基础篇

基本功

- 面向对象的三大特征以及对这些特性的理解:
  - 继承:
    - 继承是从已有类得到继承信息创建新类的过程。提供继承信息的类被称为父类（超类、基类）；得到继承信息的类被称为子类（派生类）。继承让变化中的软件系统有了一定的延续性,java中只支持单继承,不支持多继承,但是支持多重继承
  - 封装:
    - 面向对象的本质就是将现实世界描绘成一系列完全自治、封闭的对象,外界无法改变直接操作和修改。我们在类中编写的方法就是对实现细节的一种封装；我们编写一个类就是对数据和数据操作的封装。我们还可以通过方法来控制成员变量的操作,提高了代码的安全性,把代码用方法进行封装,提高了代码的复用性
  - 多态:
    - 事务在运行过程中存在不同的状态,多态存在的三个前提:
      - 要有继承关系
      - 子类要重写父类的方法
      - 父类引用子类的对象
    - 多态成员访问的特点
      - **成员变量** :编译看左边(父类),运行看左边(父类)
      - **成员方法** :编译看左边(父类),运行看右边(子类) 动态绑定
      - **静态方法** :编译看左边(父类)，运行看左边(父类)

- 对象的初始化过程
  - 默认初始化--->显示初始化--->构造初始化--->set初始化

---

final, finally, finalize 的区别

- **final修饰符**: 被final修饰的类,就不能派生新的子类,不能作为父类被子类继承
- **finally**: 是在异常处理的时候提供的finally块,不管有没有异常被抛出,捕获,finally块都会被执行
- **finalize** : 这个方法来自java.lang.Object包,用于回收资源

---

int 和 Integer 有什么区别

- int为基本数据类型,integer为包装类型
- int的初始值是0,integer的初始值是null

重载和重写的区别

- 重载涉及同一个类中的同名方法,要求方法名相同,参数列表不同,与返回值类型,访问修饰符无关
- 重写实际的是子类和父类之间的同名方法,要求方法名相同,参数列表相同,返回值相同,访问修饰符不能严于父类

抽象类和接口有什么区别

- 抽象类和就扣都是继承树的上层,他们共同特点如下:
  - 都是上层的抽象层
  - 都不能被实例化
  - 都能包换抽象的方法
- 区别如下
  - 在抽象类中可以写非抽象方法,从而避免在子类中重复书写他们,这样可以提高代码的复用性,这就是抽象类的优势,接口中只能有抽象的方法.
  - 一个类只能继承一个直接父类,这个父类可以试具体的类也可以是抽象类,但是一个类可以实现多个接口

反射的用途及实现

- 反射机制就是在运行状态中,对于任意一个类,都能够知道这个类的所有属性和方法,对于任意一个对象,都能够调用他的任意一个方法.
- 通过反射,我们可以在运行时获得程序或程序集中每一个类型成员和成员变量的信息
- 反射最主要的用途就是开发各种通用的框架,比如Spring,它是就是通过xml文件配置javabean,为了保证框架的通用性,他们可能根据配置文件加载不同的对象和类,调用不同的方法.
- 举一个例子，在运用Struts 2框架的开发中我们一般会在struts.xml里去配置Action，比如：

```
<action name="login"
           class="org.ScZyhSoft.test.action.SimpleLoginAction"
           method="execute">
       <result>/shop/shop-index.jsp</result>
       <result name="error">login.jsp</result>
   </action>
```

配置文件与Action建立了一种映射关系，当View层发出请求时，请求会被StrutsPrepareAndExecuteFilter拦截，然后StrutsPrepareAndExecuteFilter会去动态地创建Action实例。

- 实现
  - 反射的前提是要获取类的对象Class对象
  - 通过Object类的getClass方法

    ```
    World world = new world();
    Class cla = world.getClass();
    ```

  - 通过对象实例方法获取对象

```
Class cla = world.class;
```

- 通过Class.forName方式

```
Class cla = Class.forName("全类名");
```

自定义注解的场景及实现

- 适用于登陆,权限拦截这种场景
- https://www.colabug.com/2266491.html

HTTP 请求的 GET 与 POST 方式的区别 还有什么其他的请求方式

- HTTP请求的GET与Post方式的区别,还有什么请求
  - 其实都是向服务器发送请求,Get是向服务器发送索取数据的一种请求,而Post是向服务器提交数据的一种请求
  - GET请求将提交的数据防止在HTTP请求协议头中 POST提交的数据则放在实体数据中
  - 其他请求方式为
    - HEAD:只请求页面的首部
    - PUT:从客户端向服务器传送的数据取代指定的文档的内容
    - DELETE:请求服务器删除指定的内容

RestFul风格理解

- RestFul风格就是一种面向资源服务的API设计方式,它是一种设计风格
- 待加入

---

session 与 cookie 区别

- session在服务端,cookie在客户端(浏览器)
- session默认被存在服务器的一个文件里
- session的运行依赖于session id,而session id是存在cookie中的,也就是说浏览器禁用了cookie,同时session也会失效,但是可以通过其他方式实现,比如在url中传递session id
- session可以存放在文件,数据库,或者内存中都可以
- 用户验证这种场合一般会用session
- 因此,维持一个会话的核心就是客户端的唯一标识,session id

---

session 分布式处理

- 目前分布式情况下session的处理方式有以下几种

1. session粘性,就是说用户在访问了某台服务器后,之后的操作就让其走该服务器就好,那么就可以让用户只访问这台机器.

- 优点:操作简单,不对session做任何操作
- 缺点:当一台机器挂了以后,流量切向其他的机器,会丢失部分用户的session
- 使用场景:发生故障对用户产生的影响比较小的场景

nginx配置

```
upstream test{
    //这里添加的是上面启动好的两台服务器
    ip_hash;//粘性Session
     server 192.168.22.229:8080 weight=1;
     server 192.168.22.230:8080 weight=1;
}
```

2.使用广播的方式

- 当一台服务器中的session进行了增删改操作后,将这个session中的所有数据,通过广播的方式同步到别的服务器上去
- 优点:容错性高
- 缺点:机器不能太多,session数量不能太大,否贼会造成网络阻塞,使服务器变慢

3.使用中间件共享session

- 使用redis或者Memcached去当做有个中间件,session中的数据存放在其中,这里需要的是redis或者Memcached必须是集群
- 两种做法
    - 粘性:说白了就是和第一种方式一样,一个用户的请求只走一个服务器并且在拿session数据的时候,都只在该台服务器上，但是用户的session需要保存在redis上，作为备份（容灾用）。当一台服务器挂掉了，那么就可以将该用户的session复制到其他的机器上并且把流量转发。
    - 非黏性：这种情况下，就是将用户的session存放在redis上，用户在访问的时候，读取修改都在redis上
- 目前这种做法是大家使用最多的方法

4. session数据存放在数据库中

- 优点:数据可以持久化,服务器挂了也没关系
- 缺点:就是慢,而且用户过多的时候,性能低下

使用JDBC的步骤

- 注册驱动---->建立连接---->创建执行sql语句的statement---->处理执行结果---->释放资源

事物

- 事物基本概念
    - 一组要么同时执行成功,要么同时执行失败的SQL语句,是数据库操作的一个执行单元
    - 事物开始于:

        - 连接到数据库上,并执行了一条DML语句(insert,update,delete)
        - 前一个事物结束后,又输入了另一条DML语句
    - 事物结束于:
        - 执行了Commit或者Rollback语句
        - 执行一条DDL语句,例如Create table语句,在这种情况下会自动执行commit语句
        - 执行一条DCL语句,例如Grant语句,这种情况下,会自动执行commit语句
        - 断开与数据库的连接
        - 执行了一条CML语句,该语句失败了,在这种情况下,会为这个无效的DML语句执行Rollback语句
- 事物的四大特点
    - 原子性(atomicity)
        - 表示一个事物内的所有操作是一个整体,要么全部成功,要么全部失败
    - 一致性(consistency)
        - 表示一个事物内有一个操作失败时,所有的更改过的数据必须全部回滚到修改前的状态
    - 隔离性(isolation)
        - 一个事物的执行不能被其他事物干扰
    - 持久性(durability)
        - 一个事物一旦提交,他对数据库中数据的改变就应该是永久性的
- 事物的隔离级别

级别名	级别号	脏读	不可重复读	幻读
未提交读(Read Uacommitted)	0	可能	可能	可能
提交读(Read Committed)	1	不可能	可能	可能
重复读(Repeatable Read)	2	不可能	不可能	可能
序列化(Serializable)	3	不可能	不可能	不可能

- 未提交读(Read Uacommitted)：是最低的事务隔离级别，它允许另外一个事务可以看到这个事务未提交的数据。
- 提交读(Read Committed)： 保证一个事物提交后才能被另外一个事务读取。另外一个事务不能读取该事物未提交的数据。
- 重复读(Repeatable Read)： 这种事务隔离级别可以防止脏读，不可重复读。但是可能会出现幻读。它除了保证一个事务不能被另外一个事务读取未提交的数据之外还避免了以下情况产生（不可重复读）。
- 序列化(Serializable)：这是花费最高代价但最可靠的事务隔离级别。事务被处理为顺序执行。除了防止脏读，不可重复读之外，还避免了幻象读。
- 脏读、不可重复读、幻读概念说明：
-
    - 脏读：指当一个事务正字访问数据，并且对数据进行了修改，而这种数据还没有提交到数据库中，这时，另外一个事务也访问这个数据，然后使用了这个数据。因为这个数据还没有提交那么另外一个事务读取到的这个数据我们称之为脏数据。依据脏数据所做的操作肯能是不正确的。
    - 不可重复读：指在一个事务内，多次读同一数据。在这个事务还没有执行结束，另外一个事务也访问该同一数据，那么在第一个事务中的两次读取数据之间，由于第二个事务的修改第一个事务两次读到的数

据可能是不一样的，这样就发生了在一个事物内两次连续读到的数据是不一样的，这种情况被称为是不可重复读。

- 幻象读：一个事务先后读取一个范围的记录，但两次读取的纪录数不同，我们称之为幻象读（两次执行同一条 select 语句会出现不同的结果，第二次读会增加一数据行，并没有说这两次执行是在同一个事务中）

- spring事务传播特性：

  事务传播行为就是多个事务方法相互调用时，事务如何在这些方法间传播。spring支持7种事务传播行为：

  - propagation_requierd：如果当前没有事务，就新建一个事务，如果已存在一个事务中，加入到这个事务中，这是最常见的选择。
  - propagation_supports：支持当前事务，如果没有当前事务，就以非事务方法执行。
  - propagation_mandatory：使用当前事务，如果没有当前事务，就抛出异常。
  - propagation_required_new：新建事务，如果当前存在事务，把当前事务挂起。
  - propagation_not_supported：以非事务方式执行操作，如果当前存在事务，就把当前事务挂起。
  - propagation_never：以非事务方式执行操作，如果当前事务存在则抛出异常。
  - propagation_nested：如果当前存在事务，则在嵌套事务内执行。如果当前没有事务，则执行与propagation_required类似的操作

---

equals 与 == 的区别

- 通俗点讲,==是看看左右是不是一个东西,equals是看看左右是不是长的一摸一样
- 术语来讲的区别：
  1.==是判断两个变量或实例是不是指向同一个内存空间
  equals是判断两个变量或实例所指向的内存空间的值是不是相同
  2.==是指对内存地址进行比较
  equals()是对字符串的内容进行比较
  3.==指引用是否相同
  equals()指的是值是否相同

MVC 设计思想

- MVC主要就是在java开发中的一种设计模式：
  - M：Modle（模型，主要是Service业务逻辑层和Dao和数据库取得连接并发送数据的层）
  - V： view（视图，也就是用户看的界面，通常是我们所熟知的前台页面，jsp等）
  - C： controller（控制层，可以把他看作一个中转，他接收从前台用户发来的请求，并调用service,dao把数据发送到后台，后台经过数据库的操作及业务逻辑分析又将数据返回给controller，最后再返回前台jsp页面）。
- 说说Mvc的优缺点，
  - 优点:

    1，MVC设计模式可以说实现了分层开发。各个层都有各个层的作用。

    2，降低了层与层之间的依赖，有利于代码的标准化开发

    3，再用新的代码业务逻辑替换时，只需要替换相对应的层，大大降低了我们的工作量，分工明确。

- 缺点:

  麻烦，有些代码重复的过多，不利于在实际开发中使用