物联网学院

高频问题集

[一、 面向对象是什么？](#_Toc9637)

[1. 通常解法](#_Toc9305)

[2. 通用大牛级解法](#_Toc25927)

[3. 解法对比及优缺点](#_Toc2314)

[4. 延伸及扩展问题回答参考](#_Toc20746)

[5. 项目中体现经验的点](#_Toc27233)

[6. 论坛参考](#_Toc26829)

[二、 MVC是什么](#_Toc29413)

[1. 通常解法](#_Toc22137)

[2. 通用大牛级解法](#_Toc12375)

[3. 解法对比及优缺点](#_Toc13919)

[4. 延伸及扩展问题回答参考](#_Toc17677)

[5. 项目中体现经验的点](#_Toc25504)

[6. 论坛参考](#_Toc24899)

[三、什么是事务](#_Toc28859)

[1. 通常解法](#_Toc11231)

[2. 通用大牛级解法](#_Toc30138)

[3. 解法对比及优缺点](#_Toc16002)

[4. 延伸及扩展问题回答参考](#_Toc26770)

[5. 项目中体现经验的点](#_Toc14034)

[6. 论坛参考](#_Toc6579)

[四、什么是索引？](#_Toc12130)

[1. 通常解法](#_Toc32213)

[2. 通用大牛级解法](#_Toc17579)

[3. 解法对比及优缺点](#_Toc30604)

[4. 延伸及扩展问题回答参考](#_Toc12012)

[5. 项目中体现经验的点](#_Toc9775)

[6. 论坛参考](#_Toc32752)

[五、String是什么类型，大批量拼接要注意什么](#_Toc11522)

[1. 通常解法](#_Toc19457)

[2. 通用大牛级解法](#_Toc15571)

[3. 解法对比及优缺点](#_Toc32670)

[4. 延伸及扩展问题回答参考](#_Toc15210)

[5. 项目中体现经验的点](#_Toc28585)

[6. 论坛参考](#_Toc32631)

[六、什么是IOC容器](#_Toc24606)

[1、通常解法](#_Toc3296)

[2、通用大牛级解法](#_Toc23975)

[3、解法对比及优缺点](#_Toc23712)

[4、延伸及扩展问题回答参考](#_Toc26742)

[5、项目中体现经验的点](#_Toc21664)

[6、论坛参考](#_Toc9041)

[七、什么是单例模式](#_Toc30497)

[1、通用解法](#_Toc4029)

[2、大牛解法](#_Toc20148)

[3、解法对比及优缺点](#_Toc26782)

[4、延伸及扩展问题回答参考](#_Toc734)

[5、项目中体现经验的点](#_Toc30417)

[6、论坛参考](#_Toc8912)

[八、事件与委托的理解以及有什么区别](#_Toc22401)

[1、 通常解法](#_Toc24278)

[2、通用大牛级解法](#_Toc10828)

[(1),观察者](#_Toc28282)

[(2).被观察](#_Toc3445)

[(3).撤销观察](#_Toc4193)

[(4).委托](#_Toc16563)

[(5).事件](#_Toc14957)

[(6).委托与事件的区别](#_Toc19289)

[3、解法对比及优缺点](#_Toc949)

[4、延伸及扩展问题回答参考](#_Toc10481)

[(1).什么时候选择使用事件委托](#_Toc24384)

[(2).提高事件委托性能的解决方案](#_Toc22399)

[5、项目中体现经验的点](#_Toc13743)

[6、论坛参考](#_Toc15620)

[九、简述WebApi与WebService、WCF三种技术，并说明他们的区别](#_Toc7491)

[1、 通常解法](#_Toc17920)

[2、 大牛级解法](#_Toc10961)

[3、解法对比及优缺点](#_Toc27895)

[3、 延伸及扩展问题回答参考](#_Toc27562)

[5、项目中体现经验的点](#_Toc6994)

[6、论坛参考](#_Toc22407)

[十．什么是NoSql](#_Toc5648)

[1、通用解法](#_Toc17133)

[2、大牛解法](#_Toc32028)

[3、解法对比及优缺点](#_Toc23982)

[4、延伸及扩展问题回答参考](#_Toc23114)

[5、项目中体现经验的点](#_Toc26222)

[6、论坛参考](#_Toc25928)

# 面向对象是什么？

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 面试概率 | 级别 | 应用模块或方向 | 解决问题 | 考核点 |
| 90% | 4星 | .Net | 面向对象是什么？ | C# |

## 通常解法

相同点：

面向对象是一种思想，是基于面向过程而言的，就是说面向对象是将功能等通过对象来实现，将功能封装进对象之中，让对象去实现具体的细节；这种思想是将数据作为第一位，而方法或者说是算法作为其次，这是对数据一种优化，操作起来更加的方便，简化了过程。

不同点：

面向对象有三大特征：封装性、继承性、多态性，其中封装性指的是隐藏了对象的属性和实现细节，仅对外提供公共的访问方式，这样就隔离了具体的变化，便于使用，提高了复用性和安全性。对于继承性，就是两种事物间存在着一定的所属关系，那么继承的类就可以从被继承的类中获得一些属性和方法；这就提高了代码的复用性。继承是作为多态的前提的。多态是说父类或接口的引用指向了子类对象，这就提高了程序的扩展性，也就是说只要实现或继承了同一个接口或类，那么就可以使用父类中相应的方法，提高程序扩展性，但是多态有一点不好之处在于：父类引用不能访问子类中的成员。

## 通用大牛级解法

相同点：

面向对象是一种思想

面向对象有三大特征：封装性、继承性、多态性

不同点：

将复杂的事情简单化。

面向对象将以前的过程中的执行者，变成了指挥者。

面向对象这种思想是符合现在人们思考习惯的一种思想。

面向对象的三大特征：封装，继承、多态

## 解法对比及优缺点

常见解法，简要说明面向对象的解释。

通用解法，详细说明面向对象的三大特征。

## 延伸及扩展问题回答参考

问题：阐述面向对象与面向接口的关系

解答

面向接口编程和面向对象编程并不是平级的，它并不是比面向对象编程更先进的一种独立的编程思想，而是附属于面向对象思想体系，属于其一部分。或者说，它是面向对象编程体系中的思想精髓之一。

面向接口编程：每个层级并不直接向上级提供服务，而是通过定义一组接口，向上层暴露接口，而上层仅仅依赖于下层所暴露的接口而不是依赖具体的类

## 项目中体现经验的点

解答

封装

只隐藏对象的属性和实现细节，仅对外提供公共访问方式

好处：将变化隔离、便于使用、提高复用性、提高安全性

原则：将不需要对外提供的内容隐藏起来；把属性隐藏，提供公共方法对其访问

继承

提高代码复用性；继承是多态的前提

子类中所有的构造函数都会默认访问父类中的空参数的构造函数。若无空参数构造函数，子类中需指定；另外，子类构造函数中可自己用this指定自身的其他构造函数。

多态

是父类或接口定义的引用变量可以指向子类或具体实现类的实例对象

好处：提高了程序的扩展性

弊端：当父类引用指向子类对象时，虽提高了扩展性，但只能访问父类中具备的方法，不可访问子类中的方法；即访问的局限性。

前提：实现或继承关系；覆写父类方法。

## 论坛参考

[https://www.cnblogs.com/zuowj/p/4887242.html](http://www.cnblogs.com/ronli/archive/2011/10/26/2224654.html)

# MVC是什么、

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 面试概率 | 级别 | 应用模块或方向 | 解决问题 | 考核点 |
| 96% | 5星 | .Net | MVC是什么？ | C#技能点 |

## 通常解法

MVC全名是Model View Controller，是模型(model)－视图(view)－控制器(controller)的缩写，一种软件设计典范，用一种业务逻辑、数据、界面显示分离的方法组织代码，将业务逻辑聚集到一个部件里面，在改进和个性化定制界面及用户交互的同时，不需要重新编写业务逻辑。MVC被独特的发展起来用于映射传统的输入、处理和输出功能在一个逻辑的图形化用户界面的结构中。

 MVC就是类似三层的一种[架构](http://lib.csdn.net/base/architecture)，主要还是采用封装（分层）的思想，来降低耦合度，从而使我们的系统更加的灵活，扩展性更好。

## 通用大牛级解法

MVC,即模型、视图与控制器的分离，使得一个模型可以具有多个显示视图。如果用户通过某个视图的控制器改变了模型的数据，所有其它依赖于这些数据的视图都应反映到这些变化。因此，无论何时发生了何种数据变化，控制器都会将变化通知所有的视图，导致显示的更新。

内容

Model（模型）是应用程序中用于处理应用程序数据逻辑的部分。通常模型对象负责在[数据库](http://lib.csdn.net/base/mysql)中存取数据。  
View（视图）是应用程序中处理数据显示的部分。通常视图是依据模型数据创建的。  
Controller（控制器）是应用程序中处理用户交互的部分。通常控制器负责从视图读取数据，控制用户输入，并向模型发送数据

优点

（1）首先，最重要的一点是多个视图能共享一个模型。同一个模型可以被不同的视图重用，大大提高了代码的可重用性。  
（2）由于MVC的三个模块相互独立，改变其中一个不会影响其他两个，所以依据这种设计思想能构造良好的松耦合的构件。  
3）此外，控制器提高了应用程序的灵活性和可配置性。控制器可以用来联接不同的模型和视图去完成用户的需求，这样控制器可以为构造应用程序提供强有力的手段。

缺点

（1）增加了系统结构和实现的复杂性。

对于简单的界面，严格遵循MVC，使模型、视图与控制器分离，会增加结构的复杂性，并可能产生过多的更新操作，降低运行效率。  
（2）视图与控制器间的过于紧密的连接。

  视图与控制器是相互分离，但确实联系紧密的部件，视图没有控制器的存在，其应用是很有限的，反之亦然，这样就妨碍了他们的独立重用。  
（3）视图对模型数据的低效率访问。

  依据模型操作接口的不同，视图可能需要多次调用才能获得足够的显示数据。对未变化数据的不必要的频繁访问，也将损害操作性能。

（4） 目前，一般高级的界面工具或构造器不支持MVC架构。

  改造这些工具以适应MVC需要和建立分离的部件的代价是很高的，从而造成使用MVC的困难

## 解法对比及优缺点

一般解法：说明MVC的基本概念

通用解法：通过详细分析MVC的优缺点，达到深入理解MVC

## 延伸及扩展问题回答参考

* MVC与WebForm的区别
* MVC
* 通过model、view和controller有效的简化了复杂的架构，体现了很好的隔离原则。
* 一切皆可测试。
* 一切皆可扩展：ViewEngine、HtmlHelper还有Filter。
* 适用于大型架构开发。
* 强类型View实现，更安全、更可靠、更高效。
* 开源，意味着更好的控制和理解。
* 没有View State，没有Server Control，这显然是个好处。
* WebForm
* 支持事件驱动模型开发，拖拽即可形成应用，简单易懂。
* 可以应用MVP模型，为每个page添加功能。
* 应用ViewState和Server Control，显然也是优点。
* 适用于快速开发。

## 项目中体现经验的点

目前.Net类的Web项目大部分都用ASP.NET MVC进行开发。

## 论坛参考

<https://www.cnblogs.com/shidan/p/6956690.html>

# 三、什么是事务？

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 面试概率 | 级别 | 应用模块或方向 | 解决问题 | 考核点 |
| 95% | 4星 | .Net | 什么是事务？ | C#技能点 |

## 通常解法

事务是作为单一工作单元而执行的一系列操作。包括增删查改。

事务分为显示事务和隐式事务：

隐式事务：就是平常使用每一条sql 语句就是一个事务，只不过他们执行完成之后事务就跟着结束。

显示事务：需要来手写，可以控制事务的开始和结束

## 通用大牛级解法

**事务**

在数据库中有时候需要把多个步骤的指令当作一个整体来运行，这个整体要么全部成功，要么全部失败，这就需要用到事务。

**事务的特点**

事务有若干条T-SQL指令组成，并且所有的指令昨晚一个整体提交给数据库系统，执行时，这组指令要么全部执行完成，要么全部取消。因此，事务是一个不可分割的逻辑单元。

 事务有4个属性：原子性（Atomicity）、一致性（Consistency）、隔离性（Isolation）以及持久性（Durability），也称作事务的ACID属性。

 原子性：事务内的所有工作要么全部完成，要么全部不完成，不存在只有一部分完成的情况。

 一致性：事务内的然后操作都不能违反数据库的然后约束或规则，事务完成时有内部数据结构都必须是正确的。

隔离性：事务直接是相互隔离的，如果有两个事务对同一个数据库进行操作，比如读取表数据。任何一个事务看到的所有内容要么是其他事务完成之前的状态，要么是其他事务完成之后的状态。一个事务不可能遇到另一个事务的中间状态。

持久性：事务完成之后，它对数据库系统的影响是持久的，即使是系统错误，重新启动系统后，该事务的结果依然存在。

**事务的模式**

 a、 显示事务

显示事务就是用户使用T-SQL明确的定义事务的开始（begin transaction）和提交（commit transaction）或回滚事务（rollback transaction）

b、 自动提交事务

自动提交事务是一种能够自动执行并能自动回滚事务，这种方式是T-SQL的默认事务方式。例如在删除一个表记录的时候，如果这条记录有主外键关系的时候，删除就会受主外键约束的影响，那么这个删除就会取消。

可以设置事务进入隐式方式：set implicit\_transaction on;

c、 隐式事务

隐式事务是指当事务提交或回滚后，SQL Server自动开始事务。因此，隐式事务不需要使用begin transaction显示开始，只需直接失业提交事务或回滚事务的T-SQL语句即可。

使用时，需要设置set implicit\_transaction on语句，将隐式事务模式打开，下一个语句会启动一个新的事物，再下一个语句又将启动一个新事务。

**事务处理**

常用T-SQL事务语句：

a、begin transaction语句

开始事务，而@@trancount全局变量用来记录事务的数目值加1，可以用@@error全局变量记录执行过程中的错误信息，如果没有错误可以直接提交事务，有错误可以回滚。

b、commit transaction语句

回滚事务，表示一个隐式或显示的事务的结束，对数据库所做的修改正式生效。并将@@trancount的值减1；

c、rollback transaction语句

回滚事务，执行rollback tran语句后，数据会回滚到begin tran的时候的状态

**事务与锁**

在多用户都用事务同时访问同一个数据资源的情况下，就会造成以下几种数据错误。

* 更新丢失：多个用户同时对一个数据资源进行更新，必定会产生被覆盖的数据，造成数据读写异常。
* 不可重复读：如果一个用户在一个事务中多次读取一条数据，而另外一个用户则同时更新啦这条数据，造成第一个用户多次读取数据不一致。
* 脏读：第一个事务读取第二个事务正在更新的数据表，如果第二个事务还没有更新完成，那么第一个事务读取的数据将是一半为更新过的，一半还没更新过的数据，这样的数据毫无意义。
* 幻读：第一个事务读取一个结果集后，第二个事务，对这个结果集经行增删操作，然而第一个事务中再次对这个结果集进行查询时，数据发现丢失或新增。
* 然而锁定，就是为解决这些问题所生的，他的存在使得一个事务对他自己的数据块进行操作的时候，而另外一个事务则不能插足这些数据块。这就是所谓的锁定。

锁定从数据库系统的角度大致可以分为6种：

* 共享锁（S）：还可以叫他读锁。可以并发读取数据，但不能修改数据。也就是说当数据资源上存在共享锁的时候，所有的事务都不能对这个资源进行修改，直到数据读取完成，共享锁释放。
* 排它锁（X）：还可以叫他独占锁、写锁。就是如果你对数据资源进行增删改操作时，不允许其它任何事务操作这块资源，直到排它锁被释放，防止同时对同一资源进行多重操作。
* 更新锁（U）：防止出现死锁的锁模式，两个事务对一个数据资源进行先读取在修改的情况下，使用共享锁和排它锁有时会出现死锁现象，而使用更新锁则可以避免死锁的出现。资源的更新锁一次只能分配给一个事务，如果需要对资源进行修改，更新锁会变成排他锁，否则变为共享锁。
* 意向锁：SQL Server需要在层次结构中的底层资源上（如行，列）获取共享锁，排它锁，更新锁。例如表级放置了意向共享锁，就表示事务要对表的页或行上使用共享锁。在表的某一行上上放置意向锁，可以防止其它事务获取其它不兼容的的锁。意向锁可以提高性能，因为数据引擎不需要检测资源的每一列每一行，就能判断是否可以获取到该资源的兼容锁。意向锁包括三种类型：意向共享锁（IS），意向排他锁（IX），意向排他共享锁（SIX）。
* 架构锁：防止修改表结构时，并发访问的锁。
* 大容量更新锁：允许多个线程将大容量数据并发的插入到同一个表中，在加载的同时，不允许其它进程访问该表。

**事务隔离级别**

就是并发事务对同一资源的读取深度层次。分为5种。

read uncommitted：这个隔离级别最低啦，可以读取到一个事务正在处理的数据，但事务还未提交，这种级别的读取叫做脏读。

read committed：这个级别是默认选项，不能脏读，不能读取事务正在处理没有提交的数据，但能修改。

repeatable read：不能读取事务正在处理的数据，也不能修改事务处理数据前的数据。

snapshot：指定事务在开始的时候，就获得了已经提交数据的快照，因此当前事务只能看到事务开始之前对数据所做的修改。

serializable：最高事务隔离级别，只能看到事务处理之前的数据。

## 解法对比及优缺点

一般解法：简单说明事务的定义及特点

通用解法：对于事务的定义、分类、锁、隔离级别、用法进行详细说明。

## 延伸及扩展问题回答参考

事务的好处

解答：

保证数据一致性、完整性。减少系统产生的垃圾数据。

如何减少死锁

死锁耗时耗资源，在大型数据库中，高并发带来的死锁是不可避免的，所以只能让其变的更少。

* 按照同一顺序访问数据库资源，上述例子就不会发生死锁啦
* 保持是事务的简短，尽量不要让一个事务处理过于复杂的读写操作。事务过于复杂，占用资源会增多，处理时间增长，容易与其它事务冲突，提升死锁概率。
* 尽量不要在事务中要求用户响应，比如修改新增数据之后在完成整个事务的提交，这样延长事务占用资源的时间，也会提升死锁概率。
* 尽量减少数据库的并发量。
* 尽可能使用分区表，分区视图，把数据放置在不同的磁盘和文件组中，分散访问保存在不同分区的数据，减少因为表中放置锁而造成的其它事务长时间等待。
* 避免占用时间很长并且关系表复杂的数据操作。
* 使用较低的隔离级别，使用较低的隔离级别比使用较高的隔离级别持有共享锁的时间更短。这样就减少了锁争用

## 项目中体现经验的点

方法的重写和重载一样作为多态的体现伴随整个项目，通过对类的重写，方法的重写，增加灵活性

## 论坛参考

https://www.cnblogs.com/knowledgesea/p/3714417.html

# 四、什么是索引？

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 面试概率 | 级别 | 应用模块或方向 | 解决问题 | 考核点 |
| 70% | 4星 | .Net | 什么是索引？ | SQL |

## 通常解法

SQL索引有两种，聚集索引和非聚集索引，索引主要目的是提高了SQL Server系统的性能，加快数据的查询速度与减少系统的响应时间。

## 通用大牛级解法

**什么是索引**

SQL索引有两种，聚集索引和非聚集索引，索引主要目的是提高了SQL Server系统的性能，加快数据的查询速度与减少系统的响应时间

下面举两个简单的例子：

图书馆的例子：一个图书馆那么多书，怎么管理呢？建立一个字母开头的目录，例如：a开头的书，在第一排，b开头的在第二排，这样在找什么书就好说了，这个就是一个聚集索引，可是很多人借书找某某作者的，不知道书名怎么办？图书管理员在写一个目录，某某作者的书分别在第几排，第几排，这就是一个非聚集索引

字典的例子：字典前面的目录，可以按照拼音和部首去查询，想查询一个字，只需要根据拼音或者部首去查询，就可以快速的定位到这个汉字了，这个就是索引的好处，拼音查询法就是聚集索引，部首查询就是一个非聚集索引.

聚集索引存储记录是物理上连续存在，而非聚集索引是逻辑上的连续，物理存储并不连续。就像字段，聚集索引是连续的，a后面肯定是b，非聚集索引就不连续了，就像图书馆的某个作者的书，有可能在第1个货架上和第10个货架上。还有一个小知识点就是：聚集索引一个表只能有一个，而非聚集索引一个表可以存在多个。

**索引的存储机制**

首先，无索引的表，查询时，是按照顺序存续的方法扫描每个记录来查找符合条件的记录，这样效率十分低下,举个例子，如果将字典的汉字随即打乱，没有前面的按照拼音或者部首查询，那么想找一个字，按照顺序的方式去一页页的找，这样效率有多底，大家可以想象。

 聚集索引和非聚集索引的根本区别是表记录的排列顺序和与索引的排列顺序是否一致，其实理解起来非常简单，还是举字典的例子：如果按照拼音查询，那么都是从a-z的，是具有连续性的，a后面就是b，b后面就是c， 聚集索引就是这样的，他是和表的物理排列顺序是一样的，例如有id为聚集索引，那么1后面肯定是2,2后面肯定是3，所以说这样的搜索顺序的就是聚集索引。非聚集索引就和按照部首查询是一样是，可能按照偏房查询的时候，根据偏旁‘弓’字旁，索引出两个汉字，张和弘，但是这两个其实一个在100页，一个在1000页，（这里只是举个例子），他们的索引顺序和数据库表的排列顺序是不一样的，这个样的就是非聚集索引。

**建立索引的原则：**

1) 定义主键的数据列一定要建立索引。

2) 定义有外键的数据列一定要建立索引。

3) 对于经常查询的数据列最好建立索引。

4) 对于需要在指定范围内的快速或频繁查询的数据列;

5) 经常用在WHERE子句中的数据列。

6) 经常出现在关键字order by、group by、distinct后面的字段，建立索引。如果建立的是复合索引，索引的字段顺序要和这些关键字后面的字段顺序一致，否则索引不会被使用。

7) 对于那些查询中很少涉及的列，重复值比较多的列不要建立索引。

8) 对于定义为text、image和bit的数据类型的列不要建立索引。

9) 对于经常存取的列避免建立索引

9) 限制表上的索引数目。对一个存在大量更新操作的表，所建索引的数目一般不要超过3个，最多不要超过5个。索引虽说提高了访问速度，但太多索引会影响数据的更新操作。

10) 对复合索引，按照字段在查询条件中出现的频度建立索引。在复合索引中，记录首先按照第一个字段排序。对于在第一个字段上取值相同的记录，系统再按照第二个字段的取值排序，以此类推。因此只有复合索引的第一个字段出现在查询条件中，该索引才可能被使用,因此将应用频度高的字段，放置在复合索引的前面，会使系统最大可能地使用此索引，发挥索引的作用。

**如何创建索引**

创建索引的语法：

CREATE [UNIQUE][CLUSTERED | NONCLUSTERED]  INDEX  index\_name

ON {table\_name | view\_name} [WITH [index\_property [,....n]]

说明：

UNIQUE: 建立唯一索引。

CLUSTERED: 建立聚集索引。

NONCLUSTERED: 建立非聚集索引。

Index\_property: 索引属性。

 UNIQUE索引既可以采用聚集索引结构，也可以采用非聚集索引的结构，如果不指明采用的索引结构，则SQL Server系统默认为采用非聚集索引结构。

删除索引语法：

DROP INDEX table\_name.index\_name[,table\_name.index\_name]

说明：table\_name: 索引所在的表名称。

index\_name : 要删除的索引名称。

显示索引信息：

使用系统存储过程：sp\_helpindex 查看指定表的索引信息。

执行代码如下：

Exec sp\_helpindex book1;

**索引使用次数、索引效率、占用CPU检测、索引缺失**

首先认识一下DMV，DMV (dynamic management view)动态管理视图和函数返回特定于实现的内部状态数据。推出SQL Server 2005时，微软介绍了许多被称为dmvs的系统视图，让您可以探测SQL Server 的健康状况，诊断问题，或查看SQL Server实例的运行信息。统计数据是在SQL Server运行的时候开始收集的，并且在SQL Server每次启动的时候，统计数据将会被重置。当你删除或者重新创建其组件时，某些dmv的统计数据也可以被重置，例如存储过程和表，而其它的dmv信息在运行dbcc命令时也可以被重置。

当使用一个dmv时，需要紧记SQL Server收集这些信息有多长时间了，以确定这些从dmv返回的数据到底有多少可用性。如果SQL Server只运行了很短的一段时间，可能不想去使用一些dmv统计数据，因为他们并不是一个能够代表SQL Server实例可能遇到的真实工作负载的样本。另一方面，SQL Server只能维持一定量的信息，有些信息在进行SQL Server性能管理活动的时候可能丢失，所以如果SQL Server已经运行了相当长的一段时间，一些统计数据就有可能已被覆盖。

## 解法对比及优缺点

一般解法：简单说明索引的定义

通用解法：详细说明索引的语法、分类及使用索引时的注意事项。

## 延伸及扩展问题回答参考

**问题：为什么要创建索引（优点）**  
创建索引可以大大提高系统的性能。  
第一，   通过创建唯一性索引，可以保证[数据库](http://lib.csdn.net/base/mysql)表中每一行数据的唯一性。  
第二，   可以大大加快数据的检索速度，这也是创建索引的最主要的原因。  
第三，   可以加速表和表之间的连接，特别是在实现数据的参考完整性方面特别有意义。  
第四，   在使用分组和排序子句进行数据检索时，同样可以显著减少查询中分组和排序的时间。  
第五，   通过使用索引，可以在查询的过程中，使用优化隐藏器，提高系统的性能。   
**问题：建立方向索引的不利因素（缺点）**  
也许会有人要问：增加索引有如此多的优点，为什么不对表中的每一个列创建一个索引呢？这种想法固然有其合理性，然而也有其片面性。虽然，索引有许多优点，但是，为表中的每一个列都增加索引，是非常不明智的。这是因为，增加索引也有许多不利的一个方面。   
第一，   创建索引和维护索引要耗费时间，这种时间随着数据量的增加而增加。  
第二，   索引需要占物理空间，除了数据表占数据空间之外，每一个索引还要占一定的物理空间，如果要建立聚簇索引，那么需要的空间就会更大。  
第三，   当对表中的数据进行增加、删除和修改的时候，索引也要动态的维护，这样就降低了数据的维护速度。   
**问题：哪些列上需要创建索引。**  
第一，   在经常需要搜索的列上，可以加快搜索的速度；  
第二，   在作为主键的列上，强制该列的唯一性和组织表中数据的排列结构；  
第三，   在经常用在连接的列上，这些列主要是一些外键，可以加快连接的速度；  
第四，   在经常需要根据范围进行搜索的列上创建索引，因为索引已经排序，其指定的范围是连续的；  
第五，   在经常需要排序的列上创建索引，因为索引已经排序，这样查询可以利用索引的排序，加快排序查询时间；  
第六，   在经常使用在WHERE子句中的列上面创建索引，加快条件的判断速度。   
**问题：有些列不应该创建索引**  
第一、对于那些在查询中很少使用或者参考的列不应该创建索引。这是因为，既然这些列很少使用到，因此有索引或者无索引，并不能提高查询速度。相反，由于增加了索引，反而降低了系统的维护速度和增大了空间需求。  
第二、对于那些只有很少数据值的列也不应该增加索引。这是因为，由于这些列的取值很少，例如人事表的性别列，在查询的结果中，结果集的数据行占了表中数据行的很大比例，即需要在表中搜索的数据行的比例很大。增加索引，并不能明显加快检索速度。  
第三、对于那些定义为text, image和bit数据类型的列不应该增加索引。这是因为，这些列的数据量要么相当大，要么取值很少。  
第四、 当修改性能远远大于检索性能时，不应该创建索引。这是因为，修改性能和检索性能是互相矛盾的。当增加索引时，会提高检索性能，但是会降低修改性能。当减少索引时，会提高修改性能，降低检索性能。因此，当修改性能远远大于检索性能时，不应该创建索引。  
**创建索引的方法有哪些**  
创建索引有多种方法，这些方法包括直接创建索引的方法和间接创建索引的方法。  
第一、直接创建索引，例如使用CREATE INDEX语句或者使用创建索引向导。  
第二、间接创建索引，例如在表中定义主键约束或者唯一性键约束时，同时也创建了索引。  
虽然，这两种方法都可以创建索引，但是，它们创建索引的具体内容是有区别的。  
使 用CREATE INDEX语句或者使用创建索引向导来创建索引，这是最基本的索引创建方式，并且这种方法最具有柔性，可以定制创建出符合自己需要的索引。在使用这种方式创建索引时，可以使用许多选项，例如指定数据页的充满度、进行排序、整理统计信息等，这样可以优化索引。使用这种方法，可以指定索引的类型、唯一性和复合性，也就是说，既可以创建聚簇索引，也可以创建非聚簇索引，既可以在一个列上创建索引，也可以在两个或者两个以上的列上创建索引。  
通过定义主键约束或者唯一性键约束，也可以间接创建索引。主键约束是一种保持数据完整性的逻辑，它限制表中的记录有相同的主键记录。在创建主键约束时，系统自动创建了一个唯一性的聚簇索引。虽然，在逻辑上，主键约束是一种重要的结构，但是，在物理结构上，与主键约束相对应的结构是唯一性的聚簇索引。换句话说，在物理实现上，不存在主键约束，而只存在唯一性的聚簇索引。同样，在创建唯一性键约束时，也同时创建了索引，这种索引则是唯一性的非聚簇索引。因此，当使用约束创建索引时，索引的类型和特征基本上都已经确定了，由用户定制的余地比较小。  
当在表上定义主键或者唯一性键约束时，如果表中已经有了使用 CREATE INDEX语句创建的标准索引时，那么主键约束或者唯一性键约束创建的索引覆盖以前创建的标准索引。也就是说，主键约束或者唯一性键约束创建的索引的优先级高于使用CREATE INDEX语句创建的索引。  
**系统如何访问表中的数据**  
一般地，系统访问数据库中的数据，可以使用两种方法：表扫描和索引查找。第一种方法是表扫描，就是指系统将指针放置在该表的表头数据所在的数据页上，然后按照数据页的排列顺序，一页一页地从前向后扫描该表数据所占有的全部数据页，直至扫描完表中的全部记录。在扫描时，如果找到符合查询条件的记录，那么就将这条记录挑选出来。最后，将全部挑选出来符合查询语句条件的记录显示出来。第二种方法是使用索引查找。索引是一种树状结构，其中存储了关键字和指向包含关键字所在记录的数据页的指针。当使用索引查找时，系统沿着索引的树状结构，根据索引中关键字和指针，找到符合查询条件的的记录。最后，将全部查找到的符合查询语句条件的记录显示出来。  
    在SQL Server中，当访问数据库中的数据时，由SQL Server确定该表中是否有索引存在。如果没有索引，那么SQL Server使用表扫描的方法访问数据库中的数据。查询处理器根据分布的统计信息生成该查询语句的优化执行规划，以提高访问数据的效率为目标，确定是使用表扫描还是使用索引。

## 项目中体现经验的点

系统优化的时候，需要创建索引

## 论坛参考

[https://www.cnblogs.com/AK2012/archive/2013/01/04/2013-0104.html](https://www.cnblogs.com/ldyblogs/p/event.html)

# 五、String是什么类型，大批量拼接要注意什么

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 面试概率 | 级别 | 应用模块或方向 | 解决问题 | 考核点 |
| 80% | 4星 | .Net | String是什么类型？ | C# |

## 通常解法

String 对象是不可改变的。每次使用 System.String 类中的方法之一时，都要在内存中创建一个新的字符串对象，这就需要为该新对象分配新的空间。在需要对字符串执行重复修改的情况下，与创建新的 String 对象相关的系统开销可能会非常昂贵。如果要修改字符串而不创建新的对象，则可以使用 System.Text.StringBuilder 类。例如，当在一个循环中将许多字符串连接在一起时，使用 StringBuilder 类可以提升性能。

## 通用大牛级解法

**引用的命名空间**

* System.String类
* System.Text.StringBuilder类

**String**

string是用的最多的类型之一，是一个特殊的引用类型，直接派生于Object，因此它的值储存在托管堆上。构造一个新字符串的时候，不需要用new。它是”不可变的“。初始化字符串对象后，该字符串对象的长度、内容都是确定不变的。如果需要更改或者添加字符串，会做一个怎样的动作呢？假如会有这一段代码：

using System;class Program

{

static void Main(string[] args)

{

string s = "I am ";

s += "Sky";

Console.WriteLine(s);//我叫Sky }

}

1、创建一个新的字符串对象b，运行库会为新对象b分配足够的内存，储存新内容“I am Sky”。

2、变量 s重新指向新对象b。

3、旧对象a没有被任何变量引用，等待垃圾回收。

由于上面的特性，避免复制同一个字符串的多个对象，会利用CLR的字符串留用机制，减少内存的损耗。什么是字符串留用，CLR初始化的时候会创建哈希表，每构建一个新字符串都会与哈希表匹配，查找是否有相同的字符串，如果匹配，就会返回这个已存在的旧对象，由新变量进行引用。否则，就会创建一个字符串副本添加到哈希表里，Key就是字符串，Value就是string对象在堆上的地址。可见a和b的托管堆上地址是一样的。假如用new呢？new出来的字符串是不会记录在哈希表的，因此，接下来的字符串与new出来字符串对象是不一样的。

**StringBuilder**

StringBuilder和String比起来，功能较少，只有基本的属性和增删改的方法。其实，StringBuilder是由Char数组构成的。构建新的StringBuilder时能设置容量大小，当添加字符串，超过已分配的数组大小时，就会创建一个新的数组，此时容量会翻倍，旧的数据就会等待垃圾回收。

　　StringBuilder多数方法都是返回同一个StringBuilder对象，那么这个时候就可以，几个操作一起完成

**何时使用String和stringbuilder**

字符串一旦创建就不可修改大小，每次使用System.String类中的方法之一时，都要在内存中创建一个新的字符串对象，这就需要为该新对象分配新的空间。在需要对字符串执行重复修改的情况下，与创建新的String对象相关的系统开销可能会非常昂贵。如果要修改字符串而不创建新的对象，则可以使用System.Text.StringBuilder类。例如当在一个循环中将许多字符串连接在一起时，使用StringBuilder类可以提升性能。

所以对字符串添加或删除操作不频繁的话，就几个固定的string累加的时候就不需要StringBuilder了,毕竟StringBuilder的初始化也是需要时间的。对字符串添加或删除操作比较频繁的话那就用StringBuilder

## 解法对比及优缺点

一般解法：简单说明string与stringbuilder的区别

通用解法：详细介绍String与Stringbuilder的存储原理及存储效率对比。

## 延伸及扩展问题回答参考

**问题：StringBuilder和String区别**

String 在进行运算时（如赋值、拼接等）会产生一个新的实例，而 StringBuilder 则不会。所以在大量字符串拼接或频繁对某一字符串进行操作时最好使用 StringBuilder，不要使用 String。总结如下：

1.它是引用类型，在堆上分配内存

2.运算时会产生一个新的实例

3.String 对象一旦生成不可改变（Immutable）

4.定义相等运算符（== 和 !=）是为了比较 String 对象（而不是引用）的值

## 项目中体现经验的点

在项目中定义普通变量的时候，用string，如果有多个字符串进行拼接，则使用stringbuilder

## 论坛参考

[**https://www.cnblogs.com/skychen1218/p/3593678.html**](https://www.cnblogs.com/skychen1218/p/3593678.html)

[**https://www.cnblogs.com/aehyok/p/3505000.html**](https://www.cnblogs.com/aehyok/p/3505000.html)

# 六、什么是IOC容器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 面试概率 | 级别 | 应用模块或方向 | 解决问题 | 考核点 |
| 100% | 4星 | .Net | 模式 | C# |

## 1、通常解法

依赖注入是实现控制反转的一种方式

控制反转是一种设计模式，遵循了依赖倒置原则。

依赖倒置原则就是上层模块不依赖于底层模块他们都依赖于抽象

抽象不能依赖于细节，但细节依赖与抽象

依赖注入之后还会存在耦合性，通过IOC容器自动化的加载实例化类

## 2、通用大牛级解法

IoC 不是一种技术，只是一种思想，一个重要的面向对象编程的法则，它能指导我们如何设计出松耦合、更优良的程序。传统应用程序都是由我们在类内部主动创建依赖对象，从而导致类与类之间高耦合，难于测试；有了IoC容器后，把创建和查找依赖对象的控制权交给了容器，由容器进行注入组合对象，所以对象与对象之间是 松散耦合，这样也方便测试，利于功能复用，更重要的是使得程序的整个体系结构变得非常灵活

在运行期,在外部容器动态的将依赖对象注入组件,当外部容器启动后,外部容器就会初始化.创建并管理bean对象,以及销毁他.这种应用本身不负责依赖对象的创建和维护,依赖对象的创建和维护是由外部容器负责的称为控制反转.

## 3、解法对比及优缺点

依赖注入的优点:

将被依赖类的代码从依赖类中移出,不用显式的new

可以单独维护被依赖类的创建过程

方便该类被共享

依赖注入的几种方式:

属性注入

构造方法注入

接口注入(这种方法最不方便)

## 4、延伸及扩展问题回答参考

问题：IoC(控制反转)

解答

IoC（Inversion of Control，控制反转）。这是spring的核心，贯穿始终。所谓IoC，对于spring框架来说，就是由spring来负责控制对象的生命周期和对象间的关系。

问题：DI(依赖注入)

解答

IoC的一个重点是在系统运行中，动态的向某个对象提供它所需要的其他对象。这一点是通过DI（Dependency Injection，依赖注入）来实现的

## 5、项目中体现经验的点

只需要在需要的位置注入你需要的依赖，运行容器时，会自动解析依赖（包括子依赖）

## 6、论坛参考

<https://blog.csdn.net/yansuoo/article/details/51234083>

<https://blog.csdn.net/yuebinghaoyuan/article/details/7337359>

https://blog.csdn.net/u012561176/article/details/45974315

# 七、什么是单例模式

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 面试概率 | 级别 | 应用模块或方向 | 解决问题 | 考核点 |
| 70% | 3星 | .Net | 模式 | C# |

## 1、通用解法

1,在C#中最简单的一种模式

2,在C#下单例模式是一个类只能被实例化一次，有两种实现模式：懒汉模式和饿汉模式

3,懒汉模式：用到的时候进行加载，性能低。

4,饿汉模式：直接全部缓存下来，提高性能，占内存。通用大牛级解法

单例模式就是我们常用的软件设计模式，它的类只能允许一个实例存在。

有时候系统只需要一个全局对象，这样有利于我们协调系统整体的行为。

比如在某个服务器中，该服务器的配置信息存放在一个文件中，这些配置数据由一个单例对象统一读取，然后服务进程中的其他对象再通过这个单例对象获取这些配置信息。这种方式简化了在复杂环境下的配置管理。

## 2、大牛解法

1、饿汉式（静态常量）[可用]

优点：这种写法比较简单，就是在类装载的时候就完成实例化。避免了线程同步问题。

缺点：在类装载的时候就完成实例化，没有达到Lazy Loading的效果。如果从始至终从未使用过这个实例，则会造成内存的浪费。

2、饿汉式（静态代码块）[可用]

这种方式和上面的方式其实类似，只不过将类实例化的过程放在了静态代码块中，也是在类装载的时候，就执行静态代码块中的代码，初始化类的实例。优缺点和上面是一样的。

3、懒汉式(线程不安全)[不可用]

这种写法起到了Lazy Loading的效果，但是只能在单线程下使用。如果在多线程下，一个线程进入了if (singleton == null)判断语句块，还未来得及往下执行，另一个线程也通过了这个判断语句，这时便会产生多个实例。所以在多线程环境下不可使用这种方式。

4、懒汉式(线程安全，同步方法)[不推荐用]

解决上面第三种实现方式的线程不安全问题，做个线程同步就可以了，于是就对getInstance()方法进行了线程同步。

缺点：效率太低了，每个线程在想获得类的实例时候，执行getInstance()方法都要进行同步。而其实这个方法只执行一次实例化代码就够了，后面的想获得该类实例，直接return就行了。方法进行同步效率太低要改进。

5、懒汉式(线程安全，同步代码块)[不可用]

由于第四种实现方式同步效率太低，所以摒弃同步方法，改为同步产生实例化的的代码块。但是这种同步并不能起到线程同步的作用。跟第3种实现方式遇到的情形一致，假如一个线程进入了if (singleton == null)判断语句块，还未来得及往下执行，另一个线程也通过了这个判断语句，这时便会产生多个实例。

6、双重检查[推荐用]

Double-Check概念对于多线程开发者来说不会陌生，如代码中所示，我们进行了两次if (singleton == null)检查，这样就可以保证线程安全了。这样，实例化代码只用执行一次，后面再次访问时，判断if (singleton == null)，直接return实例化对象。

优点：线程安全；延迟加载；效率较高。

7、静态内部类[推荐用]

这种方式跟饿汉式方式采用的机制类似，但又有不同。两者都是采用了类装载的机制来保证初始化实例时只有一个线程。不同的地方在饿汉式方式是只要Singleton类被装载就会实例化，没有Lazy-Loading的作用，而静态内部类方式在Singleton类被装载时并不会立即实例化，而是在需要实例化时，调用getInstance方法，才会装载SingletonInstance类，从而完成Singleton的实例化。

类的静态属性只会在第一次加载类的时候初始化，所以在这里，JVM帮助我们保证了线程的安全性，在类进行初始化时，别的线程是无法进入的。

优点：避免了线程不安全，延迟加载，效率高。

8、枚举[推荐用]

借助JDK1.5中添加的枚举来实现单例模式。不仅能避免多线程同步问题，而且还能防止反序列化重新创建新的对象。可能是因为枚举在JDK1.5中才添加，所以在实际项目开发中，很少见人这么写过。

## 3、解法对比及优缺点

优点:

系统内存中该类只存在一个对象，节省了系统资源，对于一些需要频繁创建销毁的对象，使用单例模式可以提高系统性能。

缺点:

当想实例化一个单例类的时候，必须要记住使用相应的获取对象的方法，而不是使用new，可能会给其他开发人员造成困扰，特别是看不到源码的时候。

延伸及扩展问题回答参考

单例模式要求类能够有返回对象一个引用(永远是同一个)和一个获得该实例的方法（必须是静态方法，通常使用getInstance这个名称）。

单例的实现主要是通过以下两个步骤：

将该类的构造方法定义为私有方法，这样其他处的代码就无法通过调用该类的构造方法来实例化该类的对象，只有通过该类提供的静态方法来得到该类的唯一实例；

在该类内提供一个静态方法，当我们调用这个方法时，如果类持有的引用不为空就返回这个引用，如果类保持的引用为空就创建该类的实例并将实例的引用赋予该类保持的引用。

注意事项

单例模式在多线程的应用场合下必须小心使用。如果当唯一实例尚未创建时，有两个线程同时调用创建方法，那么它们同时没有检测到唯一实例的存在，从而同时各自创建了一个实例，这样就有两个实例被构造出来，从而违反了单例模式中实例唯一的原则。 解决这个问题的办法是为指示类是否已经实例化的变量提供一个互斥锁(虽然这样会降低效率)。

## 4、延伸及扩展问题回答参考

需要频繁的进行创建和销毁的对象；

创建对象时耗时过多或耗费资源过多，但又经常用到的对象；

工具类对象；

频繁访问数据库或文件的对象。

## 5、项目中体现经验的点

在加载Config文件的时候，需要保证加载的配置文件在程序中的唯一性，需要用到单例模式

## 6、论坛参考

https://www.cnblogs.com/zhaoyan001/p/6365064.html

https://blog.csdn.net/qq\_36403434/article/details/79153094

http://www.runoob.com/design-pattern/singleton-pattern.html

# 八、事件与委托的理解以及有什么区别

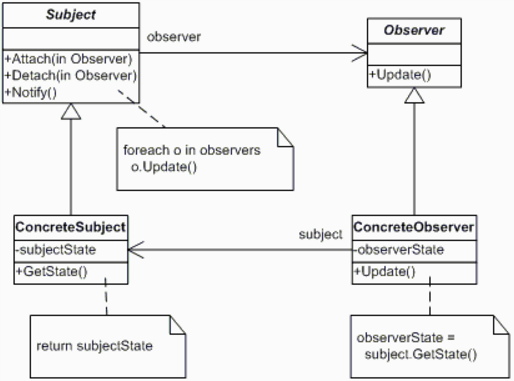
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 面试概率 | 级别 | 应用模块或方向 | 解决问题 | 考核点 |
| 70% | 4星 | .Net | C# | 事件 |

## 通常解法

委托可以将方法当作参数来进行传递，事件是一种私有的安全的委托，依赖于委托而存在。

## 2、通用大牛级解法

观察者,被观察者,撤销观察,委托,事件



### (1),观察者

（Observer）将自己注册到被观察对象（Subject）中，被观察对象将观察者存放在一个容器（Container）里。

### (2).被观察

被观察对象发生了某种变化（如图中的SomeChange），从容器中得到所有注册过的观察者，将变化通知观察者。

### (3).撤销观察

观察者告诉被观察者要撤销观察，被观察者从容器中将观察者去除。

观察者将自己注册到被观察者的容器中时，被观察者不应该过问观察者的具体类型，而是应该使用观察者的接口。这样的优点是：假定程序中还有别的观察者，那么只要这个观察者也是相同的接口实现即可。一个被观察者可以对应多个观察者，当被观察者发生变化的时候，他可以将消息一一通知给所有的观察者。基于接口，而不是具体的实现——这一点为程序提供了更大的灵活性。

### (4).委托

委托和事件在.NET Framework中的应用非常广泛，然而，较好地理解委托和事件对很多接触C#时间不长的人来说并不容易。

委托是一个类，它定义了方法的类型，使得可以将方法当作另一个方法的参数来进行传递，这种将方法动态地赋给参数的做法，可以避免在程序中大量使用If-Else(Switch)语句，同时使得程序具有更好的[可扩展性](https://www.baidu.com/s?wd=%E5%8F%AF%E6%89%A9%E5%B1%95%E6%80%A7&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)。  
使用委托可以将多个方法绑定到同一个委托变量，当调用此变量时(这里用“调用”这个词，是因为此变量代表一个方法)，可以依次调用所有绑定的方法。

### (5).事件

事件就是一个特殊的委托，委托和事件就类似于字段和属性的关系，事件是对委托做了一个封装。

事件是委托的封装，可以理解为一种特殊的委托。

事件里面其实就是两个方法(即add\_event()和remove\_event())和一个私有的委托变量，这两个方法里面分别是对这个私有的委托变量进行的合并和移除，当调用事件的+=时其实是调用的事件里面的add\_event()方法，同样-=调用的是remove\_event()方法。

（4）事件只能够从对象外部增加新的响应方法和删除已知的响应方法，而不能主动去触发事件和获取其他注册的响应方法等信息。如果使用公有的delegate则不能做这些限制，也就是说事件对委托做了限制，使委托使用起来更加方便。也有人说事件是对委托的阉割，大概也是这个意思。

（5）观察者模式很容易的就能体现出委托与事件；

### (6).委托与事件的区别

委托既可以用=赋值，也可以用+=赋值，事件只能用+=赋值；

委托可以在类外部进行调用，事件不可以；

委托是一个类型，事件不是。

## 3、解法对比及优缺点

优点：1.可以大量节省内存占用，减少事件注册。比如ul上代理所有li的click事件就很不错。  
2.可以实现当新增子对象时，无需再对其进行事件绑定，对于动态内容部分尤为合适  
缺点：事件代理的常用应用应该仅限于上述需求，如果把所有事件都用事件代理，可能会出现事件误判。即本不该被触发的事件被绑定上了事件。

## 4、延伸及扩展问题回答参考

### (1).什么时候选择使用事件委托

事件委托也是这样的，如果事件委托没有缺点，那么它就不仅仅是一个解决方案了，而是会被浏览器直接纳入规范了吧，那么当前的事件绑定规范，就要直接改掉了

既然如此，那么什么时候，才适合使用事件委托呢，如何能更优的使用呢？

结合前面我们说到的，事件委托影响性能的因素：

1：元素中，绑定事件委托的次数；

2：点击的最底层元素，到绑定事件元素之间的DOM层数；

结合这三点，在必须使用事件委托的地方，可以如下的处理：

1：只在必须的地方，使用事件委托，比如：ajax的局部刷新区域

2：尽量的减少绑定的层级，不在body元素上，进行绑定

3：减少绑定的次数，如果可以，那么把多个事件的绑定，合并到一次事件委托中去，由这个事件委托的回调，来进行分发。

### (2).提高事件委托性能的解决方案

看完前面的事件委托的一些瓶颈之外，现在要给出一些解决的方案了：

1：降低层级，这个比较好实现，在开发中，直接把事件绑定在低层级的元素上即可，这个无法继续优化；

2：减少绑定的次数，现在只能在这个点上继续优化了。

## 5、项目中体现经验的点

观察者模式经常运用事件与委托，可以最大程度解耦合，减少逻辑复杂度，解决项目中隐藏的bug，提升工作效率。

## 6、论坛参考

1,<https://baike.baidu.com/item/%E8%A7%82%E5%AF%9F%E8%80%85%E6%A8%A1%E5%BC%8F/5881786?fr=aladdin>

2,http://www.runoob.com/design-pattern/observer-pattern.html

3,<http://baike.baidu.com/view/1935199.htm>

4,https://www.cnblogs.com/RaindayXia/p/5261797.html

5.<https://blog.csdn.net/chopper7278/article/details/3145000>

6.<http://www.360doc.com/content/14/0228/08/10504424_356339873.shtml>

# 九、简述WebApi与WebService、WCF三种技术，并说明他们的区别

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 面试概率 | 级别 | 应用模块或方向 | 解决问题 | 考核点 |
| 80% | 4星 | .Net | 模式 | C# |

## 通常解法

WebService: 一种为SOA类型的项目提供服务的，基于SOAP和XML的标准

WCf：微软开发的数据通信程序框架，他把 .net Remoting,WebService,Socket的机制整合在一起。还融合了HTTP，FTP技术。

WebApi：是一种为SOA类型的项目提供接口的，轻量级的基于Http的框架

## 大牛级解法

**WebService**

Web服务：基于HTTP和XML的技术，HTTP是互联网上应用最为广泛的一种网络协议，而XML是跨平台的基础。

优点:

1、可以跨平台。

2、此外，WebService实现的技术难度要比CORBA和DCOM小得多。

3、在C/S的程序中，WebService可以实现网页无整体刷新的与服务器打交道并取数。

缺点：

1、WebService使用了XML对数据封装，会造成大量的数据要在网络中传输。

2、WebService规范没有规定任何与实现相关的细节，包括对象模型、编程语言，这一点，不如CORBA。

**WCF**

统一性：WCF全是用托管代码写的，所有WCF开发的程序和.net开发的程序没有什么区别，完全可以向创建面相对象的程序那样用WCF来创建面向服务的程序。

互操作性：WCF的通信机制是SOAP，所有保证了系统之间的互操作性，还可以跨进程，跨机器甚至还可以跨平台通信，应用程序可以运行在Windows操作系统下，还可以运行在Sun Solaris，HP Unix,Linux等操作系统下。

安全与可信赖

WS-Security,WS-Trust和WS-SecureConversation被添加到了SOAP中，主要是用于用户认证，数据完整新验证，数据隐私等安全因素，

在SOAP的header中增加了WS-ReliableMessaging允许可信赖的端对端通信。而建立在WS-Coordination和WS- AtomicTransaction之上的基于SOAP格式交换的信息，则支持两阶段的事务提交

多重WS—policy在WCF中都给与了支持。对于Messaging来说，SOAP是WebService的基本协议，包含了消息头，和消息体，在消息头中，定义了WS—Addressing用于定位SOAP消息的地址信息，同时还包含了MTOM（消息传输优化机制）。

兼容性

WCF考虑到了原有系统的兼容新，安装WCF不会影响原有的技术ASMX和.Net Remoting。对于WCF和ASMX来说，虽然两个都是用了SOAP，但基于WCF开发的应用程序，仍然可以知己与ASMX交互。

**WebApi**

1.Web API是一种无限接近于RESTful风格的轻型框架，且不是微软提出来的

2.包含两种格式：XML和JSON

1）XML：扩展标记语言，用于标记电子文件使其具有结构性的标记语言，用来标记数据、定义数据类型，是一种允许用户对自己的标记语言进行定义的源语言

XML的优点：功能强大，数据类型强大，对数据内容的描述准确，有了它，再也不用担心后端发来的数据格式不对了

XML的缺点：过分臃肿，有时强大的功能并不是我们所需要的；解析规则多，对追求高开发速度和低开发门槛的企业来说，是个致命伤。

2）JSON：是一种轻量级的数据交换格式，具有良好的可读和便于快速编写的特性。可以在不同平台之间进行数据交换。JSON采用兼容性很高的、完全独立于语言文本格式，同时也具备类似于C语言的习惯体系的行为。这些特性使JSON成为理想的数据交换语言。

Json的优点：数据结构清晰，可读性比XML强，解析规则简单；轻量级，灵活，性能比XML高

Json的缺点：数据类型定义不明确，如：{"price":12580}在json里，你无法知道这个价格是int, float还是double。

## 3、解法对比及优缺点

通用解法简单说了一下WCF,WebApi和WebService的信息，

大牛解法，详细的介绍了WCF,WebApi和WebService，及其机制。

## 延伸及扩展问题回答参考

**WebService**

Webservice涉及的协议及其作用？  
1、SOAP:简单对象访问协议  
   SOAP是表示信息交换的协议，是一种基于XML和Http的协议，它用于在分布式环境中发送消息，并执行远程过程调用。SOAP采用HTTP作为底层通讯协议，RPC作为一致性的调用途径，XML作为数据传送的格式，允许服务提供者和服务客户经过防火墙在INTERNET进行通讯交互。  
2、Http：超文本传送协议  
   Http在Webservice里的作用是SOAP的载体，客户端生成的SOAP请求会被嵌入在一个HTTP POST请求中，发送到Web服务器。有了Http，WebService在客户端和服务器之间比较容易通过防火墙和代理服务器进行通信。  
3、XML：可扩展标记语言  
   可扩展的标记语言（XML）是Web service平台中表示数据的基本格式，在SOAP中XML是它的编码模式。除了易于建立和易于分析外，XML主要的优点在于它既是平台无关的，又是厂商无关的。无关性是比技术优越性更重要的：软件厂商是不会选择一个由竞争对手所发明的技术的。

**WCF**

wcf的错误处理：FaultException 与 FaultReason 的搭配，

**WebApi**

1. webapi2跨域问题  
   跨域问题产生的原因：同源策略（Same origin policy）是一种约定，它是浏览器最核心也最基本的安全功能。、  
   解决方法：1）在nuget中搜索cors，安装Microsoft C:\Users\ZX50\AppData\Roaming\Tencent\QQ\Temp\%W@GJ$ACOF(TYDYECOKVDYB.pngASP.NET Web API 2.2 Cross-Origin Support  
    2）在文件WebApiConfig.cs中配置跨域设置  
    var cors=new EnableCorsAttribute("\*","\*","\*")'config.EnableCors(cors);(注：这样设置可以实现简单请求的跨域)  
    3）因为前端提交ajax请求，用的类型是application/json。发送post请求前会先发送一个预请求，就是option请求。要在Web.config中配置下处理option请求，让后端能够正确的响应option请求，前端才会继续发送post请求。这两个处理程序会影响option请求的正确处理。  
   2.延迟调用或多次调用第三方的Web API服务  
       1)当我们调用第三方的Web API服务的时候，不一定每次都是成功的。这时候，我们可能会再多尝试几次，也有可能延迟一段时间再去尝试调用服务。  
       2)Task的静态方法Delay允许我们延迟执行某个Task，此方法可以让我们做到延迟一段时间再去调用服务；多尝试几次调用如何实现呢？可以用循环遍历。

## 5、项目中体现经验的点

如果服务需要在Http协议上，并且需要利用http协议功能的时候，如果服务需要被各种客户端调用，可以选择WebApi

## 6、论坛参考

WCF：https://baike.so.com/doc/5568067-5783226.html

WebApi：<https://www.cnblogs.com/guyun/p/4589115.html>

WebService：<https://www.cnblogs.com/yangang2013/p/5708647.html>

三者的区别：https://www.cnblogs.com/markli/p/4460564.html

# 十．什么是NoSql

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 面试概率 | 级别 | 应用模块或方向 | 解决问题 | 考核点 |
| 80% | 4星 | .Net | 模式 | C# |

## 1、通用解法

NoSql泛指非关系型数据库，具有易扩展，大数据量高性能，多样灵活的数据模型，常用的非关系型数据库有Redis，Memcached，MongoDB等

## 2、大牛解法

NoSQL（Not Only SQL），泛指非关系型的数据库，是对不同于传统的关系型数据库的数据库管理系统的统称，强调Key-Value Stores和文档数据库的优点。为了解决大规模数据集合多重数据种类带来的挑战而兴起的数据库。有着模式自由，逆规范化，多分区存储，弹性可扩展，多副本异步复制，软事务等特点。NoSQL用于超大规模数据的存储。NoSQL数据库的四大分类：键 - 值对存储，列存储，文档存储，图形数据库

## 3、解法对比及优缺点

从以下几个维度，对 redis、memcache、mongoDB 做了对比。

1、性能

都比较高，性能对我们来说应该都不是瓶颈。

总体来讲，TPS 方面 redis 和 memcache 差不多，要大于 mongodb。

2、操作的便利性

memcache 数据结构单一。（key-value）

redis 丰富一些，数据操作方面，redis 更好一些，较少的网络 IO 次数，同时还提供 list，set，

hash 等数据结构的存储。

mongodb 支持丰富的数据表达，索引，最类似关系型数据库，支持的查询语言非常丰富。

3、内存空间的大小和数据量的大小

redis 在 2.0 版本后增加了自己的 VM 特性，突破物理内存的限制；可以对 key value 设置过

期时间（类似 memcache）

memcache 可以修改最大可用内存,采用 LRU 算法。Memcached 代理软件 magent，比如建立

10 台 4G 的 Memcache 集群，就相当于有了 40G。 magent -s 10.1.2.1 -s 10.1.2.2:11211 -b

10.1.2.3:14000 mongoDB 适合大数据量的存储，依赖操作系统 VM 做内存管理，吃内存也比较厉害，服务

不要和别的服务在一起。

4、可用性（单点问题）

对于单点问题，

redis，依赖客户端来实现分布式读写；主从复制时，每次从节点重新连接主节点都要依赖整

个快照,无增量复制，因性能和效率问题，

所以单点问题比较复杂；不支持自动 sharding,需要依赖程序设定一致 hash 机制。

一种替代方案是，不用 redis 本身的复制机制，采用自己做主动复制（多份存储），或者改成

增量复制的方式（需要自己实现），一致性问题和性能的权衡

Memcache 本身没有数据冗余机制，也没必要；对于故障预防，采用依赖成熟的 hash 或者环

状的算法，解决单点故障引起的抖动问题。

mongoDB 支持 master-slave,replicaset（内部采用 paxos 选举算法，自动故障恢复）,auto sharding 机制，对客户端屏蔽了故障转移和切分机制。

5、可靠性（持久化）

对于数据持久化和数据恢复，

redis 支持（快照、AOF）：依赖快照进行持久化，aof 增强了可靠性的同时，对性能有所影

响

memcache 不支持，通常用在做缓存,提升性能；

MongoDB 从 1.8 版本开始采用 binlog 方式支持持久化的可靠性

6、数据一致性（事务支持）

Memcache 在并发场景下，用 cas 保证一致性redis 事务支持比较弱，只能保证事务中的每个操作连续执行

mongoDB 不支持事务

7、数据分析

mongoDB 内置了数据分析的功能(mapreduce),其他不支持

8、应用场景

redis：数据量较小的更性能操作和运算上

memcache：用于在动态系统中减少数据库负载，提升性能;做缓存，提高性能（适合读多写

少，对于数据量比较大，可以采用 sharding）

MongoDB:主要解决海量数据的访问效率问题。

表格比较：

数据库 memcache redis

类型 内存数据库 内存数据库

数据类型 在定义value时就要固定数据类型 不需要

虚拟内存 不支持 支持

过期策略 支持 支持

存储数据安全 不支持 使用 save 存储到 dump.rdb 中

灾难恢复 不支持 append only file(aof)用于数据恢复

性能

1、类型——memcache 和 redis 都是将数据存放在内存，所以是内存数据库。当然，memcache 也可用于缓存其他东西，例如图片等等。

2、 数据类型——Memcache 在添加数据时就要指定数据的字节长度,而 redis 不需要。

3、 虚拟内存——当物理内存用完时，可以将一些很久没用到的 value 交换到磁盘。

4、 过期策略——memcache 在 set 时就指定，例如 set key1 0 0 8,即永不过期。Redis 可以通

过例如 expire 设定，例如 expire name 10。

5、 分布式——设定 memcache 集群，利用 magent 做一主多从;redis 可以做一主多从。都可

以一主一从。

6、 存储数据安全——memcache 断电就断了，数据没了；redis 可以定期 save 到磁盘。

7、 灾难恢复——memcache 同上，redis 丢了后可以通过 aof 恢复。

[NoSQL数据库的四大分类表格分析](https://baike.baidu.com/item/NoSQL/8828247 "百度词库")

## 4、延伸及扩展问题回答参考

问题：redis的存储方式（键值对的存储方式）

答：现在比较常用的是键值对的存储方式，说到存储方式，需用从哈希算法说起，hash算法将对象转换为一个散列①，我们在使用的时候会将这个散列转换为一个大数，这就是hashcode，而hashmap高度依赖于hash和hashcode

hashtable默认会开辟4个存储空间，将获取到的hashcode求磨，算出下标志②。也就是说hashtable是键值对的存储方式，

我们一般会用Hashmap分表分库，hashmap的原理和hashtable相似，他们之间的区别在这里就不细说了，hashmap分表分库也是对hashcode求魔然后找下标值，但是下标值里存在数据的话，会自动生成链路（在hashmap中的是倒链③）

注释：

①：HashMap 之所以速度快，因为他使用的是散列表，根据 key 的 hashcode 值生成数组下标（通过内存地址直接查找，没有任何判断），时间复杂度完美情况下可以达到 n1（和数组相同，但是比数组用着爽多了，但是需要多出很多内存，相当于以空间换时间）

②：假若hashcode为112，hashtable为4个空间，那么这个数据存储的下标值为 112%4=0，也就是说存在了下标为0的空间

③：链表存储是不连续的，查询慢，插入块。数组是连续的，查询快，插入慢

## 项目中体现经验的点 ##

在项目中使用了Redis做多点登录，将生成的token保存在redis中，在数据查询的过程中，先从缓存读取数据，加快页面运行速度，减轻服务器压力

## 5、项目中体现经验的点

在项目中使用了Redis做多点登录，将生成的token保存在redis中，在数据查询的过程中，先从缓存读取数据，加快页面运行速度，减轻服务器压力

## 6、论坛参考

[常见的NoSQL数据库](https://www.cnblogs.com/yejg1212/p/3584094.html "常见的NoSQL数据库")》

《[NoSQL简介](https://blog.csdn.net/mevicky/article/details/47397335 "NoSql简介")》

《[8 种 NoSQL 数据库系统对比](http://blog.jobbole.com/1344/ "8 种 NoSQL 数据库系统对比")》

[关于 redis、memcache、mongoDB 的对比](https://www.cnblogs.com/94cool/p/3247307.html "关于 redis、memcache、mongoDB 的对比")》