# 定义

2002年罗伯特·C.马丁为接口隔离原则（Interface Segregation Principle，ISP）给出的定义是：客户端不应该被迫依赖于它不使用的方法（Clients should not be forced to depend on methods they do not use）。一个类对另一个类的依赖应该建立在最小的接口上（The dependency of one class to another one should depend on the smallest possible interface）。

# 含义

接口隔离原则的含义是：要为各个类建立仅包含所需功能的专用接口，不要试图去建立一个包含所有功能的庞大的接口供所有类使用。

接口隔离原则认为：使用多个专门的接口比使用单一的总接口要好。

# 优点

接口隔离原则是为了约束接口、降低类对接口的依赖性，遵循接口隔离原则有以下优点：

|  |  |
| --- | --- |
| 1 | 将臃肿庞大的接口分解为多个细粒度的接口，可以预防变更的扩散，提高系统的稳定性、灵活性和可维护性。 |
| 2 | 提高系统的内聚性，降低系统的耦合性。 |
| 3 | 使用多个专门的接口能够体现对象的层次，提高代码的可读性。 |
| 4 | 定制的接口中仅包含需要的功能，使用定制接口能够减少冗余代码。 |

# 注意

接口的粒度大小定义合理，确实能够保证系统的稳定性，但是如果定义过小，则会导致接口的数量过多，系统复杂程度变高，系统维护难度提升，系统可扩展性降低。

# 使用场景

分析：学生成绩管理模块一般包含成绩录入、成绩删除、成绩修改、成绩查询等功能，模块的服务对象为教师和学生，学生只能使用成绩插叙功能。如果将这些功能全部放到一个接口中显然不太合理，正确的做法是将它们分散到成绩编辑接口和成绩查询接口中，其类图如下：



代码：

*/\*\*  
 \** ***@author*** *xzy  
 \** ***@date*** *2020-01-04 17:19  
 \* 说明：成绩编辑  
 \*/*public interface GradeEditor *{* void insert*()*;  
 void delete*()*;  
 void update*()*;  
*}*

*/\*\*  
 \** ***@author*** *xzy  
 \** ***@date*** *2020-01-04 17:20  
 \* 说明：成绩查询  
 \*/*public interface GradeQuery *{* void query*()*;  
*}*

*/\*\*  
 \** ***@author*** *xzy  
 \** ***@date*** *2020-01-04 17:22  
 \* 说明：成绩管理  
 \*/*public class GradeManage implements GradeEditor, GradeQuery *{* private GradeManage*() {  
 }* public static GradeEditor getGradeEditor*() {* return *(*GradeEditor*)* new GradeManage*()*;  
 *}* public static GradeQuery getGradeQuery*() {* return *(*GradeQuery*)* new GradeManage*()*;  
 *}* @Override  
 public void insert*() {}* @Override  
 public void delete*() {}* @Override  
 public void update*() {}* @Override  
 public void query*() {}  
}*

# 其他

## 接口隔离原则与单一职责原则

假设有“学生成绩管理模块”，模块包含学生成绩录入、学生成绩修改、学生成绩删除、学科最高分查询、学科平均分查询、学生成绩查询、及格人数查询、不及格人数查询等功能，每个功能都用一个方法实现，且所有方法都放置在一个接口中:



从单一职责原则的角度看这是合理的，因为接口承担的职责确实是唯一的——学生成绩管理。但是从接口隔离原则的角度看这是不合理的，因为所有的方法都被放置在一个接口中，接口显得过于臃肿。

假设学生使用本模块时只有查询成绩的权限，Student类只需要使用“学生成绩查询()”方法，但是按照目前的情况，Student类将被迫获取其他几个方法的使用权限，这是违背接口隔离原则的。