物联网学院

高频问题集

目录

[一、 接口和抽象类的区别？ 1](#_Toc854)

[1. 通常解法 1](#_Toc15127)

[2. 通用大牛级解法 1](#_Toc5557)

[3. 解法对比及优缺点 2](#_Toc3430)

[4. 延伸及扩展问题回答参考 2](#_Toc10275)

[5. 项目中体现经验的点 2](#_Toc18739)

[6. 论坛参考 3](#_Toc28192)

[二、什么是方法的重载 3](#_Toc4912)

[1. 通常解法 3](#_Toc9193)

[2. 通用大牛级解法 3](#_Toc32695)

[3. 解法对比及优缺点 3](#_Toc22865)

[4. 延伸及扩展问题回答参考 3](#_Toc165)

[5. 项目中体现经验的点 4](#_Toc15541)

[6. 论坛参考 4](#_Toc18188)

[三、什么是方法的重写？ 4](#_Toc11871)

[1. 通常解法 4](#_Toc24460)

[2. 通用大牛级解法 4](#_Toc26629)

[3. 解法对比及优缺点 5](#_Toc17997)

[4. 延伸及扩展问题回答参考 5](#_Toc8526)

[5. 项目中体现经验的点 5](#_Toc22512)

[6. 论坛参考 5](#_Toc8584)

[四、SQL查询的优化方案有哪些？ 5](#_Toc28408)

[1. 通常解法 6](#_Toc28652)

[2. 通用大牛级解法 6](#_Toc14707)

[3. 解法对比及优缺点 8](#_Toc7974)

[4. 延伸及扩展问题回答参考 8](#_Toc28906)

[5. 项目中体现经验的点 8](#_Toc22010)

[6. 论坛参考 9](#_Toc12005)

[五、如何解决页面渲染速度慢的问题 9](#_Toc1600)

[1. 通常解法 9](#_Toc10890)

[2. 通用大牛级解法 9](#_Toc3212)

[3. 解法对比及优缺点 11](#_Toc20501)

[4. 延伸及扩展问题回答参考 11](#_Toc1593)

[jquery和javascript、ajax的关系 11](#_Toc26948)

[5. 项目中体现经验的点 12](#_Toc22707)

[6. 论坛参考 12](#_Toc1511)

[六、什么是依赖注入 12](#_Toc31797)

[1、通常解法 12](#_Toc3641)

[2、通用大牛级解法 13](#_Toc27992)

[3、解法对比及优缺点 13](#_Toc29863)

[4、延伸及扩展问题回答参考 13](#_Toc1056)

[5、项目中体现经验的点 14](#_Toc27444)

[6、论坛参考 15](#_Toc9323)

[七、存储过程和函数的区别](#_Toc24733)

[1、通用解法 15](#_Toc6899)

[2、大牛解法 15](#_Toc27616)

[3、解法对比及优缺点 16](#_Toc21310)

[4、延伸及扩展问题回答参考 17](#_Toc16610)

[5、项目中体现经验的点 18](#_Toc13874)

[6、论坛参考 18](#_Toc24377)

[八、什么是工厂模式 19](#_Toc15915)

[1、 通常解法 19](#_Toc29810)

[2、通用大牛级解法 19](#_Toc29881)

[3、解法对比及优缺点 21](#_Toc30017)

[4、延伸及扩展问题回答参考 22](#_Toc28541)

[5、项目中体现经验的点 24](#_Toc19841)

[6、论坛参考 24](#_Toc19641)

[九、什么是Ajax异步调用24](#_Toc17652)

[1、通常解法 25](#_Toc30094)

[2、大牛级解法 25](#_Toc26387)

[3、解法对比及优缺点 26](#_Toc23294)

[4、延伸及扩展问题回答参考 26](#_Toc28285)

[5、项目中体现经验的点 27](#_Toc19500)

[6、论坛参考 27](#_Toc30919)

[十．Redis使用 27](#_Toc14195)

[1、通用解法 27](#_Toc11690)

[2、大牛解法  28](#_Toc7411)

[3、解法对比及优缺点 29](#_Toc29415)

[4、延伸及扩展问题回答参考 30](#_Toc28830)

[5、项目中体现经验的点 31](#_Toc1203)

[6、论坛参考 31](#_Toc15606)

# 接口和抽象类的区别？

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 面试概率 | 级别 | 应用模块或方向 | 解决问题 | 考核点 |
| 90% | 4星 | .Net | 接口和抽象类的区别 | C# |

## 通常解法

相同点：

不能被实例化，可以包含任意个抽象成员

不同点：

抽象类是类，具有类的特点，只能单基类继承，多接口实现，而接口不是类，可以继承多个接口，抽象类中可以包含已经实现的非抽象成员，而接口中所有成员都是抽象的，没有实现的。

## 通用大牛级解法

相同点：

* + - 1. 都不能被直接实例化，都可以通过继承实现其抽象方法。
      2. 都是面向抽象编程的技术基础，实现诸多模式

不同点：

1.接口可以多继承，抽象类不能实现多继承。

2.接口只能定义抽象规则，抽象类既能定义抽象规则也能提供已实现的成员

3.接口是一组行为规范，抽象类是一个不完全的类，着重于族的概念

4接口支持回调，抽象类不能实现回调，因为继承不支持。

5接口只包含方法、属性、索引器、事件的签名，但不能定义字段和包含实现的方法，抽象类可以定义属性、字段、包含有实现的方法

6.接口可以作用于值类型和引用类型，抽象类只能作用于引用类型（例如：Struct只能继承接口）

7.抽象类应主要用于关系密切的对象，而接口则是适合为不相关的类提供通用功能

8.接口着重于Can—Do关系类型，抽象类则偏重于IS—A式关系。

9.接口多定义对象的行为，抽象类多定义对象的属性。

10.如果要设计小而简练的功能块，则使用接口，如果要设计大的功能单元，则使用抽象类

11.接口对应是面向对象编程思想的重要原则：面向接口编程，

抽象类对应则是设计模式的的精髓：面相抽象编程

## 解法对比及优缺点

常见解法，简要说明接口与抽象类的异同。

通用解法，详细说明接口与抽象类的规则和使用场合。

## 延伸及扩展问题回答参考

**问题**：阐述面向接口与面向对象的关系

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

项目框架搭建过程中应遵循依赖倒置原则，通过依赖注入，实现控制反转的目的，

完成对于项目各个层级之间的解耦。

## 论坛参考

<http://www.cnblogs.com/ronli/archive/2011/10/26/2224654.html>

# 二、什么是方法的重载

## 通常解法

方法重载是指在同一个类中方法同名，参数不同，调用时根据实参的形式，选择匹配的方法执行操作，参数不同分为参数类型、参数个数、参数顺序不同三种情况

Html5）、Cache、数据库、GET、POST、PUT、DELETE

## 通用大牛级解法

C#中编译时的多态性是通过重载（overload）来实现的，而运行时的多态则是统重写实现。

重载并不仅仅局限于方法重载，不但可以用方法重载，也可以用运算符重载，

方法重载只有通过参数不同来判断调用哪个方法，而不能通过返回值

重写则是继承时发生，在子类中重新定义父类中的方法，子类中的方法和父类的方法是一样的，即方法名，参数，返回值都相同。常用与虚方法和抽象类

## 解法对比及优缺点

一般解法：说明重载的具体实现和重载的基本概念

通用解法：通过多态引入重载与重写之间的关系

## 延伸及扩展问题回答参考

多态的三种体现方式

解答：

重写，重载，new（隐藏基类）

## 项目中体现经验的点

方法的重载实现了编译时的多态，更好的体现面向对象编程的思想。

## 论坛参考

https://blog.csdn.net/wochuailimin/article/details/5629041

# 三、什么是方法的重写？

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 面试概率 | 级别 | 应用模块或方向 | 解决问题 | 考核点 |
| 95% | 4星 | .Net | 方法的重写 | C#技能点 |

## 通常解法

重写要求方法名、参数和返回值相同，重写是为了增强类的重用性和复用性，扩展性；重写是对类中方法的扩充

## 通用大牛级解法

通过为声明的方法提供新的实现，派生类可以重写基类的方法。在 C# 中，必须使用 virtual 修饰符才能将方法显式标记为虚方法。

派生类中要被C#方法重写是用 virtual 修饰符声明的。而在派生类中，已被C#方法重写用 override 修饰符声明。

override 修饰符表示派生类的方法或属性，这个派生类代替基类中具有相同的名称和签名的类。要被重写的基方法必须声明为 virtual、abstract 或 override：以这种方式重写非虚方法或静态方法是不可能的。已被C#方法重写或属性和重写方法或属性必须具有相同的访问级修饰符。

## 解法对比及优缺点

一般解法：简单说明方法的重写

通用解法：对于方法的重写进行详细说明。

## 延伸及扩展问题回答参考

多态的好处

解答：

设计原则之一就是要依赖于抽象，而不依赖于具体，增加灵活性。

多态就是为了体现这一原则。

常见的多态体现，如接口、抽象类等等

## 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/sylone/p/6081141.html

# 四、SQL查询的优化方案有哪些？

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

## 通常解法

1、在表中建立索引，优先考虑where、group by使用到的字段。

2、尽量避免使用select \*，返回无用的字段会降低查询效率。

3、尽量避免使用in 和not in，会导致数据库引擎放弃索引进行全表扫描。

4、尽量避免使用or，会导致数据库引擎放弃索引进行全表扫描。

5、尽量避免在字段开头模糊查询，会导致数据库引擎放弃索引进行全表扫描。

6、尽量避免进行null值的判断，会导致数据库引擎放弃索引进行全表扫描。

7、尽量避免在where条件中等号的左侧进行表达式、函数操作，会导致数据库引擎放弃索引进行全表扫描。

8、当数据量大时，避免使用where 1=1的条件。通常为了方便拼装查询条件，我们会默认使用该条件，数据库引擎会放弃索引进行全表扫描。

## 通用大牛级解法

**数据库的读写分离**

实现原理：读写分离简单的说是把对数据库读和写的操作分开对应不同的数据库服务器，这样能有效地减轻数据库压力，也能减轻io压力。主数据库提供写操作，从数据库提供

读操作，其实在很多系统中，主要是读的操作。当主数据库进行写操作时，数据要同步到从的数据库，这样才能有效保证数据库完整性。

实现方法

SQL SERVER复制技术类型有三种，分别是：快照复制、事务复制、合并复制。

SQL SERVER 主要采用出版物、订阅的方式来处理复制。

**数据库/数据表 拆分(分布式)**

通过某种特定的条件，将存放在同一个数据库中的数据分散存放到多个数据库上，实现分布存储，通过路由规则路由访问特定的数据库，这样一来每次访问面对的就不是单台

服务器了，而是N台服务器，这样就可以降低单台机器的负载压力。提示：sqlserver 2005版本之后，可以友好的支持“表分区”。

垂直(纵向)拆分：是指按功能模块拆分，比如分为订单库、商品库、用户库...这种方式多个数据库之间的表结构不同。水平(横向)拆分：将同一个表的数据进行分块保存到不同

的数据库中，这些数据库中的表结构完全相同。 ▲(纵向拆分)  水平(横向)拆分

**临时表：**

SqlServer中可以创建本地和全局临时表。本地临时表仅在当前会话中可见；全局临时表在所有会话中都可见。

本地临时表的名称前面有一个#，如(#table\_name)，而全局临时表的名称前面有两个## ，如(##table\_name)。

**sqlserver 多库查询sp\_addlinkedserver**

mssql在使用多库查询的时候会用到链接服务器，以下为链接服务器的添加方法，添加完了即可实现任意改服务器的多库查询了

Exec sp\_droplinkedsrvlogin ZYB,Null --删除映射（录与链接服务器上远程登录之间的映射）

Exec sp\_dropserver ZYB --删除远程服务器链接

EXEC sp\_addlinkedserver

@server='ZYB',--被访问的服务器别名

@srvproduct='',

@provider='SQLOLEDB',

@datasrc="/Server2" --要访问的服务器

EXEC sp\_addlinkedsrvlogin

'ZYB', --被访问的服务器别名

'false',

NULL,

'sa', --帐号

'sa' --密码

使用实例：

Select \* from ZYB.CDCenter20110822.dbo.cardbase

**数据库弱一致性四个隔离级别**

SQL-92标准中定义了四个隔离级别，这四个隔离级别在以前版本的SQL Server中即受到支持：

**READ UNCOMMITTED**

READ UNCOMMITTED是限制性最弱的隔离级别，因为该级别忽略其他事务放置的锁。使用READ UNCOMMITTED级别执行的事务，可以读取尚未由其他事务提交的修改后

的数据值，这些行为称为“脏”读。这是因为**在Read Uncommitted级别下，读取数据不需要加S锁，这样就不会跟被修改的数据上的X锁冲突**。比如，事务1修改一行，事务2在

事务1提交之前读取了这一行。如果事务1回滚，事务2就读取了一行没有提交的数据，这样的数据我们认为是不存在的。

**READ COMMITTED**

READ COMMITTED(Nonrepeatable reads)是SQL Server默认的隔离级别。该级别通过指定语句不能读取其他事务已修改但是尚未提交的数据值，禁止执行脏读。在当前事务

中的各个语句执行之间，其他事务仍可以修改、插入或删除数据，从而产生无法重复的读操作，或“影子”数据。比如，事务1读取了一行，事务2修改或者删除这一行并且提交。

如果事务1想再一次读取这一行，它将获得修改后的数据或者发现这一样已经被删除，因此事务的第二次读取结果与第一次读取结果不同，因此也叫不可重复读。

## 解法对比及优缺点

一般解法：查询语句优化

通用解法：数据库中表结构和具体库进行优化

## 延伸及扩展问题回答参考

问题：大批量数据插入形成高并发如何处理

异步编程

Redis缓存，保存入缓存中，分批次插入数据库

数据库读写分离等等

## 项目中体现经验的点

与数据库交互时，对于数据表复杂逻辑查询时，可以通过对于sql语句的优化，提高查询速度

## 论坛参考

<https://www.cnblogs.com/ldyblogs/p/event.html>

# 五、如何解决页面渲染速度慢的问题

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

## 通常解法

Ajax异步加载，减少渲染阻塞的js和css

推迟js的解析

不重要的脚本可推迟执行解析。

## 通用大牛级解法

避免长时间运行js

避免长时间运行js，因为长时间运行js将导致CSSOM，DOM和渲染页面的阻塞。如果必须执行一系列的js文件，可以将其细分成更多的状态来允许浏览器在不同的状态中间执行其他操作。

优化css使用

css被要求用来构造render tree并且在页面的最初渲染的时候页面css阻塞js的执行。

你需要确保不必要的css文件被标记为不必要（例如 @media参数设置等），还要确保关键css文件最小化，以及关键css文件加载时间最小化。

将css放置在文档头部

所有的css资源应该被放在文档头部，这样浏览器可以尽快的执行它们。

避免使用@import

CSS 导入 (@import) 可以检测和导入其他的css文件，但是最好不要使用，因为被导入的文件只有在@import规则本身被接受和解析后才会执行，增加了网络往返。

将css写在文档内部

为了最好的性能，你可能会考虑将css写在文档内部，因为这样将会减少外部css文件下载时所耗费的网络往返次数，且只有一个HTML文档被当成关键资源。

页面静态化：提前将无安全风险的页面生成为静态html页面，提高加载速度

分布式服务器：将文字等不大的文件放在一台服务器上进行加载，图片、视频、音乐等占用资源大的文件放置在专门的服务器去加载，提高加载效率

## 解法对比及优缺点

一般解法：针对页面js优化

通用解法：通过页面优化引出静态化和分布式两种技术

## 延伸及扩展问题回答参考

## jquery和javascript、ajax的关系

解答：

1、JS是一门前端语言。

2、Ajax是一门技术，它提供了异步更新的机制，使用客户端与服务器间交换数据而非整个页面文档，实现页面的局部更新。

3、jQuery是一个框架，它对JS进行了封装，使其更方便使用。jQuery使得JS与Ajax的使用更方便

关系比喻：

若把js比作木头，那么jquery就是木板（半成品）

jQuery与ajax都是js的一个框架，各有各的功能

详细情况：

1.javascript是一种在客户端执行的脚本语言。ajax是基于javascript的一种技术，它主要用途是提供异步刷新（只刷新页面的一部分，而不是整个页面都刷新）。一个是语言,一个是技术,两者有本质区别.

2.javascript是一种在浏览器端执行的脚本语言，Ajax是一种创建交互式网页应用的开发技术 ，它是利用了一系列相关的技术其中就包括javascript。

## 项目中体现经验的点

通过上述种种方式：如页面静态化、分布式等技术，从速度和效率两方面提高页面加载，解决页面渲染速度慢的问题

## 论坛参考

<https://www.cnblogs.com/RachelChen/p/5456193.html>

**https://www.cnblogs.com/thomasbc/p/6650119.html**

# 六、什么是依赖注入

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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(DI)和控制反转(IOC)是同一个概念。具体含义是:当某个角色需要另一个角色的协助时，在一般的程序设计过程中，通常由调用者来创建被调用者的实例。但在Spring里，创建被调用者的工作不再由调用者来完成，因此称为控制反转;创建被调用者 实例的工作通常由Spring容器来完成，然后注入调用者，因此也称为依赖注入。依赖注入（Dependency Injection）是用于实现控制反转（Inversion of Control）的最常见的方式之一（控制反转还有一种常见的实现方式称为依赖查找）

## 2、通用大牛级解法

所谓依赖注入就是指：在运行期，由外部容器动态地将依赖对象注入到组件中。当spring容器启动后，spring容器初始化，创建并管理bean对象，以及销毁它。所以我们只需从容器直接获取Bean对象就行，而不用编写一句代码来创建bean对象。这种现象就称作控制反转，即应用本身不负责依赖对象的创建及维护，依赖对象的创建及维护是由外部容器负责的。这样控制权就由应用转移到了外部容器，控制权的转移就是所谓反转。

参考文档<https://blog.csdn.net/coderder/article/details/51897721>

## 3、解法对比及优缺点

一般解法：简单的叙述了依赖注入和控制反转的关系，以及控制反转的思想

通用解法：详细了解依赖注入的实现方法及原理

## 4、延伸及扩展问题回答参考

**A依赖注入的优点：**

1、将被依赖类的创建代码从 依赖类中移出，不用显式的写new。

2、可以单独维护  被依赖类的创建过程。

3、方便该类的被共享。 如果该类初始化时，所需属性很多，使用配置，远比硬代码编写简单。

**B依赖注入有几种方式，分别是什么？**

三种，

（1）构造方法注入：即被注入对象可以通过在其构造方法中声明依赖对象的参数列表，让外部（通常是IOC容器）知道它需要哪些依赖对象，然后IOC容器会检查被注入对象的构造方法， 取得其所需要的依赖对象列表，进而为其注入相应对象。

（2）setter方法注入：即当前对象只需要为其依赖对象所对应的属性添加setter方法，IOC容器通过此setter方法将相应的依赖对象设置到被注入对象的方式即setter方法注入。

（3）接口注入：接口注入有点复杂，被注入对象如果想要IOC容器为其注入依赖对象，就必须实现某个接口，这个接口提供一个方法，用来为被注入对象注入依赖对象，IOC容器通过接口方法将依赖对象注入到被注入对象中去。相对于前两种注入方式，接口注入比繁琐和死板，被注入对象就必须专声明和实现另外的接口。

## 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/guanghuichenshao/article/details/79545322>

注入方法优缺点：

<https://blog.csdn.net/qq_35091777/article/details/78358887>

# 七、存储过程和函数的区别

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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

存储过程和函数目的是为了 可重复地 执行操作数据库的sql语句的集合。区别是写法和调用上。写法上：存储过程的参数列表可以有输入参数、输出参数、可输入输出的参数；函数的参数列表只有输入参数，并且有return <返回值类型，无长度说明>。

返回值上：存储过程的返回值，可以有多个值，函数的返回值，只有一个值。

## 2、大牛解法

函数：只能返回一个变量的限制。而存储过程可以返回多个。  
函数是可以嵌入在sql中使用的,可以在select中调用，而存储过程不行，执行的本质都一样。   
函数限制比较多，比如不能用临时表，只能用表变量．还有一些函数都不可用等等．而存储过程的限制相对就比较少  
1. 一般来说，存储过程实现的功能要复杂一点，而函数的实现的功能针对性比较强。  
2. 对于存储过程来说可以返回参数，而函数只能返回值或者表对象。  
3. 存储过程一般是作为一个独立的部分来执行，而函数可以作为查询语句的一个部分来调用，由于函数可以返回一个表对象，因此它可以在查询语句 中位于FROM关键字的后面。  
4. 当存储过程和函数被执行的时候，SQL Manager会到PRocedure cache中去取相应的查询语句，如果在procedure cache里没有相应的查询语句，SQL Manager就会对存储过程和函数进行。

## 3、解法对比及优缺点

一般解法：简单描述存储过程和函数写法上的区别。

通用解法：把存储过程和函数的主要区别阐述详细，主要的优缺点分析完毕，再推荐使用存储过程，详细说明使用存储过程的好处。

## 4、延伸及扩展问题回答参考

1系统存储过程

以sp\_开头,用来进行系统的各项设定.取得信息.相关管理工作。

2本地存储过程

用户创建的存储过程是由用户创建并完成某一特定功能的存储过程，事实上一般所说的存储过程就是指本地存储过程。

3临时存储过程

分为两种存储过程:

一是本地临时存储过程，以井字号(#)作为其名称的第一个字符，则该存储过程将成为一个存放在tempdb数据库中的本地临时存储过程，且只有创建它的用户才能执行它;

二是全局临时存储过程，以两个井字号(##)号开始，则该存储过程将成为一个存储在tempdb数据库中的全局临时存储过程，全局临时存储过程一旦创建，以后连接到服务器的任意用户都可以执行它，而且不需要特定的权限。

4远程存储过程

在SQL Server2005中，远程存储过程(Remote Stored Procedures)是位于远程服务器上的存储过程，通常可以使用[分布式查询](https://baike.so.com/doc/6972627-7195313.html" \t "_blank)和EXECUTE命令执行一个远程存储过程。

5扩展存储过程

扩展存储过程(Extended Stored Procedures)是用户可以使用外部程序语言编写的存储过程，而且扩展存储过程的名称通常以xp\_开头。

存储过程可以有多个in,out,inout参数，而存储函数只有输入参数类型，而不能带 输入类型

存储过程实现的功能要复杂一些；而存储函数的单一功能性(针对性)更强。

存储过程可以返回多个值；存储函数只能有一个返回值。

存储过程一般独立的来执行；而存储函数可以作为其他sql语句的组成部分来出现。

存储过程可以调用存储函数。但函数不能调用存储过程。

## 5、项目中体现经验的点

电商平台，查看商品的详细页面，URL地址需要传送商品相关的参数，数据之间的相互传递和进行编辑都需要用到。

## 6、论坛参考

<https://www.cnblogs.com/yanyao/p/5608907.html>

# 八、什么是工厂模式

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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

工厂模式是用工厂方法代替new操作的一种模式。工厂方法封装了多个相关联类的new方法，每次实例化这些类的时候不需要new多次，只需要调用工厂类的对应方法即可实例化这些类，并且是通过使用一个共同的接口来指向新创建的对象。

## 2、通用大牛级解法

**工厂模式分为三种：简单工厂、工厂方法、抽象工厂。**

简单工厂模式由三种角色组成：  
1、工厂类角色：这是本模式的核心，含有一定的商业逻辑和判断逻辑，根据逻辑不通，产生具体的工厂产品。如例子中的Driver类。  
2、抽象产品角色：它一般是具体产品继承的父类或者实现的接口。在java中由接口或者抽象类来实现。如例中的Car接口。  
3、具体产品角色：工厂类所创建的对象就是此角色的实例。在java中由一个具体类实现，如例子中的Benz、Bmw类。

工厂方法模式先来看下它的组成吧：  
1.抽象工厂角色： 这是工厂方法模式的核心，它与应用程序无关。是具体工厂角色必须实现的接口或者必须继承的父类。在java中它由抽象类或者接口来实现。  
2.具体工厂角色：它含有和具体业务逻辑有关的代码。由应用程序调用以创建对应的具体产品的对象。在java中它由具体的类来实现。  
3.抽象产品角色：它是具体产品继承的父类或者是实现的接口。在java中一般有抽象类或者接口来实现。  
4.具体产品角色：具体工厂角色所创建的对象就是此角色的实例。在java中由具体的类来实现。

抽象工厂模式的用意为：给客户端提供一个接口，可以创建多个产品族中的产品对象。  
而且使用抽象工厂模式还要满足一下条件：  
1.系统中有多个产品族，而系统一次只可能消费其中一族产品  
2.同属于同一个产品族的产品以其使用。  
来看看抽象工厂模式的各个角色（和工厂方法的如出一辙）：  
a.抽象工厂角色： 这是工厂方法模式的核心，它与应用程序无关。是具体工厂角色必须实现的接口或者必须继承的父类。在java中它由抽象类或者接口来实现。  
b.具体工厂角色：它含有和具体业务逻辑有关的代码。由应用程序调用以创建对应的具体产品的对象。在java中它由具体的类来实现。  
c.抽象产品角色：它是具体产品继承的父类或者是实现的接口。在java中一般有抽象类或者接口来实现。  
 具体产品角色：具体工厂角色所创建的对象就是此角色的实例。在java中由具体的类来实现。

## 3、解法对比及优缺点

**优点：**

1. 多态性：客户代码可以做到与特定应用无关，适用于任何实体类。
2. 子类提供挂钩。基类为工厂方法提供缺省实现，子类可以重写新的实现，也可以继承父类的实现。— 加一层间接性，增加了灵活性
3. 连接并行的类层次结构。
4. 良好的封装性，代码结构清晰。
5. 扩展性好，在增加产品类的情况下，只需要适当修改具体的工厂类或扩展一个工厂类，就可“拥抱变化”。
6. 屏蔽产品类。产品类的实现如何变化，调用者都不需要关心，只需关心产品的接口，只要接口保持不变，系统中的上层模块就不会发生变化。
7. 典型的解耦框架。高层模块只需要知道产品的抽象类，其他的实现类都不需要关心，符合迪米特法则，符合依赖倒置原则，符合里氏替换原则。

**缺点：**

1. 需要Creator和相应的子类作为factory method的载体，如果应用模型确实需要creator和子类存在，则很好；否则的话，需要增加一个类层次。
2. 简单工厂 ： 用来生产同一等级结构中的任意产品。（不支持拓展增加产品）
3. 工厂方法 ：用来生产同一等级结构中的固定产品。（支持拓展增加产品）
4. 抽象工厂 ：用来生产不同产品族的全部产品。（不支持拓展增加产品；支持增加产品族）

## 4、延伸及扩展问题回答参考

工厂方法模式是简单工厂模式的延伸，它继承了简单工厂模式的优点，同时还弥补了简单工厂模式的不足。工厂方法模式是使用频率最高的设计模式之一，是很多开源框架和API类库的核心模式。

参考优点：在工厂方法模式中，工厂方法用来创建客户所需要的产品，同时还向客户隐藏了哪种具体产品类将被实例化这一细节，用户只需要关心所需产品对应的工厂，无须关心创建细节，甚至无须知道具体产品类的类名。

基于工厂角色和产品角色的多态性设计是工厂方法模式的关键。它能够让工厂可以自主确定创建何种产品对象，而如何创建这个对象的细节则完全封装在具体工厂内部。工厂方法模式之所以又被称为多态工厂模式，就正是因为所有的具体工厂类都具有同一抽象父类。

使用工厂方法模式的另一个优点是在系统中加入新产品时，无须修改抽象工厂和抽象产品提供的接口，无须修改客户端，也无须修改其他的具体工厂和具体产品，而只要添加一个具体工厂和具体产品就可以了，这样，系统的可扩展性也就变得非常好，完全符合“开闭原则”。

参考缺点：在添加新产品时，需要编写新的具体产品类，而且还要提供与之对应的具体工厂类，系统中类的个数将成对增加，在一定程度上增加了系统的复杂度，有更多的类需要编译和运行，会给系统带来一些额外的开销。

由于考虑到系统的可扩展性，需要引入抽象层，在客户端代码中均使用抽象层进行定义，增加了系统的抽象性和理解难度，且在实现时可能需要用到DOM、反射等技术，增加了系统的实现难度。

适用的场景：客户端不知道它所需要的对象的类。在工厂方法模式中，客户端不需要知道具体产品类的类名，只需要知道所对应的工厂即可，具体的产品对象由具体工厂类创建，可将具体工厂类的类名存储在配置文件或数据库中。

抽象工厂类通过其子类来指定创建哪个对象。在工厂方法模式中，对于抽象工厂类只需要提供一个创建产品的接口，而由其子类来确定具体要创建的对象，利用面向对象的多态性和里氏代换原则，在程序运行时，子类对象将覆盖父类对象，从而使得系统更容易扩展。

## 5、项目中体现经验的点

若项目中有很多地方都需要生成A的对象，就可以使用工厂，只需要修改工厂中的代码，不需要再进行多次修改。

## 6、论坛参考

<https://blog.csdn.net/wss0130/article/details/51037721>

<http://www.runoob.com/design-pattern/factory-pattern.html>

# 九、什么是Ajax异步调用

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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# |

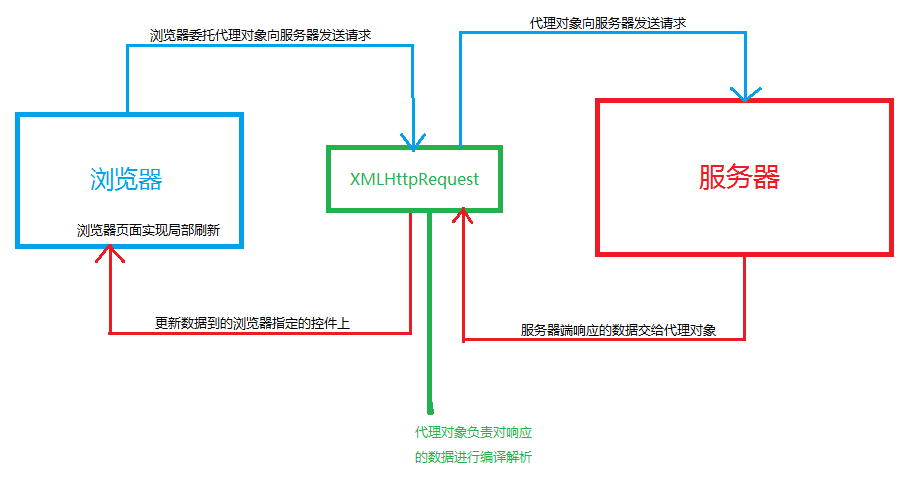
## 1、通常解法

发送方发出数据后，不等接收方发回响应，接着发送下个数据包的通讯方式。

当用户填写完一条信息后，该信息会自动向服务器提交，然后服务器响应客户端，在此过程中，用户依然在填写表格的信息，即向服务器请求多次，节省了用户的时间，提高了用户的体验。

## 2、大牛级解法

浏览器把请求交给代理对象—XMLHttpRequest（绝大多数浏览器都内置了这个对象），由代理对象向服务器发起请求，接收、解析服务器响应的数据，并把数据更新到浏览器指定的控件上。从而实现了页面数据的局部刷新。异步请求使浏览器不用等待服务器处理请求，不用重新加载整个页面来展示服务器响应的数据，在异步请求发送的过程中浏览器还能进行其它的操作。



## 3、解法对比及优缺点

通用解法通过一个小例子说明了异步请求

大牛解法通过图例阐述了ajax核心XMLHttpRequest对象

## 4、延伸及扩展问题回答参考

请介绍一下XMLhttprequest对象。

Ajax的核心是JavaScript对象XmlHttpRequest。该对象在Internet Explorer 5中首次引入，它是一种支持异步请求的技术。简而言之，XmlHttpRequest使您可以使用JavaScript向服务器提出请求并处理响应，而不阻塞用户。通过XMLHttpRequest对象，Web开发人员可以在页面加载以后进行页面的局部更新。

<https://blog.csdn.net/chow__zh/article/details/9149811>

## 5、项目中体现经验的点

项目中添加用户名验证，在填写完用户名后需请求后台判断是否重复，在请求期间，你可以继续填写其他信息

## 6、论坛参考

<https://www.cnblogs.com/cdf-opensource-007/p/6348108.html>

# 十．Redis使用

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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

Redis目前支持5种数据类型，分别是：

String（字符串）、List（列表）、Hash（字典）、Set（集合）、Sorted Set（有序集合）

String是简单的 key-value 键值对，value 不仅可以是 String，也可以是数字。

Redis列表是简单的字符串列表，简单的说就是一个链表或者说是一个队列。

Redis Hash对应Value内部实际就是一个HashMap。

Redis set对外提供的功能与list类似是一个列表的功能。

Redis有序集合类似Redis集合，不同的是增加了一个功能，即集合是有序的。

## 2、大牛解法

（1）快照（snapshots）  
　　缺省情况情况下，Redis把数据快照存放在磁盘上的二进制文件中，文件名为dump.rdb。你可以配置Redis的持久化策略，例如数据集中每N秒钟有超过M次更新，就将数据写入磁盘；或者你可以手工调用命令SAVE或BGSAVE。  
　　工作原理  
　　． Redis forks.  
　　． 子进程开始将数据写到临时RDB文件中。  
　　． 当子进程完成写RDB文件，用新文件替换老文件。  
　　． 这种方式可以使Redis使用copy-on-write技术。  
（2）AOF  
　　快照模式并不十分健壮，当系统停止，或者无意中Redis被kill掉，最后写入Redis的数据就会丢失。这对某些应用也许不是大问题，但对于要求高可靠性的应用来说，  
　　Redis就不是一个合适的选择。  
　　Append-only文件模式是另一种选择。  
　　你可以在配置文件中打开AOF模式  
（3）虚拟内存方式  
　　当你的key很小而value很大时,使用VM的效果会比较好.因为这样节约的内存比较大.  
　　当你的key不小时,可以考虑使用一些非常方法将很大的key变成很大的value,比如你可以考虑将key,value组合成一个新的value.  
　　vm-max-threads这个参数,可以设置访问swap文件的线程数,设置最好不要超过机器的核数,如果设置为0,那么所有对swap文件的操作都是串行的.可能会造成比较长时间的延迟,但是对数据完整性有很好的保证.

## 3、解法对比及优缺点

一般解法：快速上手，简单使用

通用解法：命令窗口操作，需要一定的基础经验。

## 4、延伸及扩展问题回答参考

1. Memcache和Redis区别：

1、Redis中，并不是所有的数据都一直存储在内存中的，这是和Memcache相比一个最大的区别。

2、Redis在很多方面具备数据库的特征，或者说就是一个数据库系统，而Memcache只是简单的K/V缓存。

他们的扩展都需要做集群；实现方式：master-slave、Hash。

在100k以上的数据中，Memcache性能要高于Redis。

3、如果要说内存使用效率，使用简单的key-value存储的话，Memcached的内存利用率更高，而如果Redis采用hash结构来做key-value存储，由于其组合式的压缩，其内存利用率会高于Memcache。当然，这和你的应用场景和数据特性有关。

4、如果你对数据持久化和数据同步有所要求，那么推荐你选择Redis，因为这两个特性Memcache都不具备。即使你只是希望在升级或者重启系统后缓存数据不会丢失，选择Redis也是明智的。

5、Redis和Memcache在写入性能上面差别不大，读取性能上面尤其是批量读取性能上面Memcache更强

## 5、项目中体现经验的点

产品秒杀

## 6、论坛参考

https://www.cnblogs.com/xiaoxi/p/7007695.html