## 集合

Collection继承于Iterator迭代器，是一种设计模式，也是一个对象，它可以遍历序列中的对象，而开发人员不需要关注序列的底层结构。“轻量级”对象，创建它的代价很小，只能单向移动，获取下一个，判断是否还有下一个，删除等操作

### List：都有什么类型？

### ArrayList和LinkedList的区别：

底层数据结构：ArrayList：数组；LinkedList：双向链表

扩容：ArrayList是之前的1.5倍；LinkedList没有扩容没有初始大小

线程安全：都不安全，非同步

Vector 和ArrayList比较：Vector 多了一个增长系数，线程安全（对于单一操作），

### Queue: 队列

Queue:继承于Collection，先进先出;底层数据结构：数组；

POLL和remove方法的区别在于，对于空的队列返回不同，poll返回null，remove会抛出异常没有这种元素；peek和element一样

Deque继承于Queue, 是双端队列，依然是先进先出，可以从两端增加和删除，获取数据。

ArrayDeque继承于Deque，底层数据：数组；两个索引表head，tail；扩容：保证是2的幂方， Remove是通过poll实现的

Stack 和 Queue：

Stack : 继承于Vector；先进后出；底层数据结构：数组

## 2-Map

### HashMap：

1. 底层数据结构：数组+链表（1.7）；数组+链表+红黑树（1.8）；
2. 扩容：原来bucket的2倍；牵涉到rehash，复制数据，耗性能
3. （1.7）需要优化点：冲突严重时，链表越来越长，查询效率会变低，时间复杂度O（n）。(1.8）:优化后O（logn）查询效率提高；
4. 为什么是线程不安全？答案：在多线程环境下，存在多个元素进行put，如果hash相同，相同的key可能会添加到同一个位置，其中一个线程中的数据丢失，或者扩容时多个线程都在rehash，可能会导致一个线程成功，其他线程都失败，丢失数据，会出现死循环。
5. get的实现？put的实现？hash的实现？

Put：计算hashcode，得出index，未碰撞，放入bucket中，碰撞了，放入链表，如果链表过长，放入树中。

Get：通过key，一次命中返回值，或者从表中或者树中查找值

6，数组存储的是：链表，链表中的每个节点就是哈希表中的值

### ConcurrentHashMap：

1. 底层数据结构：数组+链表（1.7）；数组+链表+红黑树（1.8）；
2. 锁机制：（1.7）分段锁：ReentrantLock一个线程占用segment，不会影响到其他的segment；（1.8）CAS+Synchronized 保证线程并发安全性。
3. 核心数据和链表都是volatile（可见性，不能保证原子性）修饰，所以在put时依然需要加锁保证原子性

## 3-Jdk版本更新

## 4-String，StringBuffer，StringBuilder

String: 字符串变量：理论上不可变，因为里面的值被final修饰，也没有set方法；都是创建新的对象，即使是拼接或者修改，引用指向new 值；频繁的修改值，堆内存就会被分配的越多，内存不足时会影响GC工作

后两者：都是字符串变量，当值发生修改时不会产生新的对象，区别在于StringBuffer是线程安全的，里面的部分操作被syncronized修饰，

## 5-数组：初始化类型？如何分配内存？什么是引用对象，引用变量？

答：两种：静态初始化，动态初始化；系统在堆内存中分配一块连续的内存给数组；基础数据类型是存在内存中的栈区，引用数据类型是引用存在栈区，而引用指向的值是存放在堆中的。

## 6-什么是重写，覆盖，重载？三者的区别？

都是针对方法的。

重写，覆盖（Override）：返回值，方法名，参数的顺序，类型都要完全一致，里面的函数重新实现。父类与子类之间的多态性的体现。

重载：返回值，方法名完全一致，参数列表不同的函数实现。类内部多态性的表现；

## 7-extends，implements区别

1. 当抽象类中包含有非抽象方法，那么只可以extends，不可以implements

## 8-抽象类，普通类，接口？

抽象类，接口：不可以实例化，抽象方法和接口中的方法只有声明，没有方法体。子类需要实现抽象方法，接口中的方法，不然子类需要定义为抽象类。两者都没有变量（默认是public static final）。

抽象类，普通类：有构造函数，可以有普通方法

抽象方法只能存在于抽象类或者接口中，不可以存在于普通类。

## 9-BIO、NIO、AIO 有什么区别？

1. 流的种类？

流的流向：输入流，输出流；

操作单元：字节流，字符流；

1. BIO面向流，阻塞io，NIO面向缓冲区，不阻塞io，有选择器；
2. 概念的理解：

BIO：当一个线程进行读或者写的时候，这个线程是被阻塞的，直到请求结果返回。这个线程在此期间什么事情也不会去做。一个请求一个应答；

NIO：将这个任务分成多个小任务，由一个线程所有的io事件，并负责分发，利用事件驱动机制，事件到了再触发，例如：线程调用读/写，不需要等待读/写完成，这个线程可以同时去做别的事情。

## 10-多线程

### 10-1：并行与并发？

并发：当有多个线程在操作时，如果系统只有一个cpu，则它不可能同时进行一个以上的线程，只能把cpu划分成等量的时间片，分配给线程，在一个时间片一个线程进行，其他的线程处于等待状态。

并行：系统中有多个cpu，有可能并行，一个cpu执行一个线程，另一个cpu执行另一个线程。

### 10-2：守护线程，？

是用来服务于用户线程（普通线程），不需要上层逻辑介入。

当线程只剩下守护线程，jvm就会退出；如果还有任意一个用户线程，jvm就不会退出；不能去访问固定资源，因为他会在任意时间发生中断，典型例子：垃圾回收站;

### 10-3：thread中的run和start方法？

Run：一个普通方法的调用；

Start：会创建一个新的线程并启动

### 10-4：thread和runnable关系？

Thread实行了runnable，使得run支持多线程，由于Java的单继承，推荐多使用runnable

### 10-5：sleep和wait区别？

Thread的sleep只会让出cpu，不会导致锁行为的改变

Object的wait不仅会让出cpu，还会释放已经占用的同步资源锁

### 10-6：notify和notifyAll的区别？

notifyAll：唤醒所有处于等待池的线程全部进入锁池中去竞争获取锁的机会

notify：随机唤醒一个处于等待池中的线程进入锁池中去竞争获取锁的机会

### 10-7：如何处理线程的返回值？

1. 主线程等待法；
2. 使用Thred类的join（）阻塞当前线程去等待子线程处理完毕
3. 通过callable接口实现，通过FutureTask 或者线程池中获获取

### 10-8：线程的六个状态

1. 新建（new）：创建后未启动的线程的状态
2. 运行Runnable：running和ready
3. 无限期等待：不会被cpu分配事件片，需要显示的唤醒；（Object的wait；Thread的join();LockSupport.park()）
4. 限期等待：在一定时间内会被系统自动唤醒；例如Thread的sleep方法，Object的wait（时间），Thread的join（时间），LockSupport.parkNanos()….LockSuppport.parkUntil()
5. 阻塞：等待获取排它锁（写锁）
6. 结束：线程已终止，结束执行

### 10-9：Runnable和Callable区别？

Runnable：可以直接被thread执行，但是没有返回值

Callable：执行之后有返回值，需要用线程池来执行。

### 10-10：四种线程池？

Execuroe接口

线程池来源：当并发的情况多的时候，而每个线程请求时间很短，频繁的创建和销毁线程，会大大降低系统的效率。那么有没有一种方法可以在执行完一个任务，并不销毁，还可以继执行别的任务。

什么场景下使用线程池：单个任务处理时间短；需要的任务处理量大；

好处：降低资源消耗（通过重复利用已创建的线程来降低不断地创建销毁线程所带来的消耗）；提高响应速度（不需要等待创建线程）；提高线程的管理性（统一分配，调优，监控）。

newCachedThreadPool：可缓存线程池，如果线程池超过处理需要，可以灵活回收空闲线程，如果没有可回收的，就新建线程。

newFixedThreadPool：定长线程池，可以控制线程并发量，超出的要等待

newScheduledThreadPool：定长线程池，支持定时和周期性任务

newSingleThreadPool：单线程话线程池，

#### 10-10-1：五种线程池状态？

1. Running：可以接收新提交的任务，也能处理阻塞队列中的任务；
2. Shutdown：关闭，不再接受新的任务，却可以继续处理阻塞队列中任务;
3. Stop：不接受新的，也不处理队列中的任务，会中断正在处理的任务；调用shutdownNow
4. Tidying：所有的任务执行完，workerCount（有效线程数）变为0，
5. Terminated：什么也不做进入这个状态的条件是：不是running，不是tidying或者terminated，是shutdown并且workerQueue为空，workCount为0，设置tidying为true

#### 10-10-2： submit()和 execute()方法有什么区别

接收参数不同，submit有三种；

Submit有返回值，execute没有

Submit会抛出异常

### 10-11：如何保证多线程安全性？

使用同步synchronized或者lock

只允许单个线程执行。

### 10-12：CAS技术？

缓存锁：当物理机计算机中发出现数据缓存不一致的时候，之前是采用总线锁，锁住cpu和内存之前的通信，其他处理器不能处理其他内存地址的数据，所以改为缓存锁来优化。实现缓存一致性协议：每个处理通过嗅探在总线上传输的数据来检查自己的数据是否过期。设计到缓存锁主要是cas技术

CompareAndSwap：内存地址（主内存变量的内存地址），旧值（工作内存缓存的），新值。当且仅当主内存中的值等于旧值才更新旧值。否则不执行。无论是否更新，都会返回旧值。

### 10-13：Monitor（jvm）对象监视器

偏斜锁，轻量级锁，重量锁

如果monitor的进入数为0，则该线程进入monitor，然后进入数设置为1，该线程为持有者。如果已经被其他线程占用，那么该线程进入阻塞状态，直到进入数变为0；

### 10-14：死锁？

产生现象：两个或多个线程在争夺临界资源，发生相互等待锁的现象，若无外力作用，他们都将无法继续推进下去。

原因：系统资源不足；进程的调度顺序不对；资源分配不相等

如何防止：确定好顺序；超时释放

### 10-15：synchronized实现？

底层原理：

## 11：反射

### 11-1：what？

动态的获取对象信息以及动态调用对象的方法的功能称为java的反射机制。

划重点：运行时

### 11-2：getFields（）和getDeclaredFields（）区别：

前者是：返回该类或接口的可访问public字段

后者：返回该类中或者接口的受保护protected，公共的，默认的，私有的字段，但是不包括继承的字段。

### 11-3：用途？

在开发各种通用框架。例如Spring都是配置化（例如通过xml文件配置bean），为了保证框架的通用性，需要动态加载配置文件中不同的对象或者类，调用不同的方法就需要用到反射.

### 11-4：序列化？

What：将java对象序列化为二进制文件。

反序列化：将字节流（二进制文件）转换成Java对象。

How：实现serializable接口，使用ObjectInputStram ObjectOutStream对对象读写

When：当java对象需要在网络上传输或者 持久化的存储到文件中。

不能被实例化：被static ，trasient声明的成员变量

### 11-5：代理

代理模式是设计模式的一种，提供了对目标对象的额外访问方式，也就是说：通过代理对象访问目标对象，这样可以在不修改原目标对象的前提下，提供额外的功能操作，扩展目标对象的功能。

分为两种：

1. 静态代理：

方式：代理对象和目标对象都要实现一样的接口

缺点：冗余（由于要实现一致的接口，会导致过多不需要 的代理类）；不易维护（接口的方法增加时，目标对象，代理对象都需要修改）

1. 动态代理由于利用了JDK API，动态的在内存中构建对象，从而实现对目标对象的代理过程。又被称为：jdk代理，接口代理。 特点：代理对象不需要实现接口，但是目标对象必须实现接口，否则不能使用动态代理。设计到的类：反射中的Proxy,InvocationHandler。
2. cglib（Code Generation Library），第三方代码生成类库。运行时在内存中动态生成一个子类从而实现对目标对象的功能扩展。

特点：1，JDK实现的动态代理，目标对象必须使用一个或者多个接口，如果想要实现没有接口的目标对象功能扩展，可以使用cglib。2，强大的高性能的，可以在运行时扩展java类，与实现接口，例如Spring AOP，DYN AOP ,的拦截。3,无需实现接口，通过生成类字节码实现代理，比反射快，不存在性能问题，要继承目标类，需要重写方法，不可以final

静态代理和动态代理区别：

* 静态代理在编译时就已经实现，编译完成后代理对象就是一个实际的class文件
* 动态代理是在运行时动态生成，即在运行时生成类字节码class文件，并加载到JVM中。

## 12：克隆

1，What：在内存中将一个已有的对象复制出另一个与之相同的对象；

1. 为什么要使用克隆？

因为在某个API需要提供一个list集合，但是又不需要调用者的修改影响到自身的变化，因此需要克隆，达到数据隔离的目的

3，种类：

浅拷贝：

使用默认的clone方法，对于原始数据的拷贝，如果是基础数据类型拷贝的是值，如果是引用型，那么拷贝的就是就是“引用”。执行快，效率较深拷贝高。数据是相关的。什么时候推荐使用：对象不需要修改。

深拷贝：

1，重写clone方法，不仅仅重写父类的clone方法，还需要调用clone方法；可以做到克隆数据和元数据分离，什么时候推荐使用：如果对象存在引用型。耗时，效率低。

·· 2，复制构造器（可以浅拷贝，也可以深拷贝），

3，序列化：目标类必须实现serializable接口

## 13，JVM

### 13-1：内存：

1. 栈：存放基础数据类型和对象的引用
2. 堆：用来存放所有的Java对象
3. 方法区：包含所有的class和static变量

## 14：网络

### 14-1：一些http状态码

重定向：

301：永久重定向：搜索引擎在抓取新的内容，旧的地址也变为新的地址，默认这个是可缓存的

例如场景：使用域名跳转

302：临时重定向

例如场景：未登陆的用户访问重定向到登录页面

### 14-2：redirect（重定向）和forward（转发）

1，重定向：；客户端行为；服务器根据逻辑，给浏览器发送一个状态码，告诉浏览器去访问新的url，然后地址栏会变更为新的地址；效率较高

使用场景：登录，根据角色跳转到不同的页面

2，转发：服务器端行为；只是将url响应数据读取后再返回给浏览器，浏览器不知道数据的来源，所以地址栏不会发生变更；服务器请求资源；服务器直接访问目标url，将这个url相应的数据直接发送给浏览器。

使用场景：用户注销返回主页面，或者跳转到其他的网站

## 15：javaweb

### 15-1：servlet

1. 生命周期

Init（）初始化----service（）处理客户端请求-----destroy（）终止------由jvm的垃圾回收器进行垃圾回收

Init（）只被调用一次；可以在用户第一次调用时创建，也可以指定在服务器第一次启动时加载

Service（）：处理客户端请求，并将响应数据返回给客户端。其中的doGet（）方法：不指定method的表单提交，或者url’的请求；doPost（）方法：method为post的表单

Destroy（）：只会被调用一次，在servlet生命周期结束时被调用

1. 是什么？

是运行在web服务器中的小型java程序，通过http接收和响应来自web客户端的请求；servlet容器处理多个线程产生的多个的请求，每个线程执行单一的servlet实例的service方法

1. 优点：

1，性能明显很好2，安全的，因为服务器上的java安全管理器执行性了一系列的操作3，平台无关，因为是java编写的4，在web服务器内的地址空间执行，所以不用创建一个单独的线程去处理每个客户端请求？？？5，java类库都可以使用

### 15-1-2：jsp和servlet区别？

Jsp本质是也是servlet，简易版的。会被服务器处理成类似servlet的Java程序，可以简化页面的生成。

不同点：servlet应用逻辑在java文件中，完全从表示层脱离出来。Jsp时Java和html组合成的jsp文件。Jsp侧重视图，servlet侧重逻辑控制。

### 15-2：过滤器Filter和拦截器Interceptor

过滤器：可以动态拦截请求和响应，以变换请求和响应中的信息。可以发生在客户端向服务器端发出请求后，拦截这些请求；可以在服务器端向客户端响应前，处理响应信息。

1. 接口定义位置不同

过滤器Filter定义在javax。Servlet；拦截器Interceptor定义在springframework的web的servlet包中

1. 起作用范围不同

Filter定义在web.xml文件中,只在servlet前后起作用不考虑servlet的实现；拦截器Interceptor可以深入到方法前后，异常抛出等，可以实现过滤器的功能。

1. 使用范围不同

过滤器只可以用于web，拦截器可以用于web，也可以用于app，swing等

1. 规范不同

过滤器是servlet规范，不能使用spring；拦截器是spring一个组件，spring管理，可以使用spring。

1. 执行顺序：由于Filter是服务器调用，interceptor是spring调用，所有filter先执行

拦截器：

1. 执行顺序：请求到达DispatcherServlet -----DispatcherServlet 发送到Interceptor ，执行preHandle，------请求到达controller-------请求结束，postHandle执行

### 15-3：session，cookie

1. Session：会话机制，是用户从访问页面到离开的时间，利用的是cookie来进行信息的处理。当用户首次进入页面时，就会创建一个cookie，到用户离开或者cookie到期表明session结束。

工作原理：浏览器向服务器端发出请求需要创建一个session时，服务器首先查看请求中是否包含sessionid，如果没有创建一个返回，如果有了就匹配出这个session标记的数据，返回到浏览器，默认是把session标记存储到cookie中，在以后的请求中自动带上session标记。考虑到cookie会被认为禁止，所以session也可以采用url重写，一种是作为url的附加信息（.com;jsessionId=000），一种是作为参数(.com?jsessionid=000)

2，Cookie：客户端中存储的数据，伴随每次请求都发送到服务器端

共同：：都是为了跟踪浏览器用户的会话方式。都不安全

区别：

1. 保存的位置不同
2. 有效期的不同：虽然都有设置有效期，session是放在服务器上，cookie是创建时设置
3. Cookie相对更不安全，一些不重要但是必须的数据可以存在cookie中，访问多了session就不适合了。

### 15-4：如何防止sql注入

SQL注入：是一种代码注入技术，用于攻击数据驱动的应用，恶意的sql语句被插入到执行的实体字段，只对sql语句编译过程有破坏作用，例如实现免登录，or 1=1

方法：1，PreparedStatement:预编译，执行阶段只是把输入串作为数据处理；2，使用正则表达式过滤传入的参数，

切记：不要拼接字符串

### 15-5：xss攻击

1. XSS：Cross-site Scripting ,跨站脚本攻击，是一种代码注入攻击。攻击者可以通过在目标网站上注入恶意脚本，使之在用户的浏览器上运行，利用这些恶意脚本，攻击者可获取用户的cookie，sessionid，危害数据安全
2. 分类：一类是反射性（非持久性）：通过url传给服务器，服务器不加处理返回给浏览器，二类是存储型（持久性）：存储到数据库中
3. 预防：1，输入过滤，一些敏感标签2，纯前端渲染，代码和数据分离3，转义html

### 15-6：CSRF攻击：

Cross Site Request Forgery：跨站域请求伪造，网络的攻击方式。攻击对象：会产生数据改变的服务，读取不需要受到保护。

手段：1，验证HTTP Referer字段，记录来源地址，不是很安全2，请求地址中添加token并验证：不把token放入cookie中，在服务器端简历一个拦截器验证token，比1相对安全。3，在HTTP头中自定义属性并验证，可以把token放入请求头中

## 16：框架

### 16-1：spring

#### 谈谈对spring的理解：

Spring是一个轻量级的框架（相对于ejb企业级的JavaBean来说，资源少，耦合低），主要的核心时ioc和aop；

**ioc**：控制反转，spring容器控制对象的创建销毁。正转：new 一个对象，在对象中主动的创建一个对象。反转:通过spring容器创建对象，注入到所需要的地方，对象被动接受。里面有个重要的核心思想：依赖注入。他是通过来实现的，概念：在程序运行时，动态的获取对象的信息以及动态的调用对象方法被称为反射，正好满足DI的要求。

**AOP**：面向切面编程。用于处理系统中分布于各个模块中间的横切关注点，比如事务管理，日志，缓存。Aop的实现关键在于代理，分为静态和动态，静态的代表为**Aspectj;**动态代理：jdk和cglib。静态代理：aop框架会在编译阶段就生成aop代理类，将aspect织入java字节码中，运行时就是增加之后的aop代理；动态代理：aop框架不会去修改字节码，而是在内存中暂时的为方法生成一个aop对象（里面包含目标对象所有的方法，并且在特定的切点做了增强处理，并回调源对象方法）；j**dk代理**：通过反射来接收被代理的类，并且要求被代理的类必须实现一个接口，核心是：InvocationHandler和Proxy;.。cglib：目标类没有实现接口，通过继承的方式做的动态代理，因此当类被生成final无法使用。

#### springMVC的工作原理

客户端发出请求提交到前端控制器DispatcherServlet，然后前端控制器寻找HandleMapping，找到处理请求的controller，处理业务逻辑后将数据返回给ModelAndView，前端控制器查询视图解析器ViewResolver，找到指定的视图，然后返回给浏览器。

### 16-2：springmvc和struts2

1. 拦截的区别：Springmvc：是方法级别的拦截，一个方法对应一个request上下文，struts是类级别的拦截，一个类对应request上下文
2. 效率，性能：springMVC较高
3. 拦截机制：struts使用的是自己的interceptor机制，spring使用的aop的形式
4. Request，response：springmvc独享，因为方法之间是独立的，struts是共享的，编码会比较乱，把每一个request进行封装，把request，session等servlet生命周期的变量封装成一个个的map，供action使用，耗内存。

### 16-3：@Component，@Repository，@Service,@Controller

等效的注解

不好归类的组件：@Component

持久层：@Repository

业务层：@Service

控制层：@Controller

被这些注解注释的类会被spring容器管理，作用和 xml中的bean节点配置组件是一样的

### 16-4：@Autowired,@Resourse

作用相当，

@Autowired：按照类型自动注入，如果想用名字byName装配，可以结合@Qualifier使用

@Resourse：默认是按照名字注入，但是也有按照类型byType自动注入的选择，如果都不指定，那么就会按照反射机制按照byName自动注入

当指定类型byType的时候，如果上下文找到类型匹配的唯一bean装配，但是如果找过多个或者没有找到都会抛出异常

### 16-5：bean的作用域

1. 原型prototype，每次创建新的对象—安全
2. 单例singleton，默认---无状态安全，其他不安全
3. Request：请求，每次http请求创建一个新对象，
4. Session：会话；同一个会话共享一个实例，不同实例使用不同实例。
5. Global-session：全局会话；所有会话共享一个实例

### 16-6：装配bean的方式

1. 自动化装配

注解的方式例如1，@Component或者加上自动名字，2，@Configuration，3，@Named（“”）为bean设置id，4，@Autowired,

1. Java代码装配
2. 声明类@Bean

//@Bean注解会告诉Spring这个方法将会返回一个对象,该对象要注册为Spring应用上下文的bean,默认情况下,bean的ID于带有@Bean注解的方法名一样,也可以重命名

1. Xml文件装配

//更好的方法是借助id属性<bean id="compactDisc" class="soundsystem.SgtPeppers" />

### 16-7：事务的实现方式

事务的四个特性：acid：

原子性：要么全做要么全不做，

一致性：执行事务前后数据保持一致，

隔离性：并发访问数据库时，各事务之间时独立的，

持久性：一个事务被提交之后在数据库中数据的改变是持久的

1. 编程式

调用beginTransaction()、commit()、rollback()-----已经很少了，除非是古老的代码

1. 声明式
2. 基于 TransactionProxyFactoryBean
3. 基于 @Transactional 的声明式事务
4. 基于Aspectj AOP配置事务

### 16-8：声明式事务的隔离机制

1. 默认
2. 允许读取尚未提交的更改，会导致脏读，不可重复读，幻影读
3. 允许从已经提交的事务去读取，防止脏读，其他两个依然可能发生
4. 对相同字段的多次读取结果是一致的，防止脏读和不可以重复读
5. 完全服从acid的隔离机制，速度比较慢，锁定当前事务的表执行。

### 16-9：dispatcherServlet或者spring的九大组件

Handler：处理器，每个被@RequestMapping标注的方法可以看作是一个handler，处理请求。

1. HandlerMapping：用来查找Handler，查找请求的需要那个handler处理
2. HandlerAdapter：适配器；让固定的servlet处理方法调用灵活的Handler处理
3. HandlerExceptionResolver：处理异常，根据异常设置ModelAndView
4. ViewResolver：解析成view视图
5. RequestToViewNameTranslator：根据viewname查找view
6. LocaleResolver：用于从request解析出locale，locale可以对不同的用户展示不通过的view
7. ThemeResolver：解析主题（对应properties文件）
8. MultipartResolver：处理上传请求；将普通的request包装成MultipartHttpServletRequest，可以直接获取到文件
9. FlashMapManager：管理FlashMap，主要用于redirect传递参数

### 16-10：@RequestMapping

@RequestMapping：处理web请求的映射，可以控制类或者方法上

### 16-11：springMVC运行过程

## 17：异常

概念：程序运行中发生的一些不正常的事情，但是并不是所有的错误都是异常

异常情形：阻止当前方法或者租用与继续执行的问题。

抛出异常：从当前环境跳出，并且把问题提交到上一层环境。Throw new 异常（），需要使用new在堆上创建异常对象，这伴随着存储空间和构造器的使用。

Try catch finally return执行顺序

1. try没有异常，有return：try，finally：返回的是finally执行后的值；执行顺序：try->finally
2. try没有异常，有return：try，catch：返回的是try块中的最后值；执行顺序：try->finally->try
3. try抛出异常，有return：都有：返回的是finally块中的最后值；执行顺序：try->catch，->finally
4. try抛出异常；有return：try和catch；返回的是catch中return值；执行顺序：try->catch->finally->catch
5. try，catch抛出异常，有return：try和catch；返回的异常;执行顺序：try->catch->finally->抛出
6. try，catch抛出异常，有return：都有；返回的是finally的return值，不检查异常；执行顺序：try->catch->finally

### 17-1：异常分类

1. 异常根类：Throwable；Error和Exception都继承自它。
2. Exception：运行时异常（RuntimeException）和流异常(IOException)；分为两类：
   1. 运行时异常：数组索引越界；空指针；找不到类；非法参数；类转换；
   2. 流异常：文件找不到

3，Error：一类，虚拟机错误：栈溢出，内存不足；一类，awt错误。

4，异常分为可检查异常，不可检查异常（运行时异常和它的子类，error类）

## 18：设计模式

概念：是一套被反复使用的，多数人知晓的，经过分类编目的，代码设计经验的总结。

类型：

1. 创建型

工厂模式：

抽象工厂模式

单例模式

建造者模式

原型模式

1. 结构型

适配器模式

过滤器模式

组合模式

装饰器模式

代理模式

1. 行为型

解释器模式

迭代器模式

观察者模式

Spring中出现的设计模式：

1. 模板模式：在父类中定义算法的主要流程，把一些个性化的步骤延迟到子类中去实现，父类始终控制着这个流程的主动权，子类只是辅助父类实现某些可定制的步骤。例如：jdc模板，hibernate模板
2. 简单工厂：又称为静态工厂方法。是由一个工厂类根据传入的参数动态的决定创建那个类。例如：Spring中的beanFactory
3. 单例模式：保证一个类只有一个实例，并提供一个访问它的全局访问点。注意：spring中默认的bean都是单例模式的
4. 适配器（Adapter）Spring Aop使用Advice (通知)来增强被代理类的功能
5. 装饰模式：也叫包装器模式。动态的给对象添加一些额外的职责。譬如：项目中连接了多个数据库，不同的客户访问不同的数据库
6. 观察者：定义对象间的一种一对多的关系，当一个对象的状态发生变化的时候，所有依赖于它的对象都得到通知并被通知更新。

## 19：SpringBoot

### 19-1：spring boot 和spring mvc

Spring boot：实现了自动配置，降低了项目搭建的复杂度。之前的spring项目需要大量的配置，spring-web，service，dao，web的xml文件等，spring boot本身并不提供spring框架的和新特性以及拓展功能，只是用于快速，敏捷的开发新一代基于spring框架的应用程序，为了让开发者更专注于业务逻辑。如果你搭建的是web项目，依然需要springmvc作为mvc框架。

Springmvc：只是提供了一种解耦合的方式来开发web应用。是一个框架。

### 19-2：spring boot优点

1. 简化配置，2，入门容易3，内嵌服务器，4，为spring cloud奠定基础5，整合很多第三方库，方便使用6，社区活跃

### 19-3：核心配置文件

Spring boot有两种上下文。格式：.yml; .properties

application文件：树形结构，中间有空格；主要用于项目的自动化配置

Boostrap文件：应用程序的父上下文，优先于application文件。主要用于从额外的资源来加载配置信息，还可以在本地外部配置文件中解密属性。需要被用于spring cloudConfig配置中心。

公用一个环境。

### 19-4: Spring Data JPA、MyBatis还有Hibernate有什么区别

性能角度：h比b要高