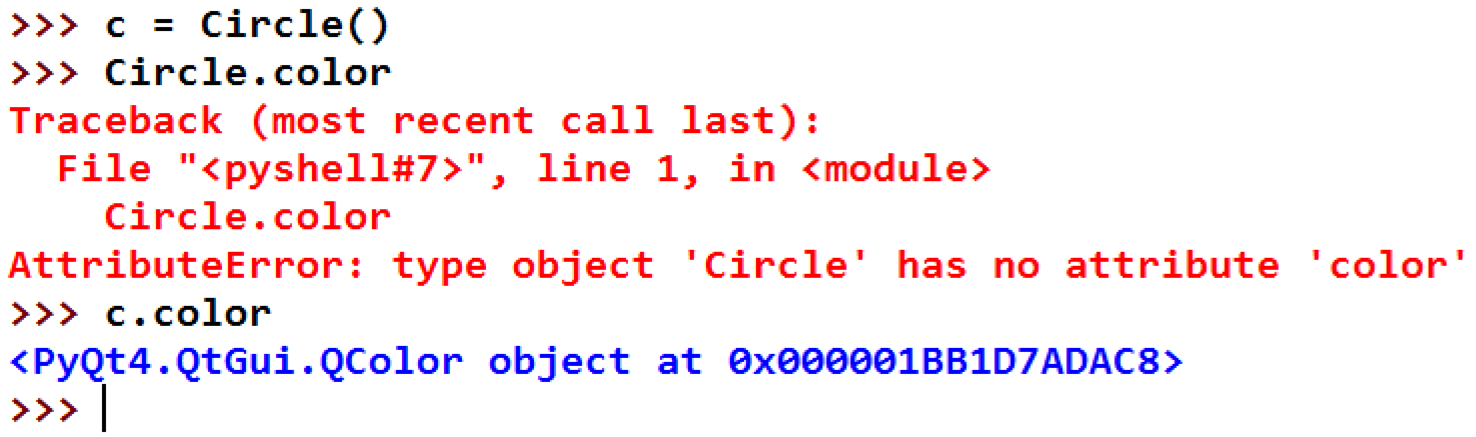
```
from traits.api import HasTraits
```

Traits属性表示颜色的例子

```
from traits.api import HasTraits, Color
class Circle(HasTraits):
    color = Color
```

Corlor是一个Trait类型，在Circle类中用它定义了一个color属性。

Traits属性表示颜色的例子



```
>>> c = Circle()
>>> Circle.color
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    Circle.color
AttributeError: type object 'Circle' has no attribute 'color'
>>> c.color
<PyQt4.QtGui.QColor object at 0x000001BB1D7ADAC8>
>>> 
```

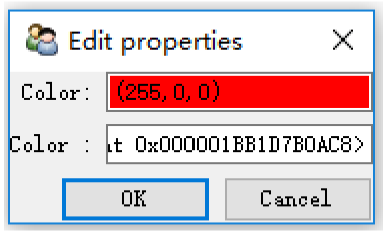
Trait属性可以像类的属性来定义，像实例的属性来使用

Traits属性表示颜色的例子

```
>>> c.color = 'red'
>>> c.color.getRgb()
(255, 0, 0, 255)
>>> c.color = 'abc'
Traceback (most recent call last):
  File "<pyshell#35>", line 1, in <module>
    c.color = 'abc'
  File "C:\Python36\lib\site-packages\traits\trait_handlers.py", line 172, in error
    value )
traits.trait_errors.TraitError: The 'color' trait of a Circle instance must be a st
ring of the form (r,g,b) or (r,g,b,a) where r, g, b, and a are integers from 0 to 2
55, a QColor instance, a Qt.GlobalColor, an integer which in hex is of the form 0xR
RGGBB, a string of the form #RGB, #RRGGBB, #RRRGGGBBB or #RRRRGGGGBBBB or 'aliceblu
e' or 'antiquewhite' or 'aqua' or 'aquamarine' or 'azure' or 'beige' or 'bisque' or
'black' or 'blanchedalmond' or 'blue' or 'blueviolet' or 'brown' or 'burlywood' or
```

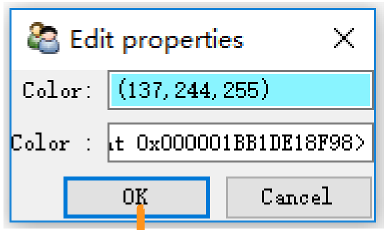
Traits属性表示颜色的例子

```
>>> c.configure_traits()
```



```
Color:  (255, 0, 0)
Color : ıt 0x000001BB1D7B0AC8>
```

交互的选择颜色



```
Color:  (137, 244, 255)
Color : ıt 0x000001BB1DE18F98>
```

点击OK按钮

```
>>> c.configure_traits()
True
>>> c.color.getRgb()
(137, 244, 255, 255)
>>> |
```

Trait属性的功能

Trait属性的功能

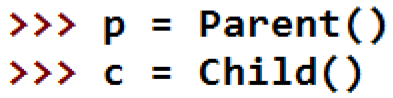
- Trait库为Python对象的属性增加了类型定义功能
- 还提供了功能：
    - 初始化：每个Trait属性都有自己的默认值
    - 验证：Trait属性有明确的类型定义，满足定义的值才能赋值给属性
    - 代理：Trait属性值可以代理给其他对象的属性。
    - 监听：Trait属性值发生变化时，运行事先指定的函数
    - 可视化：拥有Trait属性的对象，可生成编辑Trait属性的界面

Trait属性的功能

```
from traits.api import Delegate, HasTraits, Instance, Int, Str

class Parent (HasTraits):
    #初始化:last_name被初始化为'Zhang'
    last_name = Str('Zhang')

class Child (HasTraits):
    age = Int
    #验证:father属性的值必须是Parent类的实例
    father = Instance(Parent)
    #代理:Child实例的last_name属性代理给其father属性的last_name
    last_name = Delegate('father')
    #监听:当age属性的值被修改时，下面的函数将被运行
    def _age_changed (self, old, new):
        print('Age changed from %s to %s' % (old, new))
```

Trait属性的功能

创建两个类的实例：



```
>>> p = Parent()
>>> c = Child()
```

Trait属性的功能

没有设置c.father属性，无法获得它的last_name属性：

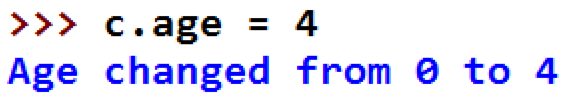
```
>>> c.last_name
Traceback (most recent call last):
  File "<pyshell#46>", line 1, in <module>
    c.last_name
AttributeError: 'NoneType' object has no attribute 'last_name'
```

设置father属性后，可以得到c的last_name属性：

```
>>> c.father = p
>>> c.last_name
'Zhang'
```

Trait属性的功能

c的age属性值发生变化时，将触发其监听函数_age_changed()：

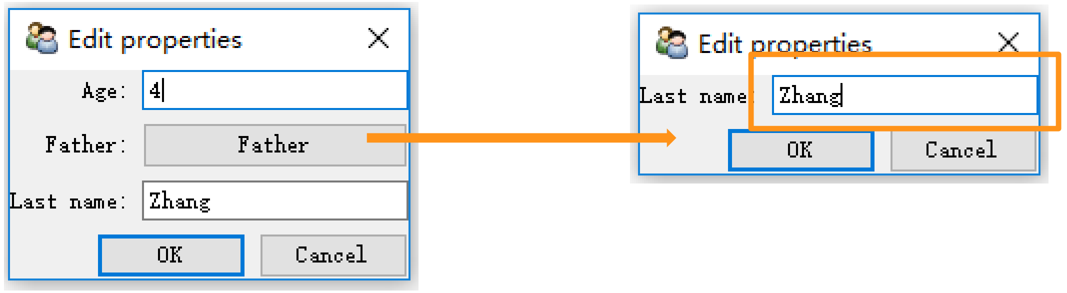


```
>>> c.age = 4
Age changed from 0 to 4
```

Trait属性的功能

调用configure_traits()显示一个修改属性值的对话框

```
>>> c.configure_traits()
```

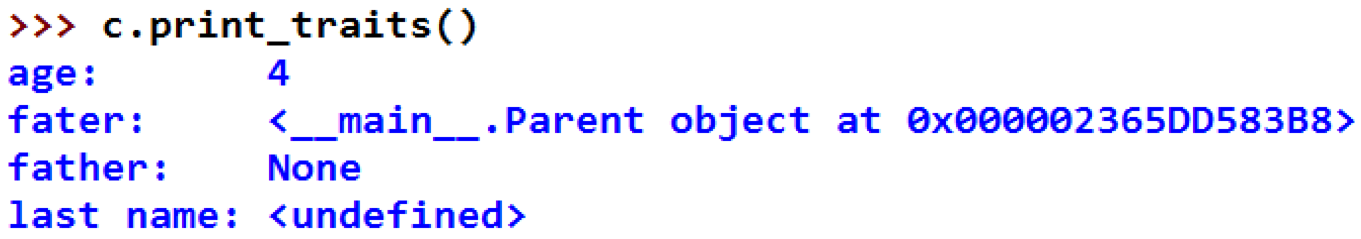


自动生成的界面，属性按照英文名排序。

由于father的属性是Parent类的对象，所以界面中以一个按钮来表示。

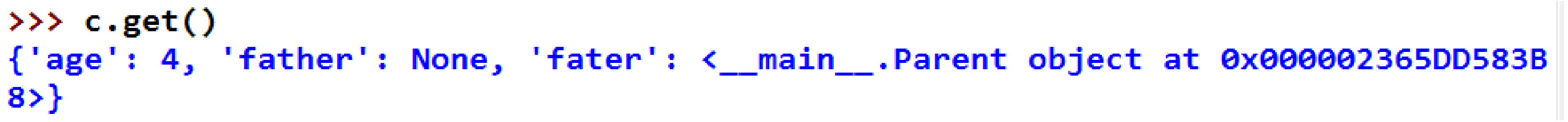
Trait的其它方法

调用print_traits()方法输出所有trait属性与其值：



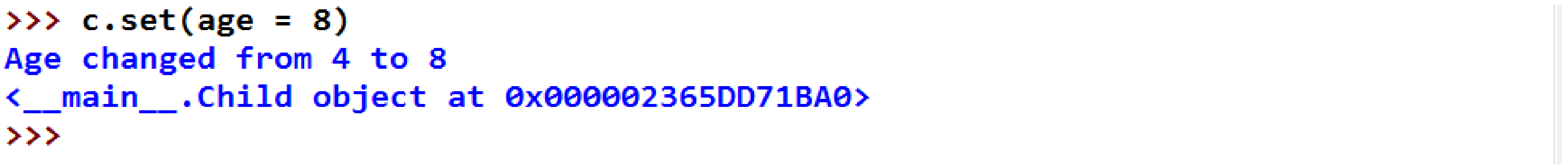
```
>>> c.print_traits()
age:       4
fater:     <__main__.Parent object at 0x000002365DD583B8>
father:    None
last name: <undefined>
```

调用get()方法获得描述对象所有trait属性的字典：

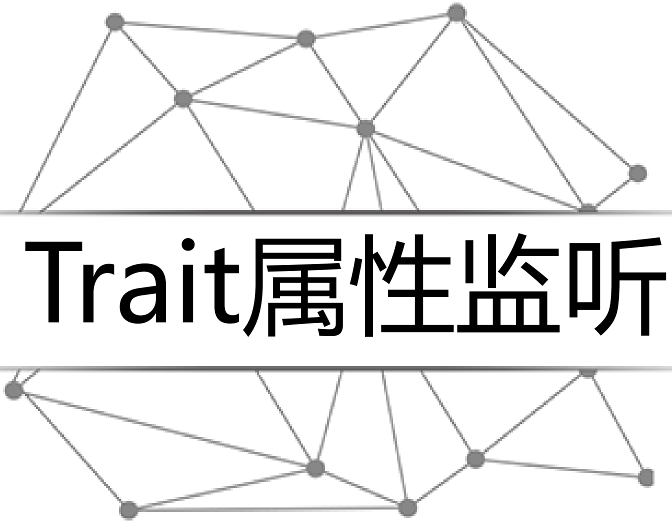


```
>>> c.get()
{'age': 4, 'father': None, 'fater': <__main__.Parent object at 0x000002365DD583B
8>}
```

调用set()方法trait属性的值：



```
>>> c.set(age = 8)
Age changed from 4 to 8
<__main__.Child object at 0x000002365DD71BA0>
>>>
```



Trait属性监听

Trait属性监听

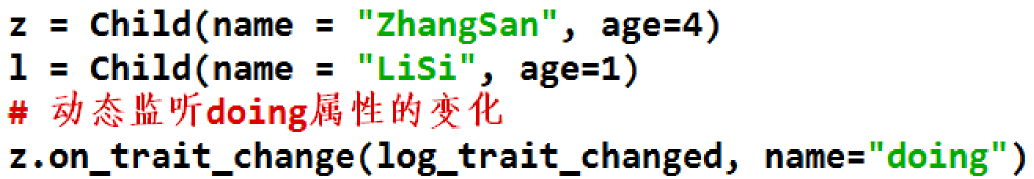
两种监听模式：

- 静态监听
- 动态监听

```python
class Child ( HasTraits ):
    name = Str
    age = Int
    doing = Str

    def __str__(self):
        return "%s<%x>" % (self.name, id(self))
    # 静态监听age属性的变化
    def _age_changed ( self, old, new ):
        print ("%s.age changed: form %s to %s" % (self, old, new))
    # 静态监听任何Trait属性的变化
    def _anytrait_changed(self, name, old, new):
        print ("anytrait changed: %s.%s from %s to %s" % (self, name, old, new))

def log_trait_changed(obj, name, old, new):
    print ("log: %s.%s changed from %s to %s" % (obj, name, old, new))

z = Child(name = "ZhangSan", age=4)
l = Child(name = "LiSi", age=1)
# 动态监听doing属性的变化
z.on_trait_change(log_trait_changed, name="doing")
```

Trait属性监听

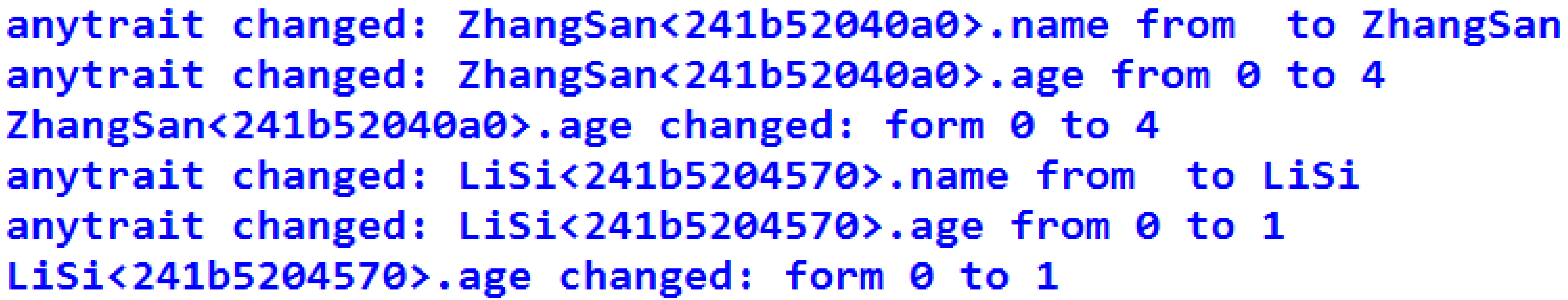
静态监听：



```
z = Child(name = "ZhangSan", age=4)
l = Child(name = "LiSi", age=1)
# 动态监听doing属性的变化
z.on_trait_change(log_trait_changed, name="doing")
```

运行程序，观察程序运行结果！

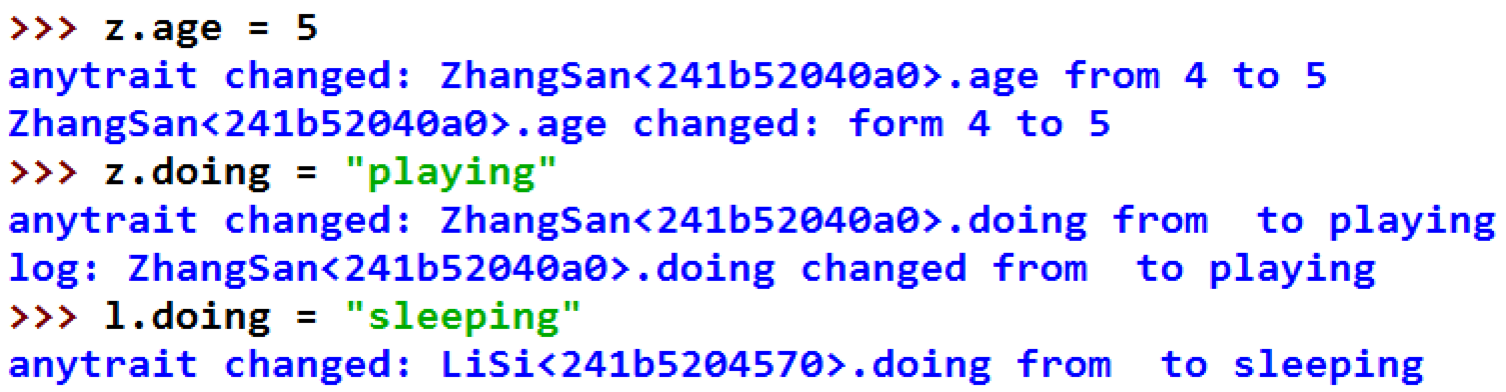
Trait属性监听



```
anytrait changed: ZhangSan<241b52040a0>.name from  to ZhangSan
anytrait changed: ZhangSan<241b52040a0>.age from 0 to 4
ZhangSan<241b52040a0>.age changed: form 0 to 4
anytrait changed: LiSi<241b5204570>.name from  to LiSi
anytrait changed: LiSi<241b5204570>.age from 0 to 1
LiSi<241b5204570>.age changed: form 0 to 1
```

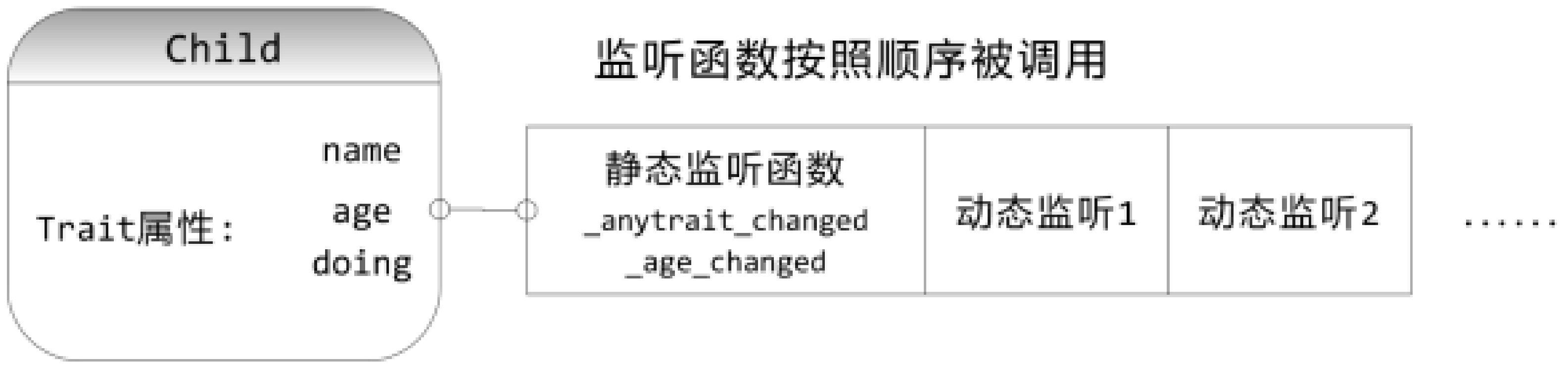
Trait属性监听

动态监听：



```
>>> z.age = 5
anytrait changed: ZhangSan<241b52040a0>.age from 4 to 5
ZhangSan<241b52040a0>.age changed: form 4 to 5
>>> z.doing = "playing"
anytrait changed: ZhangSan<241b52040a0>.doing from  to playing
log: ZhangSan<241b52040a0>.doing changed from  to playing
>>> l.doing = "sleeping"
anytrait changed: LiSi<241b5204570>.doing from  to sleeping
```

Trait属性监听



Trait属性的监听函数调用顺序

Trait属性监听

静态监听函数的几种形式：

```
_age_changed(self)

_age_changed(self, new)

_age_changed(self, old, new)

_age_changed(self, name, old, new)
```

Trait属性监听

动态监听函数的几种形式：

```
observer()

observer(new)

observer(name, new)

observer(obj, name, new)

observer(obj, name, old, new)
```

Trait属性监听

```
@on_trait_change(names)

def any_method_name(self, …)

        … …
```

Event和button属性

Event和Button属性

Event属性与其他Trait属性的区别

	Event属性	Trait属性
触发与其绑定的监听事件	当任何值对Event属性赋值时；不存储属性值，所赋值将会被忽略；如果试图获取属性值会产生异常	只有在值发生改变时
监听函数名	`_event_fired()`	`_trait_changed()`

Event和Button属性

Button属性：

- 具备Event事件处理功能
- 通过TraitsUI库,自动生成界面中的按钮控件

Event监听

```
from traits.api import HasTraits, Str, Int, Event

class Child(HasTraits):
        name = Str("ZhangSan")
        age = Int(4)
        Infoupdated = Event
```

Event监听

```
from traits.api import on_trait_change

@on_trait_change(“name, age”)
def Info_changed(self):
    self.Infoupdated = True
```

Event监听

```
def _Infoupdated_fired(self):
        self.reprint()
```

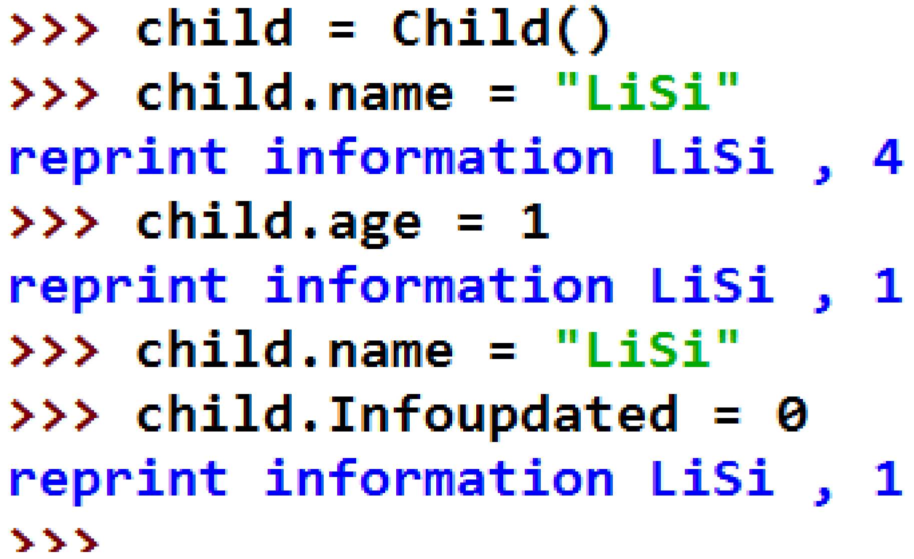
Event监听

```
def reprint(self):
        print (“reprint Information: s%, s%: ”% (self.name,self.age) )
```

Event监听

```
from traits.api import HasTraits, Str, Int, Event, on_trait_change

class Child (HasTraits):
    name = Str("ZhangSan")
    age = Int(4)
    Infoupdated = Event
    # 对_Info_changed()方法进行修饰
    @on_trait_change("name,age")
    def _Info_changed (self):
        self.Infoupdated = True
    # Inforupdated事件处理方法
    def _Infoupdated_fired(self):
        self.reprint()
    def reprint(self):
        print ("reprint information %s , %s" % (self.name, self.age))
```

Event监听



```
>>> child = Child()
>>> child.name = "LiSi"
reprint information LiSi , 4
>>> child.age = 1
reprint information LiSi , 1
>>> child.name = "LiSi"
>>> child.Infoupdated = 0
reprint information LiSi , 1
>>>
```

Property属性

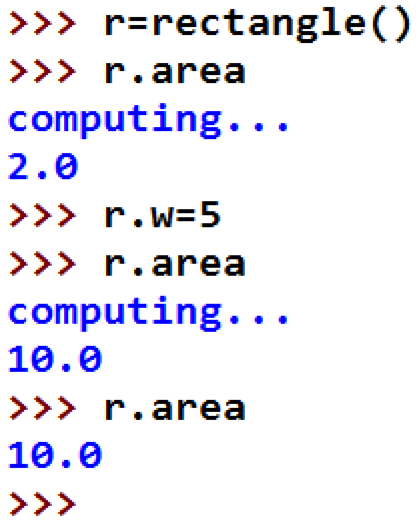
Property属性

```
from traits.api import Property
```

Property属性

```python
from traits.api import HasTraits, Float, Property, cached_property
class rectangle(HasTraits):
    w = Float(1.0)
    h = Float(2.0)
    area = Property(depends_on = ['w', 'h'])

    @cached_property
    def _get_area(self):
        print("computing...")
        return (self.w * self.h)
```

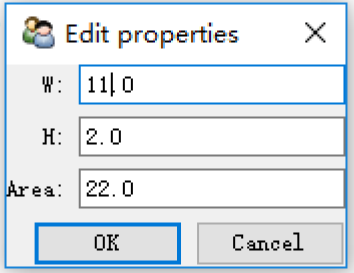
Property属性



```
>>> r=rectangle()
>>> r.area
computing...
2.0
>>> r.w=5
>>> r.area
computing...
10.0
>>> r.area
10.0
>>>
```

Property属性

```
>>> r.edit_traits()
computing...
<traitsui.ui.UI object at 0x000001C80AC35BA0>
>>> computing...
```



Edit properties

W: 11.0

H: 2.0

Area: 22.0