观察者模式

姚永舟

2024.11.29

动机

在一个系统中，难免会有一些类之间存在相互协作的关系，如果我们单独地去维护这些对象之间的一致性，各个类之间就可能会紧密耦合。采用观察者模式就是为了避免这样的问题产生。

简介

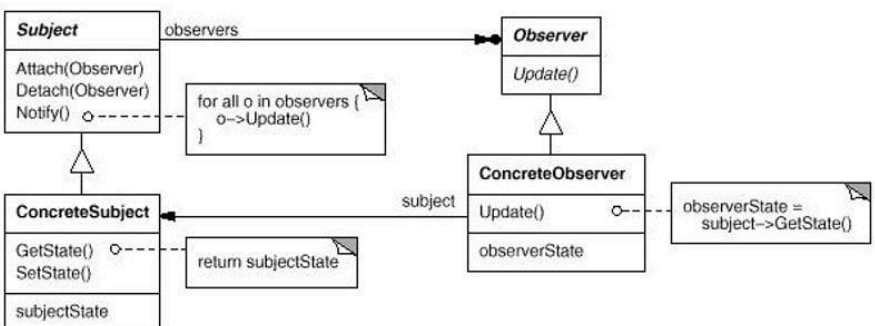
观察者模式指多个对象间存在一对多的依赖关系，当一个对象的状态发生改变时，所有依赖于它的对象都得到通知并被自动更新。这种模式有时又称作发布-订阅模式、模型-视图模式。

结构

抽象目标：提供一个集合类用于保存观察者对象，并包含注册、删除观察者的方法以及通知所有观察者的抽象方法。观察者数量可以任意多。

抽象观察者：一个包含更新方法的抽象类或接口，在接收到主题更改通知时调用。

结构


Subject
observers
Attach(Observer)
Detach(Observer)
Notify()
for all o in observers {
o->Update()
}
Observer
Update()
ConcreteObserver
Update()
observerState
observerState =
subject->GetState()
subject
ConcreteSubject
GetState()
SetState()
subjectState
return subjectState

协作

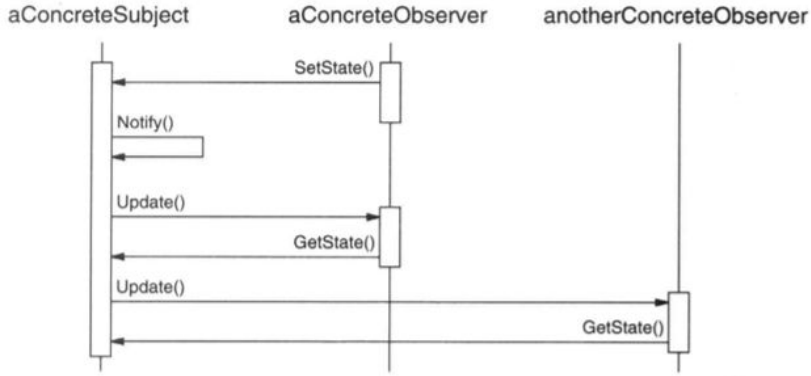
目标改变、通知

当具体目标对象发生任何可能导致其观察者与本身状态不一致的变化时，观察者将被通知

观察者收到、更新

在得到一个具体目标的改变通知后,具体观察者对象可向目标对象查询信息。具体观察者对象使用这些信息以使它的状态与目标对象的状态一致。

协作


aConcreteSubject
aConcreteObserver
anotherConcreteObserver
SetState()
Notify()
Update()
GetState()
Update()
GetState()

实际使用场景

E3

微信公众号

只有订阅的用户才会收到对应公众号的内容更新推送

E2

线上拍卖竞价

最高出价的更新会被每一个竞价者收到。

E1

气象数据更新

手机app中的天气数据会随着相应的气象站数据更新而更新。

实现——解决更多问题

观察多个对象

需要扩展update接口，观察者得知道通知是哪一个目标对象发来的。

更新触发

1.每次subject状态改变后自动调用通知。

2.用户来选择何时通知。

开销和准确性的权衡。

维护映射关系

可以显式保存引用关系，也可以创建关联查找机制用时间换空间。

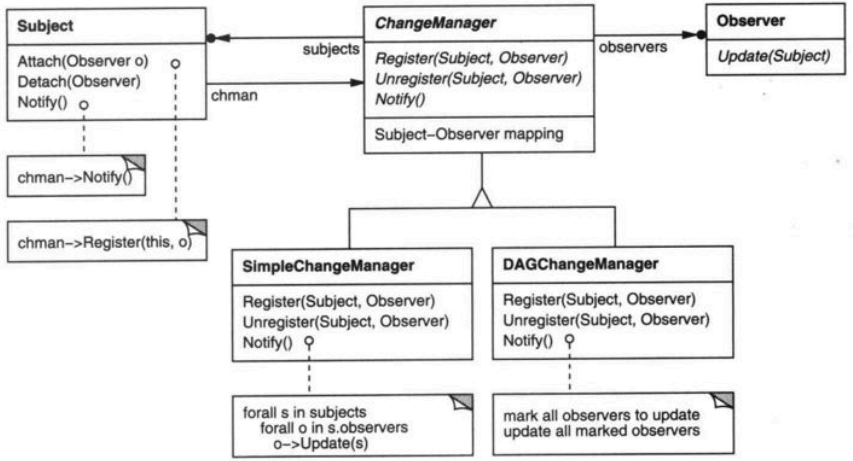
不同粒度的观察

推模型与拉模型，分别适应不同需求。

封装复杂的更新语义

ChangeManager作为观察者和目标之间的桥梁。

实现——
解决更多问题


Subject
Attach(Observer o)
Detach(Observer)
Notify()
subjects
chman
ChangeManager
Register(Subject, Observer)
Unregister(Subject, Observer)
Notify()
Subject–Observer mapping
observers
Observer
Update(Subject)
chman–>Notify()
chman–>Register(this, o)
SimpleChangeManager
Register(Subject, Observer)
Unregister(Subject, Observer)
Notify()
DAGChangeManager
Register(Subject, Observer)
Unregister(Subject, Observer)
Notify()
forall s in subjects
forall o in s.observers
o–>Update(s)
mark all observers to update
update all marked observers

开源项目

DuckDB中的DependencyManager

DependencyManager负责管理数据库中不同条目间的依赖关系，当依赖关系的双方发生改变时，DependencyManager负责更新对应的依赖关系。
这里并没有使用严格的观察者模式实现，从代码中可以看到。对依赖关系的更新是遍历所有观察者并依次处理依赖关系，而不是由观察者本身进行update，也并没有明显的通知机制的实现。

```
// Keep old dependencies
dependency_set_t dependents;
ScanSubjects(transaction, old_info, [&](DependencyEntry &dep) {
    auto entry = LookupEntry(transaction, dep);
    if (!entry) {
        return;
    }

    auto dep_info = DependencyInfo::FromSubject(dep);
    dep_info.dependent.entry = new_info;
    dependencies.emplace_back(dep_info);
});

// FIXME: we should update dependencies in the future
// some alters could cause dependencies to change (imagine types of table columns)
// or DEFAULT depending on a sequence
if (!StringUtil::CIEquals(old_obj.name, new_obj.name)) {
    // The name has been changed, we need to recreate the dependency links
    CleanupDependencies(transaction, old_obj);
}

// Reinstate the old dependencies
for (auto &dep : dependencies) {
    CreateDependency(transaction, dep);
}
```

开源项目

- **DuckDB中的DependencyManager**

原因可能有两点:

一是数据库(特别是关系复杂、数据量大的数据库)中，依赖关系可能需要非常频繁的更新，采用观察者模式可能会带来效率上的降低。

二是按照观察者模式的思路，此处条目间的**依赖关系**才是真正的观察者，因此这样可能会导致观察者过多，目标对象也过多

这里其实是非常符合观察者模式的需求的，将具有依赖关系的条目作为目标，它们之间的依赖关系作为观察者，由DependencyManager作为中间环节决定如何通知。
但实际需要综合性能进行权衡，这也提醒我不能盲目套用设计模式，实际开发需要做好tradeoff

感谢！

姚永舟

2024.11.29