4种设计模式

1. 工厂模式：将对象的创建和使用相分离，以达到降低系统耦合度、提高可维护性的目的。例如，在人才招聘部门的简历筛选流程中，通过工厂模式来解耦简历信息的读取和处理逻辑，可以大大减少代码的重复，提高代码整体的可读性和可维护性。

优点：

- 可以封装复杂的对象创建过程，降低系统间的耦合度。
- 根据需要提供不同类型或配置的对象实例，方便扩展和维护。

缺点：

- 需要增加代码量，引入新的类和接口。
- 可能导致多层嵌套的工厂类结构，增加了代码的复杂性和维护成本。

2. 观察者模式：定义一种一对多的依赖关系，当一个对象状态改变时，所有依赖于它的对象都会得到通知并自动更新。例如，在人事考核模块中，当员工的考核结果发生变化时，通过观察者模式，可以使相关人员（如部门经理）能够及时了解到该员工的考核情况，并根据考核情况实时调整相关的权责范围和薪资水平等。

优点：

- 可以建立一种松散耦合的关系，减少代码之间的直接交互。
- 当被观察对象发生变化时，可以及时通知观察者做出相应反应。

缺点：

- 如果被观察对象频繁发生变化，会带来较大的系统开销。
  - 观察者的过多或不合理调度可能引起性能问题和资源浪费。

3. 单例模式：保证一个类只有一个实例存在，且易于全局访问。例如，在人事薪酬管理模块中，为了避免出现多份数据导致计算错误的情况，使用单例模式确保只有一份薪资表存在，并保证在全局范围内都能够方便快捷地使用该表格中的信息。

优点：

- 可以有效地控制系统内存资源，避免重复对象的创建。
  - 保证在全局范围内只存在一个对象实例，方便数据的共享和交换。

缺点：

- 违反单一职责原则，将对象的创建和管理放在了一处。
  - 可能引发线程安全性相关问题，需要特别注意多线程方面的实现。

4. 策略模式：定义一系列算法，将每个算法都封装起来，并且使它们之间可以相互替换。例如，在人才培养部门的晋升流程设计中，通过策略模式将晋升条件定义为一系列可替换的策略对象，随着企业对员工要求不断变化，只需调整晋升策略对象的行为即可符合新的要求。

优点：

- 可以有效地封装一系列的算法策略，并动态地选择合适的策略对象。
  - 通过多态性和接口隔离原则，减少了代码的冗余和复杂度。

缺点：

- 需要创建大量的策略类，会增加类的数量并提高系统的耦合度。
- 对应不同的策略类，需要仔细选择合适的命名方式和配置参数。