More C++ Idioms/Capability Query

Contents

Capability Query

Intent

To check at runtime whether an object supports an interface

Also Known As

Motivation

Separating interface from implementation is a good object oriented software design practice. In C++, the Interface Class idiom is used to separate interface from implementation and invoke the public methods of any abstraction using runtime polymorphism. Extending the example in the interface class idiom, a concrete class may implement multiple interfaces as shown below.

```
class Shape { // An interface class
   public:
    virtual ~Shape();
    virtual void draw() const = 0;
    //...
};
class Rollable { // One more interface class
  public:
    virtual ~Rollable();
    virtual void roll() = 0;
};
class Circle : public Shape, public Rollable { // circles roll - concrete class
    //...
    void draw() const;
    void roll();
    //...
};
class Square : public Shape { // squares don't roll - concrete class
```

```
    //...
    void draw() const;
    //...
};
```

Now if we are given a container of pointers to abstract Rollable class, we can simply invoke the roll function on every pointer, as described in the interface class idiom.

```
std::vector<Rollable *> rollables;
//  Fill up rollables vector somehow.
for (vector<Rollable *>::iterator iter (rollables.begin());
     iter != rollables.end();
     ++iter)
{
  iter->roll();
}
```

Sometimes it is not possible to know in advance whether or not an object implements a particular interface. Such a situation commonly arises if an object inherits from multiple interface classes. To discover the presence or absence of the interface at runtime, capability query is used.

Solution and Sample Code

In C++, a capability query is typically expressed as a `dynamic_cast` between unrelated types .

```
Shape *s = getSomeShape();
if (Rollable *roller = dynamic_cast<Rollable *>(s))
  roller->roll();
```

This use of `dynamic_cast` is often called a **cross-cast,** because it attempts a conversion across a hierarchy, rather than up or down a hierarchy. In our example hierarchy of shapes and rollables, `dynamic_cast` to `Rollable` will succeed only for `Circle` and not for `Square` as the later one does not inherit from `Rollable` interface class.

Excessive use of capability queries is often an indication of bad object-oriented design.

Known Uses

Acyclic Visitor Pattern (http://www.objectmentor.com/resources/articles/acv.pdf) - Robert C. Martin.

Related Idioms

- Interface Class
- Inner Class

References

Capability Queries - C++ Common Knowledge by Stephen C. Dewhurst

title=More_C%2B%2B_Idioms/Capability_Query&oldid=2495301"
