## get/post区别

答：GET在浏览器回退时是无害的，而POST会再次提交请求。

GET产生的URL地址可以被Bookmark，而POST不可以。

GET请求会被浏览器主动cache，而POST不会，除非手动设置。

GET请求只能进行url编码，而POST支持多种编码方式。

GET请求参数会被完整保留在浏览器历史记录里，而POST中的参数不会被保留。

GET请求在URL中传送的参数是有长度限制的，而POST么有。

对参数的数据类型，GET只接受ASCII字符，而POST没有限制。

GET比POST更不安全，因为参数直接暴露在URL上，所以不能用来传递敏感信息。

GET参数通过URL传递，POST放在Request body中

## 分页展示

1. 对查询到的数据而言，可以一次性查出，客户端分页，这是所谓的伪分页，适合数据量较少的分页查询，也可以动态获取数据，根据请丢，动态查询数据库的数据，立即显示到客户端，确保实时性和数据真实性，这才是正儿八经的服务器端分页，适合大量数据分页场景。
2. 分页方式，很多框架，插件都带有分页功能，实际工作中根据需要选择合适的插件分页即可，比如DataTables, Mybatis分页插件PageHelper

## Final

答：特性

final关键字可以用于成员变量、本地变量、方法以及类。

final成员变量必须在声明的时候初始化或者在构造器中初始化，否则就会报编译错误。

你不能够对final变量再次赋值。

本地变量必须在声明时赋值。

在匿名类中所有变量都必须是final变量。

final方法不能被重写。

final类不能被继承。

final关键字不同于finally关键字，后者用于异常处理。

final关键字容易与finalize()方法搞混，后者是在Object类中定义的方法，是在垃圾回收之前被JVM调用的方法。

接口中声明的所有变量本身是final的。

final和abstract这两个关键字是反相关的，final类就不可能是abstract的。

final方法在编译阶段绑定，称为静态绑定(static binding)。

没有在声明时初始化final变量的称为空白final变量(blank final variable)，它们必须在构造器中初始化，或者调用this()初始化。不这么做的话，编译器会报错“final变量(变量名)需要进行初始化”。

将类、方法、变量声明为final能够提高性能，这样JVM就有机会进行估计，然后优化。

按照Java代码惯例，final变量就是常量，而且通常常量名要大写：

优点

final关键字提高了性能。JVM和Java应用都会缓存final变量。

final变量可以安全的在多线程环境下进行共享，而不需要额外的同步开销。

使用final关键字，JVM会对方法、变量及类进行优化。

## 访问作用域

答：公共权限 public

public可以修饰类，数据成员，构造方法，方法成员。被public修饰的成员 ，可以在任何一个类中被调用，不管同包或不同包，是权限最大的一个修饰符。 它往往用于对外的情况，也就是对象或类对外的一种接口的形式。

受保护权限protected

protected可以修饰数据成员，构造方法，方法成员，不能修饰类（此处指外部类，不考虑内部类）。被protected修饰的成员，能在定义它们的类中，同包的类中被调用。如果有不同包的类想调用它们，那么这个类必须是定义它们的类的子类。

默认权限 (default)

类，数据成员，构造方法，方法成员，都能够使用默认权限，即不写任何关键字。默认权限即同包权限，同包权限的元素只能在定义它们的类中，以及同包的类中被调用。

私有权限 private

private可以修饰数据成员，构造方法，方法成员，不能修饰类（此处指外部类，不考虑内部类）。被private修饰的成员，只能在定义它们的类中使用，在其他类中不能调用。

使用注意

1．并不是每个关键字都可以修饰类（指外部类），只有public和default可 以。

2．所有关键字都可以修饰数据成员，方法成员，构造方法。

3．为了代码安全起见，关键字不要尽量使用权限大的，而是适用即可。比如 ，数据成员，如果没有特殊需要，尽可能用private。

| 作用域 | 类内部 | 本包 | 子类 | 外部包 |
| --- | --- | --- | --- | --- |
| public | √ | √ | √ | √ |
| protected | √ | √ | √ | × |
| （default） | √ | √ | × | × |
| private | √ | × | × | × |

4．关键字修饰的是“被访问”的权限

## 多态的特点

答：多态的定义：指允许不同类的对象对同一消息做出响应。即同一消息可以根据发送对象的不同而采用多种不同的行为方式。（发送消息就是函数调用）

多态存在的三个必要条件  
一、要有继承；  
二、要有重写；  
三、父类引用指向子类对象。

多态的作用：消除类型之间的耦合关系。

多态的好处：

1.可替换性（substitutability）。多态对已存在代码具有可替换性。例如，多态对圆Circle类工作，对其他任何圆形几何体，如圆环，也同样工作。

2.可扩充性（extensibility）。多态对代码具有可扩充性。增加新的子类不影响已存在类的多态性、继承性，以及其他特性的运行和操作。实际上新加子类更容易获得多态功能。例如，在实现了圆锥、半圆锥以及半球体的多态基础上，很容易增添球体类的多态性。

3.接口性（interface-ability）。多态是超类通过方法签名，向子类提供了一个共同接口，由子类来完善或者覆盖它而实现的。如图8.3 所示。图中超类Shape规定了两个实现多态的接口方法，computeArea()以及computeVolume()。子类，如Circle和Sphere为了实现多态，完善或者覆盖这两个接口方法。

4.灵活性（flexibility）。它在应用中体现了灵活多样的操作，提高了使用效率。

5.简化性（simplicity）。多态简化对应用软件的代码编写和修改过程，尤其在处理大量对象的运算和操作时，这个特点尤为突出和重要。

## 封装的作用

答：1．隔离性：

被封装后的对象（这里的对象是泛指代码的编程单元，一般指：程序集，命名空间，类，方法，属性，变量等）其外部对象是无法直接访问对象的内部实现细节，内部实现细节的的改动不会影响到外部对象的访问原则（即：对象内部修改后，在公开暴露指定的访问路径不变的情况下，外部访问它的对象是无需修改的），这是隔离性的体现，同时也是实现高内聚，低耦合的最根本的思想之一；

2．可复用性：

被封装后的对象可以被外部多个对象访问，而无需为每个外部对象去指定不同的服务对象；如：所有的对象的基类都是object类，object类里面的公共成员可以被其所有子类使用，Ado.Net相关的数据访问类及其公共成员均可被其它所有的对象使用等。

3．可读性：

被封装后的对象的名称（如：程序集名，类名，方法名）如果命名恰当，那么就能在不看里面的实现细节的前提下，了解该对象的作用；如：DataTable就是用来装表格数据的；ToString就是转换为字符串,Length就是指长度等。

## abstract使用场景和特点

场景：描述一类事物的时候，发现该事物确实存在着某种行为，但是目前该行为是不具体的，那么这时候我们应该抽取该方法的声明，不去实现该方法，这时候我们应该使用抽象类

特点：  
1、抽象类和抽象方法都需要用abstract修饰

2、抽象方法必须在抽象类中，抽象类中可以没有抽象方法

3、抽象类不能直接创造对象（实例化）

4、抽象类的子类如果重写所有抽象方法，那么就是一个具体的类

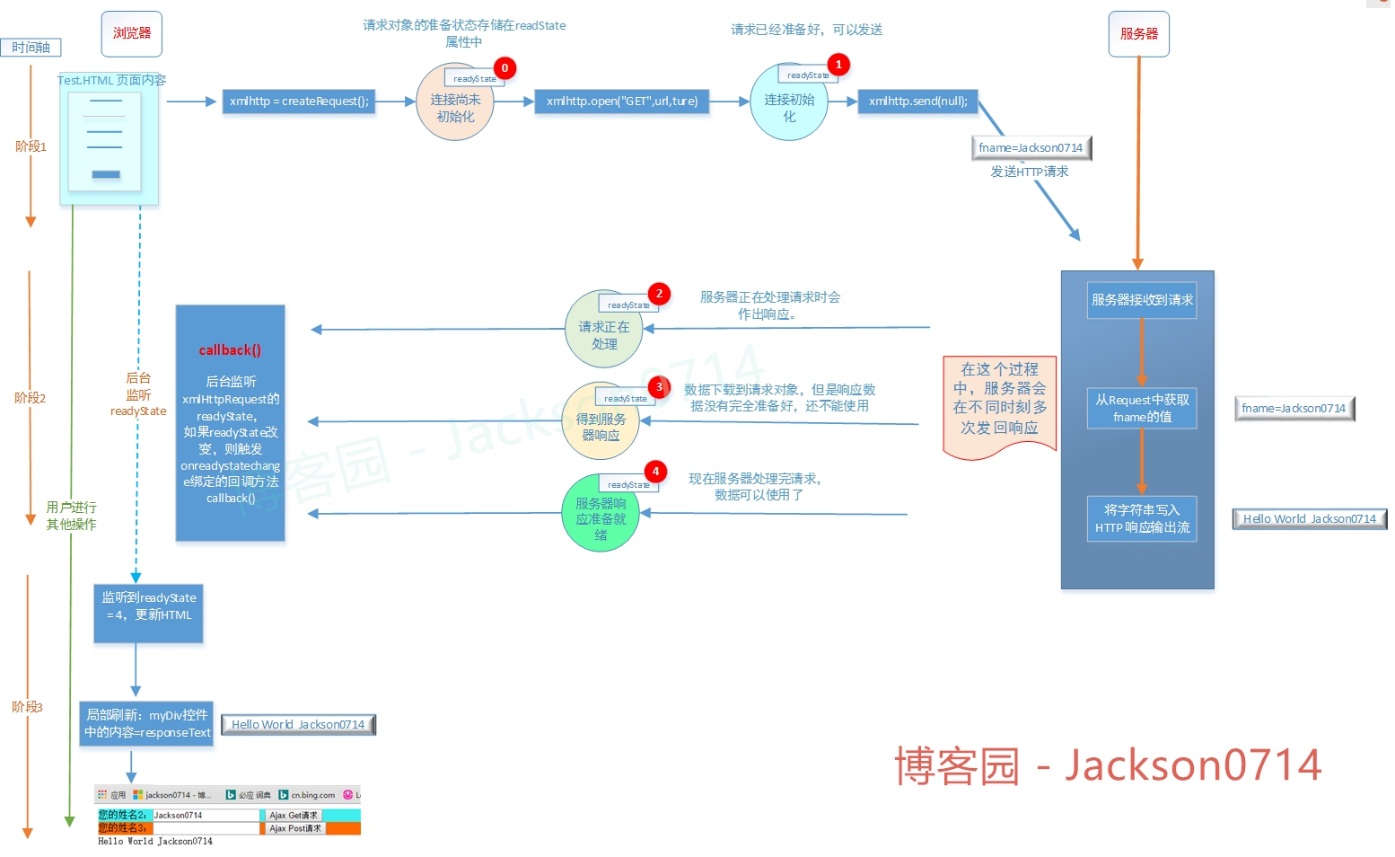
5、抽象类的子类不重写所有抽象方法，那么子类还是一个抽象类

6、抽象类有构造方法，目的是让子类来调用构造方法初始化

## Ajax底层

Ajax的底层原理简单来说通过XmlHttpRequest对象来向服务器发异步请求，从服务器获得数据，然后用javascript来操作DOM而更新页面。这其中最关键的一步就是从服务器获得请求数据。要清楚这个过程和原理，我们必须对 XMLHttpRequest有所了解。

XMLHttpRequest是ajax的核心机制，它是在IE5中首先引入的，是一种支持异步请求的技术。简单的说，也就是javascript可以及时向服务器提出请求和处理响应，而不阻塞用户。达到无刷新的效果。



## 耦合度

一、什么是耦合度

软件设计中通常用耦合度和内聚度作为衡量模块独立程度的标准。划分摸块的一个准则就是高内聚低耦合。 耦合度(Coupling)是对模块间关联程度的度量。耦合的强弱取决与模块间接口的复杂性、调用模块的方式以及通过界面传送数据的多少。 模块间的耦合度是指模块之间的依赖关系，包括控制关系、调用关系、数据传递关系。模块间联系越多，其耦合性越强，同时表明其独立性越差。降低模块间的耦合度能减少模块间的影响，防止对某一模块修改所引起的“牵一发动全身”的水波效应，保证系统设计顺利进行。 内聚和耦合密切相关，同其它模块存在强耦合关系的模块常意味这弱内聚，强内聚常意味着弱耦合。

耦合度就是某模块（类）与其它模块（类）之间的关联、感知和依赖的程度，是衡量代码独立性的一个指标，也是软件工程设计

及编码质量评价的一个标准。耦合的强度依赖于以下几个因素：

（1）一个模块对另一个模块的调用；

（2）一个模块向另一个模块传递的数据量；

（3）一个模块施加到另一个模块的控制的多少；

（4）模块之间接口的复杂程度。

二、降低耦合度的方法

1、少使用类的继承，多用接口隐藏实现的细节。 java面向对象编程引入接口除了支持多态外， 隐藏实现细节也是其中一个目的。

2、模块的功能化分尽可能的单一，道理也很简单，功能单一的模块供其它模块调用的机会就少。（其实这是高内聚的一种说法，高内聚低

3、遵循一个定义只在一个地方出现。

4、少使用全局变量。

5、类属性和方法的声明少用public，多用private关键字，

6、多用设计模式，比如采用MVC的设计模式就可以降低界面与业务逻辑的耦合度。

7、尽量不用“硬编码”的方式写程序，同时也尽量避免直接用SQL语句操作数据库。

8、最后当然就是避免直接操作或调用其它模块或类（内容耦合）；如果模块间必须存在耦合，原则上尽量使用数据耦合，少用控制耦合。

## 方法重载、方法重写分别解释以及二者区别

答：Overload是重载的意思，Override是覆盖的意思，也就是重写。

重载Overload表示同一个类中可以有多个名称相同的方法，但这些方法的参数列表各不相同（即参数个数或类型不同）。

重写Override表示子类中的方法可以与父类中的某个方法的名称和参数完全相同，通过子类创建的实例对象调用这个方法时，将调用子类中的定义方法，这相当于把父类中定义的那个完全相同的方法给覆盖了，这也是面向对象编程的多态性的一种表现。子类覆盖父类的方法时，只能比父类抛出更少的异常，或者是抛出父类抛出的异常的子异常，因为子类可以解决父类的一些问题，不能比父类有更多的问题。子类方法的访问权限只能比父类的更大，不能更小。如果父类的方法是private类型，那么，子类则不存在覆盖的限制，相当于子类中增加了一个全新的方法。

**方法重载、方法重写分别解释以及二者区别**：

方法重载是让类以统一的方式处理不同类型数据的一种手段

方法重写：即方法覆盖。指的是父子类中方法的关系



## 数据库性能优化

1.表的设计合理化(符合3NF)

2.添加适当索引(index) [四种: 普通索引、主键索引、唯一索引unique、全文索引]

3.分表技术(水平分割、垂直分割)

4.读写[写: update/delete/add]分离

5.存储过程 [模块化编程，可以提高速度]

6.对mysql配置优化 [配置最大并发数my.ini, 调整缓存大小 ]

7.mysql服务器硬件升级

8.定时的去清除不需要的数据,定时进行碎片整理

## 线程同步场景和解决办法

线程同步其核心就在于一个“同”。所谓“同”就是协同、协助、配合，“同步”就是协同步调，也就是按照预定的先后顺序进行运行，即“你先，我等， 你做完，我再做”

解决办法

1同步代码块

2同步方法

3使用同步锁

## Jsp Servlet交互流程

答：JSP是Servlet的一种特殊形式，每个JSP页面就是一个Servlet实例——JSP页面由系统编译成Servlet，Servlet再负责响应用户请求。JSP其实也是Servlet的一种简化，使用JSP时，其实还是使用Servlet，因为Web应用中的每个JSP页面都会由Servlet容器生成对应的Servlet。对于Tomcat而言，JSP页面生成的Servlet放在work路径对应的Web应用下。

## 表单提交方式

### 无刷新页面提交表单

表单可实现无刷新页面提交，无需页面跳转，如下，通过一个隐藏的iframe实现，form表单的target设置为iframe的name名称，  
form提交目标位当前页面iframe则不会刷新页面

### 通过type=submit提交

一般表单提交通过type=submit实现，input type="submit",浏览器显示为button按钮，通过点击这个按钮提交表单数据跳转到/url.do

### js提交form表单

js事件触发表单提交，通过button、链接等触发事件，js调用submit()方法提交表单数据，jquery通过submit()方法

### ajax异步提交表单数据

采用ajax异步方式，通过js获取form中所有input、select等组件的值，将这些值组成Json格式，通过异步的方式与服务器端进行交互，  
一般将表单数据传送给服务器端，服务器端处理数据并返回结果信息等

## MySQL多表连接查询可能碰到的问题

查询结果可能重复，效率低

## 集合框架及各自特点

答：集合主要有Collection和[Map接口](https://www.baidu.com/s?wd=Map%E6%8E%A5%E5%8F%A3&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)。

**Collection接口**

1、是List、Set和Queue接口的父接口；

2、定义了可用于操作List、Set和Queue的方法——增删改查；

**List接口及其实现类——ArrayList**

* List（序列），元素有序，并且可重复；
* List可以精确控制元素的插入位置，或删除指定位置的元素；
* ArrayList——数组序列，是List的一个重要实现类；
* ArrayList底层是由数组实现的

泛型集合中，不能添加泛型规定的类型及其子类型以外的对象，否则会报错。

泛型集合中的限定类型，不能使用基本类型，必须用基本类型的包装类。

**Set接口及其实现类——HashSet**

* Set（集），元素无序，并且不可以重复；
* HashSet——哈希集，是Set的一个重要实现类；

**Map接口**

* Map提供了一种映射关系，元素是以键值对（key-value）的形式存储的，能根据key快速查找value；
* Map中的键值对以Entry类型的对象实例形式存在；
* key值不能重复，value值可以重复；
* key对value是多（一）对一的关系；
* Map接口提供了返回key值集合、value值集合、Entry值集合，的方法；
* Map支持泛型，形式如：Map<K,V>

**HashMap类**

* HashMap是Map的一个重要实现类，也是最常用的，基于哈希表实现；
* HashMap中的Entry对象是无序排列的；
* Key值和value值都可以为null，但是一个HashMap只能有一个Key值为null的映射（Key值不可重复）；

List特点：元素有放入顺序，元素可重复   
Map特点：元素按键值对存储，无放入顺序   
Set特点：元素无放入顺序，元素不可重复（注意：元素虽然无放入顺序，但是元素在set中的位置是有该元素的HashCode决定的，其位置其实是固定的）   
List接口有三个实现类：LinkedList，ArrayList，Vector   
LinkedList：底层基于链表实现，链表内存是散乱的，每一个元素存储本身[内存地址](https://www.baidu.com/s?wd=%E5%86%85%E5%AD%98%E5%9C%B0%E5%9D%80&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)的同时还存储下一个元素的地址。链表增删快，查找慢   
ArrayList和Vector的区别：ArrayList是非[线程安全](https://www.baidu.com/s?wd=%E7%BA%BF%E7%A8%8B%E5%AE%89%E5%85%A8&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)的，效率高；Vector是基于[线程安全](https://www.baidu.com/s?wd=%E7%BA%BF%E7%A8%8B%E5%AE%89%E5%85%A8&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)的，效率低   
Set接口有两个实现类：HashSet(底层由HashMap实现)，LinkedHashSet   
SortedSet接口有一个实现类：TreeSet（底层由[平衡二叉树](https://www.baidu.com/s?wd=%E5%B9%B3%E8%A1%A1%E4%BA%8C%E5%8F%89%E6%A0%91&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)实现）   
Query接口有一个实现类：LinkList   
[Map接口](https://www.baidu.com/s?wd=Map%E6%8E%A5%E5%8F%A3&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)有三个实现类：HashMap，HashTable，LinkeHashMap   
 HashMap非[线程安全](https://www.baidu.com/s?wd=%E7%BA%BF%E7%A8%8B%E5%AE%89%E5%85%A8&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)，高效，支持null；HashTable线程安全，低效，不支持null   
SortedMap有一个实现类：TreeMap   
其实最主要的是，list是用来处理序列的，而set是用来处理集的。Map是知道的，存储的是键值对   
set 一般无序不重复.map kv 结构 list 有序 。

## xml解析

**XML的解析方式分为四种：1、DOM解析；2、SAX解析；3、JDOM解析；4、DOM4J解析。其中前两种属于基础方法，是官方提供的平台无关的解析方式；后两种属于扩展方法，它们是在基础的方法上扩展出来的，只适用于java平台。**

**DOM4J性能最好，连Sun的JAXM也在用DOM4J。目前许多开源项目中大量采用DOM4J，例如大名鼎鼎的Hibernate也用DOM4J来读取XML配置文件。如果不考虑可移植性，那就采用DOM4J。  
      JDOM和DOM在性能测试时表现不佳，在测试10M文档时内存溢出。在小文档情况下还值得考虑使用DOM和JDOM。虽然JDOM的开发者已经说明他们期望在正式发行版前专注性能问题，但是从性能观点来看，它确实没有值得推荐之处。另外，DOM仍是一个非常好的选择。DOM实现广泛应用于多种编程语言。它还是许多其它与XML相关的标准的基础，因为它正式获得W3C推荐（与基于非标准的Java模型相对），所以在某些类型的项目中可能也需要它（如在JavaScript中使用DOM）。**

## Json

答：JSON 指的是 JavaScript 对象表示法（JavaScript Object Notation）

JSON 是轻量级的文本数据交换格式

JSON 独立于语言：JSON 使用 Javascript语法来描述数据对象，但是 JSON 仍然独立于语言和平台。JSON 解析器和 JSON 库支持许多不同的编程语言。目前非常多的动态（PHP，JSP，.NET）编程语言都支持JSON。

JSON 具有自我描述性，更易理解

## 接口

答：1、接口是对外暴露的规则；

2、接口是程序的功能扩展；

3、接口的出现降低耦合度；

4、接口可以用来多实现；

5、类与接口之间是实现关系，而且类可以继承一个类的同时实现多个接口；

6、接口与接口直接可以有继承关系，且接口可以多继承；

## 构造方法特点

构造方法是一个类中特殊的成员方法  
在创建对象时，构造方法会自动被调用  
在父类和本类的构造方法相互调用用super（）和this（）且必须要放在第一行  
遵循[方法重载](https://www.baidu.com/s?wd=%E6%96%B9%E6%B3%95%E9%87%8D%E8%BD%BD&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)的规则  
调用该类之前 初始化数据  
构造方法名与类名相同  
 构造方法没有[返回类型](https://www.baidu.com/s?wd=%E8%BF%94%E5%9B%9E%E7%B1%BB%E5%9E%8B&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)，只有[访问权限](https://www.baidu.com/s?wd=%E8%AE%BF%E9%97%AE%E6%9D%83%E9%99%90&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)符加方法名  
 每个类都要有构造方法，如果没有，java虚拟机将提供一个默认空的构造方法，如果手动添加的构造方法 java虚拟机就不在提供空的构造方法  
其余普通方法不能以任何形式调用构造方法

## StringBuffer和String区别

答：String:

是对象不是原始类型.为不可变对象,一旦被创建,就不能修改它的值.  
对于已经存在的String对象的修改都是重新创建一个新的对象,然后把新的值保存进去.  
String 是final类,即不能被继承.

StringBuffer:  
是一个可变对象,当对他进行修改的时候不会像String那样重新建立对象它只能通过构造函数来建立,  
StringBuffer sb = new StringBuffer();  
对象被建立以后,在内存中就会分配内存空间,并初始保存一个null.通过它的append方法向其赋值.

sb.append("hello");

字符串连接操作中StringBuffer的效率要明显比String高:

String对象是不可变对象,每次操作Sting 都会重新建立新的对象来保存新的值.

StringBuffer对象实例化后，只对这一个对象操作。

## 异常的结构、检查型、运行时

答：java提供了两种异常机制。一种是运行时异常(RuntimeExepction)，一种是检查式异常(checked execption)。

检查式异常：我们经常遇到的IO异常及sql异常就属于检查式异常。对于这种异常，java编译器要求我们必须对出现的这些异常进行catch 所以 面对这种异常不管我们是否愿意，只能自己去写一堆catch来捕捉这些异常。

运行时异常：我们可以不处理。当出现这样的异常时，总是由虚拟机接管。比如：我们从来没有人去处理过NullPointerException异常，它就是运行时异常，并且这种异常还是最常见的异常之一。

## 值传递、引用传递

答：值传递是将变量的一个副本传递到方法中，方法中如何操作该变量副本，都不会改变原变量的值。

引用传递是将变量的内存地址传递给方法，方法操作变量时会找到保存在该地址的变量，对其进行操作。会对原变量造成影响。

## 双等号和equals区别

答：==比较的是两个对象的内存地址。 equals方法是String类重写object类的equals方法，会比较值是否相等（其他类与Object类的equals方法一致：比较地址）。

## 多线程及实现方式、线程同步

答：a.继承Thread:由于子类重写了Thread类的run()方法

b.实现Runnable接口，并实现其中run方法的方式。

利弊： 继承Thread代码简单，可是有局限性，如果某个类已经继承了其他父类，则此方法无法使用。 Runnable:代码比较复杂，不能直接使用Thread的方法，需要先获取线程对象后，才能得到Thread方法。

synchronized(同步)：Java提供了专业的解决方案，synchronized，让一个线程执行完毕后，另一线程才参与执行。

## 网页中各种状态码含义

答：HTTP状态码（英语：HTTP Status Code）是用以表示网页服务器HTTP响应状态的3位数字代码。

所有状态码的第一个数字代表了响应的五种状态之一。

1xx消息

这一类型的状态码，代表请求已被接受，需要继续处理。这类响应是临时响应，只包含状态行和某些可选的响应头信息，并以空行结束。由于HTTP/1.0协议中没有定义任何1xx状态码，所以除非在某些试验条件下，服务器禁止向此类客户端发送1xx响应。 这些状态码代表的响应都是信息性的，标示客户应该采取的其他行动。

100 Continue

客户端应当继续发送请求。这个临时响应是用来通知客户端它的部分请求已经被服务器接收，且仍未被拒绝。客户端应当继续发送请求的剩余部分，或者如果请求已经完成，忽略这个响应。服务器必须在请求完成后向客户端发送一个最终响应。

101 Switching Protocols

服务器已经理解了客户端的请求，并将通过Upgrade消息头通知客户端采用不同的协议来完成这个请求。在发送完这个响应最后的空行后，服务器将会切换到在Upgrade消息头中定义的那些协议。

只有在切换新的协议更有好处的时候才应该采取类似措施。例如，切换到新的HTTP版本比旧版本更有优势，或者切换到一个实时且同步的协议以传送利用此类特性的资源。

102 Processing

由WebDAV扩展的状态码，代表处理将被继续执行。

2xx成功

这一类型的状态码，代表请求已成功被服务器接收、理解、并接受。

200 OK

请求已成功，请求所希望的响应头或数据体将随此响应返回。

201 Created

请求已经被实现，而且有一个新的资源已经依据请求的需要而创建，且其URI已经随Location头信息返回。假如需要的资源无法及时创建的话，应当返回'202 Accepted'。

202 Accepted

服务器已接受请求，但尚未处理。正如它可能被拒绝一样，最终该请求可能会也可能不会被执行。在异步操作的场合下，没有比发送这个状态码更方便的做法了。

返回202状态码的响应的目的是允许服务器接受其他过程的请求（例如某个每天只执行一次的基于批处理的操作），而不必让客户端一直保持与服务器的连接直到批处理操作全部完成。在接受请求处理并返回202状态码的响应应当在返回的实体中包含一些指示处理当前状态的信息，以及指向处理状态监视器或状态预测的指针，以便用户能够估计操作是否已经完成。

203 Non-Authoritative Information

服务器已成功处理了请求，但返回的实体头部元信息不是在原始服务器上有效的确定集合，而是来自本地或者第三方的拷贝。当前的信息可能是原始版本的子集或者超集。例如，包含资源的元数据可能导致原始服务器知道元信息的超集。使用此状态码不是必须的，而且只有在响应不使用此状态码便会返回200 OK的情况下才是合适的。

204 No Content

服务器成功处理了请求，但不需要返回任何实体内容，并且希望返回更新了的元信息。响应可能通过实体头部的形式，返回新的或更新后的元信息。如果存在这些头部信息，则应当与所请求的变量相呼应。

如果客户端是浏览器的话，那么用户浏览器应保留发送了该请求的页面，而不产生任何文档视图上的变化，即使按照规范新的或更新后的元信息应当被应用到用户浏览器活动视图中的文档。

由于204响应被禁止包含任何消息体，因此它始终以消息头后的第一个空行结尾。

205 Reset Content

服务器成功处理了请求，且没有返回任何内容。但是与204响应不同，返回此状态码的响应要求请求者重置文档视图。该响应主要是被用于接受用户输入后，立即重置表单，以便用户能够轻松地开始另一次输入。

与204响应一样，该响应也被禁止包含任何消息体，且以消息头后的第一个空行结束。

206 Partial Content

服务器已经成功处理了部分GET请求。类似于FlashGet或者迅雷这类的HTTP 下载工具都是使用此类响应实现断点续传或者将一个大文档分解为多个下载段同时下载。

该请求必须包含Range头信息来指示客户端希望得到的内容范围，并且可能包含If-Range来作为请求条件。

响应必须包含如下的头部域：

Content-Range用以指示本次响应中返回的内容的范围；如果是Content-Type为multipart/byteranges的多段下载，则每一multipart段中都应包含Content-Range域用以指示本段的内容范围。假如响应中包含Content-Length，那么它的数值必须匹配它返回的内容范围的真实字节数。

Date

ETag和／或Content-Location，假如同样的请求本应该返回200响应。

Expires, Cache-Control，和／或Vary，假如其值可能与之前相同变量的其他响应对应的值不同的话。

假如本响应请求使用了If-Range强缓存验证，那么本次响应不应该包含其他实体头；假如本响应的请求使用了If-Range弱缓存验证，那么本次响应禁止包含其他实体头；这避免了缓存的实体内容和更新了的实体头信息之间的不一致。否则，本响应就应当包含所有本应该返回200响应中应当返回的所有实体头部域。

假如ETag或Last-Modified头部不能精确匹配的话，则客户端缓存应禁止将206响应返回的内容与之前任何缓存过的内容组合在一起。

任何不支持Range以及Content-Range头的缓存都禁止缓存206响应返回的内容。

207 Multi-Status

由WebDAV扩展的状态码，代表之后的消息体将是一个XML消息，并且可能依照之前子请求数量的不同，包含一系列独立的响应代码。

3xx重定向

这类状态码代表需要客户端采取进一步的操作才能完成请求。通常，这些状态码用来重定向，后续的请求地址（重定向目标）在本次响应的Location域中指明。

当且仅当后续的请求所使用的方法是GET或者HEAD时，用户浏览器才可以在没有用户介入的情况下自动提交所需要的后续请求。客户端应当自动监测无限循环重定向（例如：A→B→C→……→A或A→A），因为这会导致服务器和客户端大量不必要的资源消耗。按照HTTP/1.0版规范的建议，浏览器不应自动访问超过5次的重定向。

300 Multiple Choices

被请求的资源有一系列可供选择的回馈信息，每个都有自己特定的地址和浏览器驱动的商议信息。用户或浏览器能够自行选择一个首选的地址进行重定向。

除非这是一个HEAD请求，否则该响应应当包括一个资源特性及地址的列表的实体，以便用户或浏览器从中选择最合适的重定向地址。这个实体的格式由Content-Type定义的格式所决定。浏览器可能根据响应的格式以及浏览器自身能力，自动作出最合适的选择。当然，RFC 2616规范并没有规定这样的自动选择该如何进行。

如果服务器本身已经有了首选的回馈选择，那么在Location中应当指明这个回馈的URI；浏览器可能会将这个Location值作为自动重定向的地址。此外，除非额外指定，否则这个响应也是可缓存的。

301 Moved Permanently

被请求的资源已永久移动到新位置，并且将来任何对此资源的引用都应该使用本响应返回的若干个URI之一。如果可能，拥有链接编辑功能的客户端应当自动把请求的地址修改为从服务器反馈回来的地址。除非额外指定，否则这个响应也是可缓存的。

新的永久性的URI应当在响应的Location域中返回。除非这是一个HEAD请求，否则响应的实体中应当包含指向新的URI的超链接及简短说明。

如果这不是一个GET或者HEAD请求，因此浏览器禁止自动进行重定向，除非得到用户的确认，因为请求的条件可能因此发生变化。

注意：对于某些使用HTTP/1.0协议的浏览器，当它们发送的POST请求得到了一个301响应的话，接下来的重定向请求将会变成GET方式。

302 Found

请求的资源现在临时从不同的URI响应请求。由于这样的重定向是临时的，客户端应当继续向原有地址发送以后的请求。只有在Cache-Control或Expires中进行了指定的情况下，这个响应才是可缓存的。

新的临时性的URI应当在响应的Location域中返回。除非这是一个HEAD请求，否则响应的实体中应当包含指向新的URI的超链接及简短说明。

如果这不是一个GET或者HEAD请求，那么浏览器禁止自动进行重定向，除非得到用户的确认，因为请求的条件可能因此发生变化。

注意：虽然RFC 1945和RFC 2068规范不允许客户端在重定向时改变请求的方法，但是很多现存的浏览器将302响应视作为303响应，并且使用GET方式访问在Location中规定的URI，而无视原先请求的方法。状态码303和307被添加了进来，用以明确服务器期待客户端进行何种反应。

303 See Other

对应当前请求的响应可以在另一个URI上被找到，而且客户端应当采用GET的方式访问那个资源。这个方法的存在主要是为了允许由脚本激活的POST请求输出重定向到一个新的资源。这个新的URI不是原始资源的替代引用。同时，303响应禁止被缓存。当然，第二个请求（重定向）可能被缓存。

新的URI应当在响应的Location域中返回。除非这是一个HEAD请求，否则响应的实体中应当包含指向新的URI的超链接及简短说明。

注意：许多HTTP/1.1版以前的浏览器不能正确理解303状态。如果需要考虑与这些浏览器之间的互动，302状态码应该可以胜任，因为大多数的浏览器处理302响应时的方式恰恰就是上述规范要求客户端处理303响应时应当做的。

304 Not Modified

如果客户端发送了一个带条件的GET请求且该请求已被允许，而文档的内容（自上次访问以来或者根据请求的条件）并没有改变，则服务器应当返回这个状态码。304响应禁止包含消息体，因此始终以消息头后的第一个空行结尾。

该响应必须包含以下的头信息：

Date，除非这个服务器没有时钟。假如没有时钟的服务器也遵守这些规则，那么代理服务器以及客户端可以自行将Date字段添加到接收到的响应头中去（正如RFC 2068中规定的一样），缓存机制将会正常工作。

ETag和／或Content-Location，假如同样的请求本应返回200响应。

Expires, Cache-Control，和／或Vary，假如其值可能与之前相同变量的其他响应对应的值不同的话。

假如本响应请求使用了强缓存验证，那么本次响应不应该包含其他实体头；否则（例如，某个带条件的GET请求使用了弱缓存验证），本次响应禁止包含其他实体头；这避免了缓存了的实体内容和更新了的实体头信息之间的不一致。

假如某个304响应指明了当前某个实体没有缓存，那么缓存系统必须忽视这个响应，并且重复发送不包含限制条件的请求。

假如接收到一个要求更新某个缓存条目的304响应，那么缓存系统必须更新整个条目以反映所有在响应中被更新的字段的值。

305 Use Proxy

被请求的资源必须通过指定的代理才能被访问。Location域中将给出指定的代理所在的URI信息，接收者需要重复发送一个单独的请求，通过这个代理才能访问相应资源。只有原始服务器才能创建305响应。

注意：RFC 2068中没有明确305响应是为了重定向一个单独的请求，而且只能被原始服务器创建。忽视这些限制可能导致严重的安全后果。

306 Switch Proxy

在最新版的规范中，306状态码已经不再被使用。

307 Temporary Redirect

请求的资源现在临时从不同的URI响应请求。由于这样的重定向是临时的，客户端应当继续向原有地址发送以后的请求。只有在Cache-Control或Expires中进行了指定的情况下，这个响应才是可缓存的。

新的临时性的URI应当在响应的Location域中返回。除非这是一个HEAD请求，否则响应的实体中应当包含指向新的URI的超链接及简短说明。因为部分浏览器不能识别307响应，因此需要添加上述必要信息以便用户能够理解并向新的URI发出访问请求。

如果这不是一个GET或者HEAD请求，那么浏览器禁止自动进行重定向，除非得到用户的确认，因为请求的条件可能因此发生变化。

4xx客户端错误[编辑]

这类的状态码代表了客户端看起来可能发生了错误，妨碍了服务器的处理。除非响应的是一个HEAD请求，否则服务器就应该返回一个解释当前错误状况的实体，以及这是临时的还是永久性的状况。这些状态码适用于任何请求方法。浏览器应当向用户显示任何包含在此类错误响应中的实体内容。

如果错误发生时客户端正在传送数据，那么使用TCP的服务器实现应当仔细确保在关闭客户端与服务器之间的连接之前，客户端已经收到了包含错误信息的数据包。如果客户端在收到错误信息后继续向服务器发送数据，服务器的TCP栈将向客户端发送一个重置数据包，以清除该客户端所有还未识别的输入缓冲，以免这些数据被服务器上的应用程序读取并干扰后者。

400 Bad Request

由于包含语法错误，当前请求无法被服务器理解。除非进行修改，否则客户端不应该重复提交这个请求。

401 Unauthorized

当前请求需要用户验证。该响应必须包含一个适用于被请求资源的WWW-Authenticate信息头用以询问用户信息。客户端可以重复提交一个包含恰当的Authorization头信息的请求。如果当前请求已经包含了Authorization证书，那么401响应代表着服务器验证已经拒绝了那些证书。如果401响应包含了与前一个响应相同的身份验证询问，且浏览器已经至少尝试了一次验证，那么浏览器应当向用户展示响应中包含的实体信息，因为这个实体信息中可能包含了相关诊断信息。参见RFC 2617。

402 Payment Required

该状态码是为了将来可能的需求而预留的。

403 Forbidden

服务器已经理解请求，但是拒绝执行它。与401响应不同的是，身份验证并不能提供任何帮助，而且这个请求也不应该被重复提交。如果这不是一个HEAD请求，而且服务器希望能够讲清楚为何请求不能被执行，那么就应该在实体内描述拒绝的原因。当然服务器也可以返回一个404响应，假如它不希望让客户端获得任何信息。

404 Not Found

请求失败，请求所希望得到的资源未被在服务器上发现。没有信息能够告诉用户这个状况到底是暂时的还是永久的。假如服务器知道情况的话，应当使用410状态码来告知旧资源因为某些内部的配置机制问题，已经永久的不可用，而且没有任何可以跳转的地址。404这个状态码被广泛应用于当服务器不想揭示到底为何请求被拒绝或者没有其他适合的响应可用的情况下。

405 Method Not Allowed

请求行中指定的请求方法不能被用于请求相应的资源。该响应必须返回一个Allow头信息用以表示出当前资源能够接受的请求方法的列表。

鉴于PUT，DELETE方法会对服务器上的资源进行写操作，因而绝大部分的网页服务器都不支持或者在默认配置下不允许上述请求方法，对于此类请求均会返回405错误。

406 Not Acceptable

请求的资源的内容特性无法满足请求头中的条件，因而无法生成响应实体。

除非这是一个HEAD请求，否则该响应就应当返回一个包含可以让用户或者浏览器从中选择最合适的实体特性以及地址列表的实体。实体的格式由Content-Type头中定义的媒体类型决定。浏览器可以根据格式及自身能力自行作出最佳选择。但是，规范中并没有定义任何作出此类自动选择的标准。

407 Proxy Authentication Required

与401响应类似，只不过客户端必须在代理服务器上进行身份验证。代理服务器必须返回一个Proxy-Authenticate用以进行身份询问。客户端可以返回一个Proxy-Authorization信息头用以验证。参见RFC 2617。

408 Request Timeout

请求超时。客户端没有在服务器预备等待的时间内完成一个请求的发送。客户端可以随时再次提交这一请求而无需进行任何更改。

409 Conflict

由于和被请求的资源的当前状态之间存在冲突，请求无法完成。这个代码只允许用在这样的情况下才能被使用：用户被认为能够解决冲突，并且会重新提交新的请求。该响应应当包含足够的信息以便用户发现冲突的源头。

冲突通常发生于对PUT请求的处理中。例如，在采用版本检查的环境下，某次PUT提交的对特定资源的修改请求所附带的版本信息与之前的某个（第三方）请求向冲突，那么此时服务器就应该返回一个409错误，告知用户请求无法完成。此时，响应实体中很可能会包含两个冲突版本之间的差异比较，以便用户重新提交归并以后的新版本。

410 Gone

被请求的资源在服务器上已经不再可用，而且没有任何已知的转发地址。这样的状况应当被认为是永久性的。如果可能，拥有链接编辑功能的客户端应当在获得用户许可后删除所有指向这个地址的引用。如果服务器不知道或者无法确定这个状况是否是永久的，那么就应该使用404状态码。除非额外说明，否则这个响应是可缓存的。

410响应的目的主要是帮助网站管理员维护网站，通知用户该资源已经不再可用，并且服务器拥有者希望所有指向这个资源的远端连接也被删除。这类事件在限时、增值服务中很普遍。同样，410响应也被用于通知客户端在当前服务器站点上，原本属于某个个人的资源已经不再可用。当然，是否需要把所有永久不可用的资源标记为'410 Gone'，以及是否需要保持此标记多长时间，完全取决于服务器拥有者。

411 Length Required

服务器拒绝在没有定义Content-Length头的情况下接受请求。在添加了表明请求消息体长度的有效Content-Length头之后，客户端可以再次提交该请求。

412 Precondition Failed

服务器在验证在请求的头字段中给出先决条件时，没能满足其中的一个或多个。这个状态码允许客户端在获取资源时在请求的元信息（请求头字段数据）中设置先决条件，以此避免该请求方法被应用到其希望的内容以外的资源上。

413 Request Entity Too Large

服务器拒绝处理当前请求，因为该请求提交的实体数据大小超过了服务器愿意或者能够处理的范围。此种情况下，服务器可以关闭连接以免客户端继续发送此请求。

如果这个状况是临时的，服务器应当返回一个Retry-After的响应头，以告知客户端可以在多少时间以后重新尝试。

414 Request-URI Too Long

请求的URI长度超过了服务器能够解释的长度，因此服务器拒绝对该请求提供服务。这比较少见，通常的情况包括：

本应使用POST方法的表单提交变成了GET方法，导致查询字符串（Query String）过长。

重定向URI“黑洞”，例如每次重定向把旧的URI作为新的URI的一部分，导致在若干次重定向后URI超长。

客户端正在尝试利用某些服务器中存在的安全漏洞攻击服务器。这类服务器使用固定长度的缓冲读取或操作请求的URI，当GET后的参数超过某个数值后，可能会产生缓冲区溢出，导致任意代码被执行[1]。没有此类漏洞的服务器，应当返回414状态码。

415 Unsupported Media Type

对于当前请求的方法和所请求的资源，请求中提交的实体并不是服务器中所支持的格式，因此请求被拒绝。

416 Requested Range Not Satisfiable

如果请求中包含了Range请求头，并且Range中指定的任何数据范围都与当前资源的可用范围不重合，同时请求中又没有定义If-Range请求头，那么服务器就应当返回416状态码。

假如Range使用的是字节范围，那么这种情况就是指请求指定的所有数据范围的首字节位置都超过了当前资源的长度。服务器也应当在返回416状态码的同时，包含一个Content-Range实体头，用以指明当前资源的长度。这个响应也被禁止使用multipart/byteranges作为其Content-Type。

417 Expectation Failed

在请求头Expect中指定的预期内容无法被服务器满足，或者这个服务器是一个代理服务器，它有明显的证据证明在当前路由的下一个节点上，Expect的内容无法被满足。

418 I'm a teapot

本操作码是在1998年作为IETF的传统愚人节笑话, 在RFC 2324 超文本咖啡壶控制协议中定义的，并不需要在真实的HTTP服务器中定义。

421 There are too many connections from your internet address

从当前客户端所在的IP地址到服务器的连接数超过了服务器许可的最大范围。通常，这里的IP地址指的是从服务器上看到的客户端地址（比如用户的网关或者代理服务器地址）。在这种情况下，连接数的计算可能涉及到不止一个终端用户。

422 Unprocessable Entity

请求格式正确，但是由于含有语义错误，无法响应。

423 Locked

当前资源被锁定。

424 Failed Dependency

由于之前的某个请求发生的错误，导致当前请求失败，例如PROPPATCH。

425 Unordered Collection

在草案中定义，但是未出现在《WebDAV顺序集协议》中。

426 Upgrade Required

客户端应当切换到TLS/1.0。

449 Retry With

由微软扩展，代表请求应当在执行完适当的操作后进行重试。

5xx服务器错误[编辑]

这类状态码代表了服务器在处理请求的过程中有错误或者异常状态发生，也有可能是服务器意识到以当前的软硬件资源无法完成对请求的处理。除非这是一个HEAD请求，否则服务器应当包含一个解释当前错误状态以及这个状况是临时的还是永久的解释信息实体。浏览器应当向用户展示任何在当前响应中被包含的实体。

这些状态码适用于任何响应方法。

500 Internal Server Error

服务器遇到了一个未曾预料的状况，导致了它无法完成对请求的处理。一般来说，这个问题都会在服务器的程序码出错时出现。

501 Not Implemented

服务器不支持当前请求所需要的某个功能。当服务器无法识别请求的方法，并且无法支持其对任何资源的请求。

502 Bad Gateway

作为网关或者代理工作的服务器尝试执行请求时，从上游服务器接收到无效的响应。

503 Service Unavailable

由于临时的服务器维护或者过载，服务器当前无法处理请求。这个状况是临时的，并且将在一段时间以后恢复。如果能够预计延迟时间，那么响应中可以包含一个Retry-After头用以标明这个延迟时间。如果没有给出这个Retry-After信息，那么客户端应当以处理500响应的方式处理它。

504 Gateway Timeout

作为网关或者代理工作的服务器尝试执行请求时，未能及时从上游服务器（URI标识出的服务器，例如HTTP、FTP、LDAP）或者辅助服务器（例如DNS）收到响应。

注意：某些代理服务器在DNS查询超时时会返回400或者500错误。

505 HTTP Version Not Supported

服务器不支持，或者拒绝支持在请求中使用的HTTP版本。这暗示着服务器不能或不愿使用与客户端相同的版本。响应中应当包含一个描述了为何版本不被支持以及服务器支持哪些协议的实体。

506 Variant Also Negotiates

由《透明内容协商协议》扩展，代表服务器存在内部配置错误：被请求的协商变元资源被配置为在透明内容协商中使用自己，因此在一个协商处理中不是一个合适的重点。

507 Insufficient Storage

服务器无法存储完成请求所必须的内容。这个状况被认为是临时的。

509 Bandwidth Limit Exceeded

服务器达到带宽限制。这不是一个官方的状态码，但是仍被广泛使用。

510 Not Extended

获取资源所需要的策略并没有被满足。

## 拦截器（过滤器）作用

答：过滤器，是在java web中，你传入的request,response提前过滤掉一些信息，或者提前设置一些参数，然后再传入servlet或者struts的 action进行业务逻辑，比如过滤掉非法url（不是login.do的地址请求，如果用户没有登陆都过滤掉）,或者在传入servlet或者 struts的action前统一设置字符集，或者去除掉一些非法字符

拦截器，是在面向切面编程的就是在你的service或者一个方法，前调用一个方法，或者在方法后调用一个方法比如动态代理就是拦截器的简单实现，在你调用方法前打印出字符串（或者做其它业务逻辑的操作），也可以在你调用方法后打印出字符串，甚至在你抛出异常的时候做业务逻辑的操作。

拦截器与过滤器的区别 ：

拦截器是基于java的反射机制的，而过滤器是基于函数回调。

拦截器不依赖与servlet容器，过滤器依赖与servlet容器。

拦截器只能对action请求起作用，而过滤器则可以对几乎所有的请求起作用。

拦截器可以访问action上下文、值栈里的对象，而过滤器不能访问。

在action的生命周期中，拦截器可以多次被调用，而过滤器只能在容器初始化时被调用一次

执行顺序 ：过滤前 - 拦截前 - Action处理 - 拦截后 - 过滤后。

个人认为过滤是一个横向的过程，首先把客户端提交的内容进行过滤(例如未登录用户不能访问内部页面的处理)；过滤通过后，拦截器将检查用户提交数据的验证，做一些前期的数据处理，接着把处理后的数据发给对应的Action；Action处理完成返回后，拦截器还可以做其他过程(还没想到要做啥)，再向上返回到过滤器的后续操作

## 高并发处理

答：先来熟悉几个关于高并发的关键参数！

QPS:每秒处理的请求数量！

响应时间:处理一个请求需要的时间！

吞吐量:单位时间内的处理请求数量！

最大并发数:同一时间能支持的最大请求数！

一般来说有下面这些常规办法:

1，更多的静态资源:将代码中的大量枚举（容器加载时写入map，放入本地缓存），数据库中的定义表（定时任务放入缓存），固定配置，HTML文件等静态化处理，缓存起来！

2，图片服务器:一般来说，图片在一个页面上属于数据量比较大的东西，尽量避免动态数据和图片的顺序渲染，使用图片服务器分离数据和图片！

3，优化代码:尽量避免多层循环，避免多次访问数据库，使用多线程提高cpu使用率和执行速度，使用java8的流式处理和并行处理提高速度！

4，数据库:采用分库分表，mysql5.7之后，据说可以支持秒级百万级数据查询。速度相当之快，使用八库1024表，可以满足数据库一秒数百万的并发！同时可以开启缓存，写入存储过程等加快访问时间！分库分表之后还可以根据分表字段使用联合查询，避免了大多数数据库架构分布式之后不能联合查询的缺点！

5，使用内存型数据存储:使用redis等内存缓存可以提高读写速度，在数据落库之前快速读写数据，使用mongodb等作为大字段，多字节的key value保存方式，防止关系型数据库的不足！

6，负载均衡:使用nginx等负载均衡中间件，将请求分布到不同的机器上，避免单个应用持续的处理引起血崩！

## 数据库优化

1、单库数据库

一个初建的网站往往用户群都是很小的，最简单的网站架构就能解决实际的用户需求，当然为了保证网站的稳定性和安全性，我们会把网站的应用部署到至少两台机器上，后台的存储使用数据库，如果经济实力允许，数据库使用单台服务器部署

2、数据库读写分离

一个数据库主要负责写操作我们称之为主库，一个数据库专门负责读操作我们称之为副库，副库的数据都是从主库导入的，数据库的读写分离可以有效的保证关键数据的安全性，但是有个缺点就是当用户浏览数据时候，读的数据都会有点延时，这种延时比起全站不可用那肯定是可以接受的。

3、缓存技术

缓存主要是适用于读操作，并且缓存的读操作的效率要远远高于从数据库以及硬盘读取数据的效率。

4、数据库拆分

## 数据库三大范式

第一范式：当关系模式R的所有属性都不能在分解为更基本的数据单位时，称R是满足第一范式的，简记为1NF。满足第一范式是关系模式规范化的最低要

求，否则，将有很多基本操作在这样的关系模式中实现不了。

第二范式：如果关系模式R满足第一范式，并且R得所有非主属性都完全依赖于R的每一个候选关键属性，称R满足第二范式，简记为2NF。

第三范式：设R是一个满足第一范式条件的关系模式，X是R的任意属性集，如果X非传递依赖于R的任意一个候选关键字，称R满足第三范式，简记为3NF.

## 事务的作用和特点

答：事务：

　　事务是指作为单个逻辑工作单元执行的一组相关操作。

　　这些操作要求全部完成或者全部不完成。

使用事务的原因：保证数据的安全有效。

事务的四个特点：（ACID）

　　1、原子性（Atomic）：事务中所有数据的修改，要么全部执行，要么全部不执行。

　　2、一致性（Consistence）：事务完成时，要使所有所有的数据都保持一致的状态，换言之：通过事务进行的所有数据修改，必须在所有相关的表中得到反映。

　　3、隔离性（Isolation）：事务应该在另一个事务对数据的修改前或者修改后进行访问。

4、持久性（Durability）：保证事务对数据库的修改是持久有效的，即使发生系统故障，也不应该丢失。

## I18N实现方式（大型项目可能用到）

解决i18n的问题有三种经典思路，第一个是为每个页面提供每种语言的相关页面。第二种是把内容从表现形式中分离出来，做不同语言的内容文件。第三种是动态翻译页面内容。第三种很少见，而且机器翻译技术还很难达到人们的预期。第2种就是我们常用的properties

使用SpringMVC实现：

1、实现原理

　　默认情况下，SpringMVC 根据 Accept-Language 参数判断客户端的本地化类型。当接受到请求时，SpringMVC 会在上下文中查找一个本地化解析器(LocalResolver)，找到后使用它获取请求所对应的本地化类型信息。 springMVC 还允许装配一个动态更改本地化类型的拦截器，这样通过指定一个请求参数就可以控制单个请求的本地化类型。

　　2、工程结构

　　src下存放着中文和英文、默认语言的资源文件，显示的效果会根据Jsp页面的key值从国际豪华资源文件中寻找匹配的value值。

　　springDispatcherServlet-servlet.xml写入SpringMVC的相关配置。

　　3、目录详解

　　src下properties文件存放需要国际化的字段。以key-value的形式存在。 名称必须遵守 basename\_国家语言码.properties格式。如下：i18n\_zh\_CN.properties。

　　lib下存放有关于springMVC所依赖的jar包。

　　WebContent下pages 文件夹名称必须springDispatcherServlet-servlet.xml. 　 所配置的<property name="prefix" value="/WEB-INF/pages/"></property>一致。

　　下面两个jsp页面login负责展现中英文状态的切换，success负责展现切换页面中英文状态的保持。index.jsp负责初始语言的选择。

## 前端的PC端和移动端适配

答：目前针对跨终端的方案，主要分为两大阵营：一套资源Vs两套资源。

　　第一种是通过响应式或页面终端判断去实现一套资源适配所有终端；

　　第二种是通过终端判断分别调取两套资源以适配所有终端。

　　思路一:通过响应式或页面终端判断去实现一套资源适配所有终端

　　优势：只需维护一套资源，维护成本较低。

　　劣势：需加载适配各个终端的各个资源，在不同终端通过响应式布局实现不同展现，部分交互效果需要在页面中做终端判断，代价较大，若图片资源为一套，部分图片在超高分辨率设备（例如iphone系列）下会失真，且在非wifi情况下即使加了延时加载也易出现加载慢的情况。

　　技术选型：jquery（或原生js等）+ 响应式 + 前端模块加载器（seajs或RequireJS等）+ css预处理器（sass 或less等）。jquery较好的兼容性配合响应式可相对代价较小地实现跨终端。前端模块加载器主要负责按需加载，以提高页面加载速度，css预处理器 的变量、运算、嵌套等特性可大大提高手动计算响应式的效率，妈妈再也不用担心我把比例算错了。当然后两者可参考需求及成本决定是否采用。

　　思路二:通过终端判断分别调取两套资源以适配所有终端

　　优势：可根据不同端做个性设计及个性化信息推送且可按需加载，如移动端可配合重力感应、不同手势做各种炫酷拽效果，pc页面可不受流量限制做适合pc端的效果。

　　劣势：需维护两套资源，维护成本增加。

　　技术选型：zepto（或xui等移动端轻量级框架）+ 响应式 + 前端模块加载器 + css预处理器 + 终端适配。zepto作为jquery的移动端版本，依然延续其自身优势，大幅优化了移动端API且摒弃了兼容”非现代浏览器”的冗余代码，成为移动端轻 便可用的js框架代表，对于习惯了jquery的同学来说简直是不二之选！

终端适配目前一般通过ua判断来实现。ua判断可放在服务端也可放在页面中，在代理服务器中做跳转更快、更　　准确且不走应用程序层，即使浏览器禁用了js依然可以跳转到相应的地址，同时秉承着公共服务放在服务端这样的云端服务理念，我们选择了通过代理服务器做终端适配。

User-Agent嗅探，即Web浏览器发送一个Web页面或资源请求时，会发送一个User-Agent首部作为HTTP请求的一部分，那么我们就可以在服务器端获取想要的信息，进而判断并引导用户到达相应的页面地址

## JSP 9大内置对象

答：输入输出对象：out对象、response对象、request对象

通信控制对象：pageContext对象、session对象、application对象

Servlet对象:page对象、config对象

错误处理对象：exception对象

out对象：用于向客户端、浏览器输出数据。

request对象：封装了来自客户端、浏览器的各种信息。

response对象：封装了服务器的响应信息。

exception对象：封装了jsp程序执行过程中发生的异常和错误信息。

config对象：封装了应用程序的配置信息。

page对象：指向了当前jsp程序本身。

session对象：用来保存会话信息。也就是说，可以实现在同一用户的不同请求之间共享数

application对象：代表了当前应用程序的上下文。可以在不同的用户之间共享信息。

pageContext对象：提供了对jsp页面所有对象以及命名空间的访问。

## 存储过程

存储过程是一组予编译的SQL语句

它的优点：

1.允许模块化程序设计，就是说只需要创建一次过程，以后在程序中就可以调用该过程任意次。

2.允许更快执行，如果某操作需要执行大量SQL语句或重复执行，存储过程比SQL语句执行的要快。

3.减少网络流量，例如一个需要数百行的SQL代码的操作有一条执行语句完成，不需要在网络中发送数百行代码。

4.更好的安全机制，对于没有权限执行存储过程的用户，也可授权他们执行存储过程。

## Tamcat集群以及session共享的问题

答：对于web应用集群的技术实现而言，最大的难点就是如何能在集群中的多个节点之间保持数据的一致性，而session是这些数据中非常重要的一块，要实现这一点，答题有两种方式：**（方式有很多，下面只是很普通的两种，其他方式自己去了解）**

1. 把一台服务器作为session服务器，集群中的所有节点都来访问这个session服务器来获取数据

2. 就是集群中所有的节点之间都进行session数据的同步拷贝，任何一个节点均保存所有session数据。

## Hibernate和mybatis区别，现实中应该使用哪一个

答：1. mybatis是通过mapper.xml维护映射结果，而Hibernate是通过高度封装的HQL映射

2. hibernate的hql语句使得数据库的移植性更好，而hibernate在级联删除的时候小路更低数据量大

3. 具体在现实中应该使用哪一个，这个因项目而言，因为两者都是ORM框架，都能起到相同的作用，一般项目大建议用Hibernate,小项目可以用Mybatis

## SpringMVC工作原理

1、 用户发送请求至前端控制器DispatcherServlet。

2、 DispatcherServlet收到请求调用HandlerMapping处理器映射器。

3、 处理器映射器找到具体的处理器(可以根据xml配置、注解进行查找)，生成处理器对象及处理器拦截器(如果有则生成)一并返回给DispatcherServlet。

4、 DispatcherServlet调用HandlerAdapter处理器适配器。

5、 HandlerAdapter经过适配调用具体的处理器(Controller，也叫后端控制器)。

6、 Controller执行完成返回ModelAndView。

7、 HandlerAdapter将controller执行结果ModelAndView返回给DispatcherServlet。

8、 DispatcherServlet将ModelAndView传给ViewReslover视图解析器。

9、 ViewReslover解析后返回具体View。

10、DispatcherServlet根据View进行渲染视图（即将模型数据填充至视图中）。

11、 DispatcherServlet响应用户。

1. 用户发请求-->DispatcherServlet，前端控制器收到请求后自己不进行处理，而是委托给其他的解析器进行处理，作为统一访问点，进行全局的流程控制。

2.DispatcherServlet-->HandlerMapping，HandlerMapping将会把请求映射为HandlerExecutionChain对象（包含一个Handler处理器,多个HandlerInterceptor拦截器)。

3.DispatcherServlet-->HandlerAdapter,HandlerAdapter将会把处理器包装为适配器，从而支持多种类型的处理器。

4.HandlerAdapter-->处理器功能处理方法的调用，HandlerAdapter将会根据适配的结果调用真正的处理器的功能处理方法，完成功能处理，并返回一个ModelAndView对象(包含模型数据，逻辑视图名)

5.ModelAndView的逻辑视图名-->ViewResolver，ViewResoler将把逻辑视图名解析为具体的View。

6.View-->渲染，View会根据传进来的Model模型数据进行渲染，此处的Model实际是一个Map数据结构

7.返回控制权给DispatcherServlet，由DispatcherServlet返回响应给用户。

## Springaop工作原理

用动态代理技术，利用截取消息的方式，对该消息进行装饰，以取代原有对象行为的执行；同时采用静态织入的方式，引入特定的语法创建“切面”，从而使得编译器可以在编译期间织入有关“切面”的代码。

## Spring事务管理有哪两种方式

spring支持编程式事务管理和声明式事务管理两种方式。

1.编程式事务管理使用TransactionTemplate或者直接使用底层的PlatformTransactionManager。对于编程式事务管理，spring推荐使用TransactionTemplate。

2.声明式事务管理建立在AOP之上的。其本质是对方法前后进行拦截，然后在目标方法开始之前创建或者加入一个事务，在执行完目标方法之后根据执行情况提交或者回滚事务。声明式事务最大的优点就是不需要通过编程的方式管理事务，这样就不需要在业务逻辑代码中掺杂事务管理的代码，只需在配置文件中做相关的事务规则声明

显然声明式事务管理要优于编程式事务管理，这正是spring倡导的非侵入式的开发方式,能够很好的降低程序的耦合性

## Hibernate延迟加载的了解

延迟加载，也叫懒加载，它是Hibernate为提高程序执行效率而提供的一种机制，即只有真正使用该对象的数据时才会创建。

Hibernate中主要是通过代理（proxy）机制来实现延迟加载。它的具体过程：Hibernate丛数据库获取某一个对象数据时、获取某一个对象的集合属性值时，或获取某一个对象所关联的另一个对象时，由于没有使用该对象的数据，hibernate并不是数据库加载真正的数据，而只是为该对象创建一个代理对象来代表这个对象，这个对象上的所有属性都是默认值；只有在真正需要使用该对象的数据时才创建这个真实对象，真正从数据库中加载它的数据，这样在某些情况下，就可以提高查询效率

## Memcache和Redis的比较

1. Redis和Memcache都是将数据存放在内存中，都是内存数据库。不过memcache还可用于缓存其他东西，例如图片、视频等等；

2. Redis不仅仅支持简单的k/v类型的数据，同时还提供list，set，hash等数据结构的存储；

3. 虚拟内存--Redis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挂掉后，数据没了；redis可以定期保存到磁盘（持久化）；

7. 灾难恢复--memcache挂掉后，数据不可恢复; redis数据丢失后可以通过aof恢复；

8. Redis支持数据的备份，即master-slave模式的数据备份；

9. 应用场景不一样：Redis出来作为NoSQL数据库使用外，还能用做消息队列. 数据堆栈和数据缓存等；Memcached适合于缓存SQL语句. 数据集. 用户临时性数据. 延迟查询数据和session等。

## Git和svn的区别

答：1）Git是分布式的，SVN不是：

这 是GIT和其它非分布式的版本控制系 统，例如SVN，CVS等，最核心的区别。好处是跟其他同事不会有太多的冲突，自己写的代码放在自己电脑上，一段时间后再提交、合并，也可以不用联网在本地提交；GIT并不是目前第一个或唯一的分布式版本控制系统。还有一些系统，例如Bitkeeper, Mercurial等，也是运行在分布式模式上的。但GIT在这方面做的更好，而且有更多强大的功能特征。

GIT跟SVN一样有自己的 集中式版本库或服务器。但，GIT更倾向于被使用于分布式模式，也就是每个开发人员从中心版本库/服务器上chect out代码后会在自己的机器上克隆一个自己的版本库。可以这样说，如果你被困在一个不能连接网络的地方时，你仍然能够提交文件，查看历史版本记录，创建项 目分支等。

2）GIT把内容按元数据方式存储，而SVN是按文件：

所有的资源控 制系统都是把文件的元信息隐藏在一个类似.svn,.cvs等的文件夹里。如果你把.git目录的体积大小跟.svn比较，你会发现它们差距很大。因 为,.git目录是处于你的机器上的一个克隆版的版本库，它拥有中心版本库上所有的东西，例如标签，分支，版本记录等。

3）GIT分支和SVN的分支不同：

分支在SVN中一点不特别，就是版本库中的另外的一个目录。如果你想知道是否合并了一个分支，你需要手工运行像这样的命令svn propget svn:mergeinfo，来确认代码是否被合并。 然而，处理GIT的分支却是相当的简单和有趣。

Git鼓励分Branch，而SVN，说实话，我用Branch的次数还挺少的，SVN自带的Branch merge我还真没用过，有merge时用的是Beyond Compare工具合并后再Commit的；

4）GIT没有一个全局的版本号，而SVN有：

目前为止这是跟SVN相比GIT缺少的最大的一个特征。

5）GIT的内容完整性要优于SVN：

GIT的内容存储使用的是SHA-1哈希算法。这能确保代码内容的完整性，确保在遇到磁盘故障和网络问题时降低对版本库的破坏。

6）Git下载下来后，在本地不必联网就可以看到所有的log，很方便学习，SVN却需要联网；

7）SVN在Commit前，我们都建议是先Update一下，跟本地的代码编译没问题，并确保开发的功能正常后再提交，这样其实挺麻烦的，有好几次同事没有先Updata，就Commit了，发生了一些错误，耽误了大家时间，Git可能这种情况会少些。

其他区别：

1。速度：

克隆一份全新的目录，以同样拥有五个（才五个）分支来说，SVN是同时复製5个版本的文件，也就是说重复五次同样的动作。而Git只是获取文件的每个版本的 元素，然后只载入主要的分支（master）。在我的经验，克隆一个拥有将近一万个提交（commit），五个分支，每个分支有大约1500个文件的 SVN，耗了将近一个小时！而Git只用了区区的1分鐘！

2。版本库（repository）：

据我所知，SVN只能有一个指定中央版本库。当这个中央版本库有问题时，所有工作成员都一起瘫痪直到版本库维修完毕或者新的版本库设立完成。

而 Git可以有无限个版本库。或者，更正确的说法，每一个Git都是一个版本库，区别是它们是否拥有活跃目录（Git Working Tree）。如果主要版本库（例如：置於GitHub的版本库）发生了什麼事，工作成员仍然可以在自己的本地版本库（local repository）提交，等待主要版本库恢复即可。工作成员也可以提交到其他的版本库！

3.分支（Branch）

在SVN，分支是一个完整的目录。且这个目录拥有完整的实际文件。如果工作成员想要开啟新的分支，那将会影响“全世界”！每个人都会拥有和你一样的分支。如果你的分支是用来进行破坏工作（安检测试），那将会像传染病一样。

而 Git，每个工作成员可以任意在自己的本地版本库开啟无限个分支。举例：当我想尝试破坏自己的程序（安检测试），并且想保留这些被修改的文件供日后使用， 我可以开一个分支，做我喜欢的事。完全不需担心妨碍其他工作成员。只要我不合并及提交到主要版本库，没有一个工作成员会被影响。等到我不需要这个分支时， 我只要把它从我的本地版本库删除即可。无痛无痒。

Git的分支名是可以使用不同名字的。例如：我的本地分支名為testing，而在主要版本库的名字其实是master。 最值得一提，我可以在Git的任意一个提交点（commit point）开啟分支！（其中一个方法是使用gitk –all 可观察整个提交记录，然后在任意点开啟分支。）

4.提交（Commit）

在SVN，当你提交你的完成品时，它将直接记录到中央版本库。当你发现你的完成品存在严重问题时，你已经无法阻止事情的发生了。如果网路中断，你根本没办法提交！

而Git的提交完全属於本地版本库的活动。而你只需“推”（git push）到主要版本库即可。Git的“推”其实是在执行“同步”（Sync）。

5.重新设立起点（Rebase）

我没在SVN尝试过，不知道有没有这样的功能。

在 Git，如果你想把别人的最新提交设立為现在这个分支的起点，只要执行git rebase branch\_name 即可。这个和合并（merge）不同点是，merge会依据修改的时间视為最新，而Rebase会要求你去解决双方都有修改过的地方的矛盾 （conflict）。

6.系统档案

SVN会在每一个目录置放一个.svn。如果想移除这些.svn是很累的。

而Git会在目录起点拥有一个.git目录，以及.gitignore。

对我而言，管理一个Git 的版本库是很容易的事。

补充：

git优点

git每台电脑都有一个版本库，可以在本地做版本管理；

速度快。git的速度远超大部分版本管理系统，包括svn

强大的分支管理功能

活跃的开源社区，如最著名的github

git 缺点

Git 没有严格的权限管理控制，一般通过系统设置文件读写权限的方式来做权限控制。

工作目录只能是整个项目。比如 checkout，建分支，都是基于整个项目的。而 svn 可以基于项目中的某一个目录

## kafka和activtyMQ的区别，为什么选择kafka？

1）在架构模型方面

RabbitMQ遵循AMQP协议，RabbitMQ的broker由Exchange,Binding,queue组成，其中exchange和binding组成了消息的路由键；客户端Producer通过连接channel和server进行通信，Consumer从queue获取消息进行消费（长连接，queue有消息会推送到consumer端，consumer循环从输入流读取数据）。rabbitMQ以broker为中心；有消息的确认机制。

kafka遵从一般的MQ结构，producer，broker，consumer，以consumer为中心，消息的消费信息保存的客户端consumer上，consumer根据消费的点，从broker上批量pull数据；无消息确认机制。

2）在吞吐量

kafka具有高的吞吐量，内部采用消息的批量处理，zero-copy机制，数据的存储和获取是本地磁盘顺序批量操作，具有O(1)的复杂度，消息处理的效率很高。

rabbitMQ在吞吐量方面稍逊于kafka，他们的出发点不一样，rabbitMQ支持对消息的可靠的传递，支持事务，不支持批量的操作；基于存储的可靠性的要求存储可以采用内存或者硬盘。

3）在可用性方面

rabbitMQ支持miror的queue，主queue失效，miror queue接管。

kafka的broker支持主备模式。

4）在集群负载均衡方面，

kafka采用zookeeper对集群中的broker、consumer进行管理，可以注册topic到zookeeper上；通过zookeeper的协调机制，producer保存对应topic的broker信息，可以随机或者轮询发送到broker上；并且producer可以基于语义指定分片，消息发送到broker的某分片上。

rabbitMQ的负载均衡需要单独的loadbalancer进行支持。

## 抽象类、接口的区别

答：1、抽象类和接口都不能直接实例化，如果要实例化，抽象类变量必须指向实现所有抽象方法的子类对象，接口变量必须指向实现所有接口方法的类对象。

2、抽象类要被子类继承，接口要被类实现。

3、接口只能做方法申明，抽象类中可以做方法申明，也可以做方法实现

4、接口里定义的变量只能是公共的静态的常量，抽象类中的变量是普通变量。

5、抽象类里的抽象方法必须全部被子类所实现，如果子类不能全部实现父类抽象方法，那么该子类只能是抽象类。同样，一个实现接口的时候，如不能全部实现接口方法，那么该类也只能为抽象类。

6、抽象方法只能申明，不能实现，接口是设计的结果 ，抽象类是重构的结果

7、抽象类里可以没有抽象方法

8、如果一个类里有抽象方法，那么这个类只能是抽象类

9、抽象方法要被实现，所以不能是静态的，也不能是私有的。

10、接口可继承接口，并可多继承接口，但类只能单根继承。

## Static

答：1）static方法

　　static方法一般称作静态方法，由于静态方法不依赖于任何对象就可以进行访问，因此对于静态方法来说，是没有this的，因为它不依附于任何对象，既然都没有对象，就谈不上this了。并且由于这个特性，在静态方法中不能访问类的非静态成员变量和非静态成员方法，因为非静态成员方法/变量都是必须依赖具体的对象才能够被调用。

2）static变量

　　static变量也称作静态变量，静态变量和非静态变量的区别是：静态变量被所有的对象所共享，在内存中只有一个副本，它当且仅当在类初次加载时会被初始化。而非静态变量是对象所拥有的，在创建对象的时候被初始化，存在多个副本，各个对象拥有的副本互不影响。

　　static成员变量的初始化顺序按照定义的顺序进行初始化。

3）static代码块

　　static关键字还有一个比较关键的作用就是 用来形成静态代码块以优化程序性能。static块可以置于类中的任何地方，类中可以有多个static块。在类初次被加载的时候，会按照static块的顺序来执行每个static块，并且只会执行一次。