# Oracle数据库。★★

## 地址学习https://www.cnblogs.com/ysocean/

## 使用工具Oracle SQL Developer

## 表空间

表空间，ORacle数据库的逻辑单元，一个表空间可以与多个数据文件关联，一个数据库下可以建立多个表空间，一个表空间可以建立多个用户，一个用户可以建立多个表，Oracle是多用户的的，mysql是多数据库的。



**创建用户：**



Oracle数据和其他数据库产品的区别在于，表和其他的数据库对象都是存储在用户下的，所以说Oracle是基于用户的数据库。

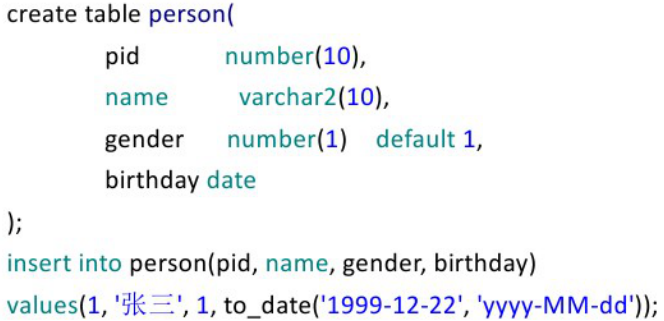
1. 用户赋权限

Oracle有是三个权限，connect权限，resource权限和dba权限。



DBA角色：拥有全部特权，是系统的最高权限，只有dba才可以创建数据库结构，并且系统权限也需要dba授权，切dba用户可以操作全体用户的任意基表，包括删除

1. 建表



Varchar2()---1.varchar2把所有字符都占两字节处理(一般情况下)，varchar只对汉字和全角等字符占两字节，数字，英文字符等都是一个字节；VARCHAR2把空串等同于null处理，而varchar仍按照空串处理；VARCHAR2字符要用几个字节存储，要看数据库使用的字符集，

大部分情况下建议使用varchar2类型，可以保证更好的兼容性。

关于number：Oracle中没有int数据类型，为了照顾其他数据库的兼容性，新增了int类型作为number的子集，Oracle只用NUMBER(m,n)就可以表示任何复杂的数字数据。

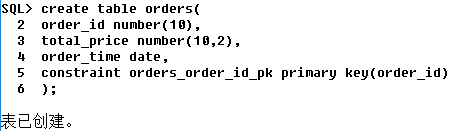
1. 约束

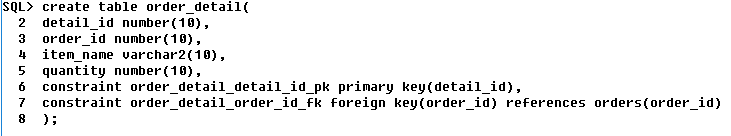


检查约束可以用来约束字段值的合法范围，例如性别用1和2 表示，

Gender number(1) check (gender in(1,2)),

外键约束：





## 事务

Oracle的3种事务隔离级别，READ COMMITED,SERIALIZABLE,READ ONLY,默认的事务隔离级别是READ COMMITED

## 管理其他数据库对象

视图：封装了一条复杂查询的语句，视图是一个虚表，最大的有点就是简化复杂的查询。不建议通过视图对数据进行修改，因为会受到很多限制。

序列：在很多数据库中存在一个自动增长的列，如果想要在Oracle中完成自动增长的功能，则只能依靠序列完成，Oracle将序列值装入内存可以提高访问效率。

## 数据的导入导出

使用cmd命令整库导

使用cmd命令按用户导

Exp system(用户)/itcast(密码) owner=itcast001 file=

Imp system/itcast file=

使用cmd命令按表导

Exp itcast001/itcast file= tables=t\_person,t\_studnent

使用PLSQL Developer导

## 数据回滚

Select \* from 表名 as of timestamp to\_timestamp(‘2019-02-26’,’yyyy-mm-dd hh24:mi:ss’);//查询该表在某时间点以后的操作

Alter table 表名 enable row movement;

Flashback table 表名 to\_timestamp(‘2019-02-16’,’yyyy-mm-dd hh24:mi:ss’);

//回滚到某时间点之前

## 表和视图

**区别：表** -- 是数据库中的主要结构，它总是表示单个的、特定的集合。每个表至少包含一个字段，即一个主键(primary key)，他唯一地标识表的每条记录。

**视图 -**- 是由来自数据库中的一个或多个表或多个表的字段所组成的一个虚拟的表。实际上它自身不存储任何数据，存储在数据库中的有关视图的唯一信息几句是它的结构。

表指数据表，相当于数据原始资料。视图是对表的运算，生成的结果类似于表，存储的是运算方法，而不是具体数据

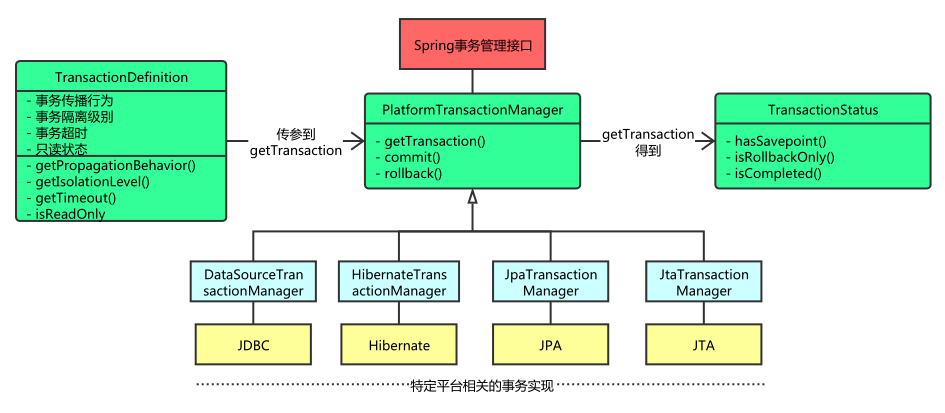
视图是查看数据表的一种方法，可以查询数据表中某些字段构成的数据，只是一些SQL语句的集合。从安全的角度说，视图可以不给用户接触数据表，从而不知道表结构。

视图的建立和删除只影响视图本身，不影响对应的基本表。

不能对视图进行update或者insert into操作。

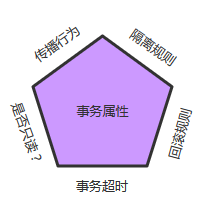
**联系：**视图（view）是在基本表之上建立的表，它的结构（即所定义的列）和内容（即所有数据行）都来自基本表，它依据基本表存在而存在。一个视图可以对应一个基本表，也可以对应多个基本表。视图是基本表的抽象和在逻辑意义上建立的新关系。

# Spring★★★★



Spring事务管理器的接口是 org.springframework.transaction.PlatformTransactionManager，如上图所示，Spring并不直接管理事务，通过这个接口，Spring为各个平台如JDBC、Hibernate等都提供了对应的事务管理器，也就是将事务管理的职责委托给Hibernate或者JTA等持久化机制所提供的相关平台框架的事务来实现。

事务属性包含了5个方面，



**传播行为：当事务方法被另一个事务方法调用时，必须指定事务应该如何传播。**

**隔离级别：定义了一个事务可能受其他并发事务影响的程度。**

**只读、事务超时、回滚规则**

## Spring编程式事务和声明式事务的区别：

**编程式事务处理**：所谓编程式事务指的是通过编码方式实现事务，允许用户在代码中精确定义事务的边界。即类似于JDBC编程实现事务管理。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。

**声明式事务处理**：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(或通过基于@Transactional注解的方式)，便可以将事务规则应用到业务逻辑中。

　　简单地说，编程式事务侵入到了业务代码里面，但是提供了更加详细的事务管理；而声明式事务由于基于AOP，所以既能起到事务管理的作用，又可以不影响业务代码的具体实现。

## Spring事务练习。。。。。。

## 概念

### OOP和AOP

OOP（面向对象编程）针对业务处理过程的实体及其属性和行为进行抽象封装，以获得更加清晰高效的逻辑单元划分。

而AOP则是针对业务处理过程中的切面进行提取，它所面对的是处理过程中的某个步骤或阶段，以获得逻辑过程中各部分之间低耦合性的隔离效果。这两种设计思想在目标上有着本质的差异。

举个简单的例子，对于“雇员”这样一个业务实体进行封装，自然是OOP/OOD的任务，我们可以为其建立一个“Employee”类，并将“雇员”相关的属性和行为封装其中。而用AOP设计思想对“雇员”进行封装将无从谈起。同样，对于“权限检查”这一动作片断进行划分，则是AOP的目标领域。

## 3.IOC

## 4.DI

## 5.注解配置DI、IOC

1. 持久层、业务层、控制层 分贝用@Repository@Service@Controller

\* @Commonent对那些比较中立的类进行凝视，交给spring进行管理，不清楚是哪个类

\* @Autowired注解和@Resource用来修饰字段、构造方法或设置方法并做注入的

\* @Component("persnID") value的值是bean的id 即bean的唯一id

@Resource 注解，它可以对类成员变量、方法及构造函数进行标注，完成自动装配的工作。 通过 @Resource 的使用来消除 set ，get方法

1、IOC操作

      1）@Component  ：用于实现ioc控制反转,属性value用于实现对象id定义

      2）@Scope    ：用于实现单例，多例，request，session等的配置

      3）@Controller  ： 专门用于controller层的注解，实现控制反转,属性value

      4）@Service  ： 专门用于service层的注解，实现控制反转,属性value

      5）@Repository  ： 专门用于持久层的注解，实现控制反转,属性value

    2、DI操作

      1）@Resource(name="userService")--实现依赖注入

                        先按照name加type的方式检索；其次缺失name或者确实type只按照一项进行检索；其次name和type都省略，那么默认还是按照name检索

      2）@Autowired---自动装配，此注解实现的自动装配是按照类实现注入，如果出现同名的类，那么此注解方式就不能识别是哪个类实现注入，此时

      会出现异常；如果为了避免出现异常，可以采用@Qualifier(value="es")(熟悉)实现辅助，即Qualifier可以指明对象id

1. @component、@Repository、@Service、@Service和@Controller四个注解在普遍使用中是等效的，在web项目中为了区分三层架构中不同层之间bean的创建，为了避免注解使用的混乱，做了区分。①对于bean属性的依赖注入分为两类，一时属性是string类型或者基本数据类型，spring容器提供了@value这个注解，另一类是对于属性是对象的，spring提供了@Autowired和@Resource这两个注解，其中@Autowired这注解是spring框架自带的注解，而@Resource这个注解是javax扩展包里注解规范的一种，②③

## AOP

AOP（Aspect Oriented Programming），通常称为面向切面编程。它利用一种称为"横切"的技术，剖解开封装的对象内部，并将那些影响了多个类的公共行为封装到一个可重用模块，并将其命名为"Aspect"，即切面。所谓"切面"，简单说就是那些与业务无关，却为业务模块所共同调用的逻辑或责任封装起来，便于减少系统的重复代码，降低模块之间的耦合度，并有利于未来的可操作性和可维护性。

### 6.1使用静态代理

UserService接口-》UserService的实现类-》创建事务类MyTransaction-》创建代理类ProxyUser(重写Userservice的**所有**方法)  
这是一个很基础的静态代理，业务类UserServiceImpl 只需要关注业务逻辑本身，保证了业务的重用性，这也是代理类的优点，没什么好说的。我们主要说说这样写的缺点：

　　①、代理对象的一个接口只服务于一种类型的对象，如果要代理的方法很多，势必要为每一种方法都进行代理，静态代理在程序规模稍大时就无法胜任了。

②、如果接口增加一个方法，比如 UserService 增加修改 updateUser(）方法，则除了所有实现类需要实现这个方法外，所有代理类也需要实现此方法。增加了代码维护的复杂度。

### 6.2使用JDK动态代理

维护更方便，不需要一个个复写。

AOP术语：

1.target：目标类，需要被代理的类。例如：UserService

　　2.Joinpoint(连接点):所谓连接点是指那些可能被拦截到的方法。例如：所有的方法

　　3.PointCut 切入点：已经被增强的连接点。例如：addUser()

　　4.advice 通知/增强，增强代码。例如：after、before

　　5. Weaving(织入):是指把增强advice应用到目标对象target来创建新的代理对象proxy的过程.

　　6.proxy 代理类：通知+切入点

7. Aspect(切面): 是切入点pointcut和通知advice的结合

Spring按照通知Advice在目标类方法的连接点位置，可以分为5类

前置通知 org.springframework.aop.MethodBeforeAdvice

在目标方法执行前实施增强，比如上面例子的 before()方法

后置通知 org.springframework.aop.AfterReturningAdvice

在目标方法执行后实施增强，比如上面例子的 after()方法

环绕通知 org.aopalliance.intercept.MethodInterceptor

在目标方法执行前后实施增强

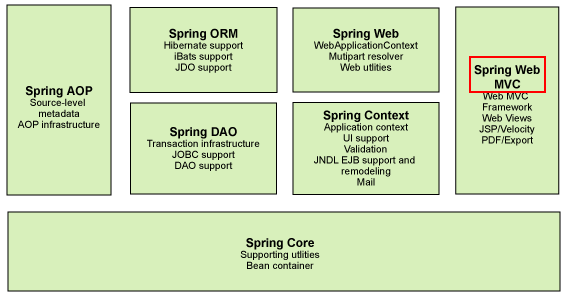
异常抛出通知 org.springframework.aop.ThrowsAdvice

在方法抛出异常后实施增强

引介通知 org.springframework.aop.IntroductionInterceptor

在目标类中添加一些新的方法和属性

# SpringMVC

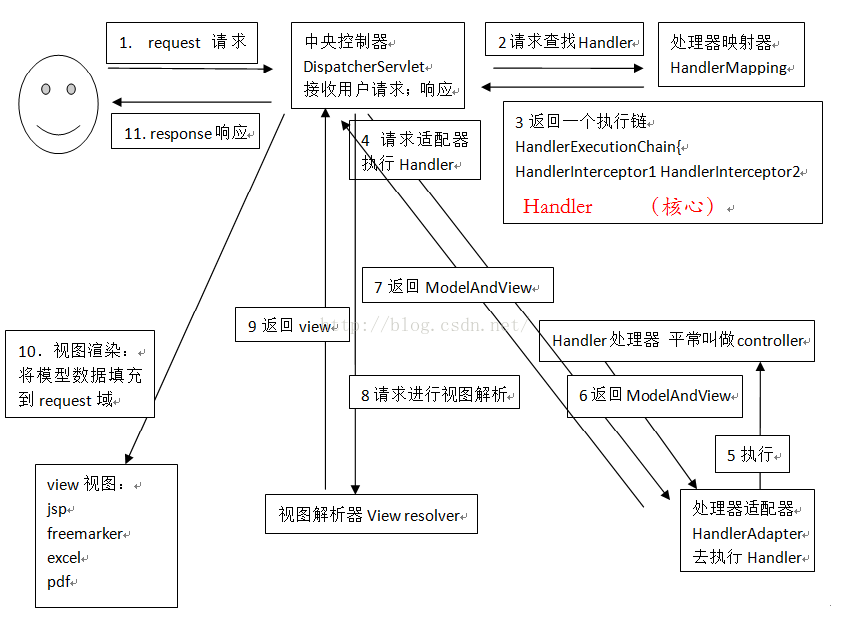
 Spring 的基本架构中，红色圈起来的 Spring Web MVC

SpringMVC 是类似于 Struts2 的一个 MVC 框架，在实际开发中，接收浏览器的请求响应，对数据进行处理，然后返回页面进行显示，但是上手难度却比 Struts2 简单多了。而且由于 Struts2 所暴露出来的安全问题，SpringMVC 已经成为了大多数企业优先选择的框架。

Spring Web MVC是一种基于Java的实现了Web MVC设计模式的请求驱动类型的轻量级Web框架，即使用了MVC架构模式的思想，将web层进行职责解耦，基于请求驱动指的就是使用请求-响应模型，框架的目的就是帮助我们简化开发，Spring Web MVC也是要简化我们日常Web开发的。

　　与之相反的是基于组件的、事件驱动的Web框架，如Tapestry、JSF等，在此就不介绍了。

　　Spring Web MVC也是服务到工作者模式的实现，但进行可优化。前端控制器是DispatcherServlet；应用控制器其实拆为处理器映射器(Handler Mapping)进行处理器管理和视图解析器(View Resolver)进行视图管理；页面控制器/动作/处理器为Controller接口（仅包含ModelAndView handleRequest(request, response) 方法）的实现（也可以是任何的POJO类）；支持本地化（Locale）解析、主题（Theme）解析及文件上传等；提供了非常灵活的数据验证、格式化和数据绑定机制；提供了强大的约定大于配置（惯例优先原则）的契约式编程支持



第一步：用户发送请求到前端控制器（DispatcherServlet）。

第二步：前端控制器请求 HandlerMapping 查找 Handler，可以根据 xml 配置、注解进行查找。

第三步： 处理器映射器 HandlerMapping 向前端控制器返回 Handler

第四步：前端控制器调用处理器适配器去执行 Handler

第五步：处理器适配器执行 Handler

第六步：Handler 执行完成后给适配器返回 ModelAndView

第七步：处理器适配器向前端控制器返回 ModelAndView

　　　　ModelAndView 是SpringMVC 框架的一个底层对象，包括 Model 和 View

第八步：前端控制器请求试图解析器去进行视图解析

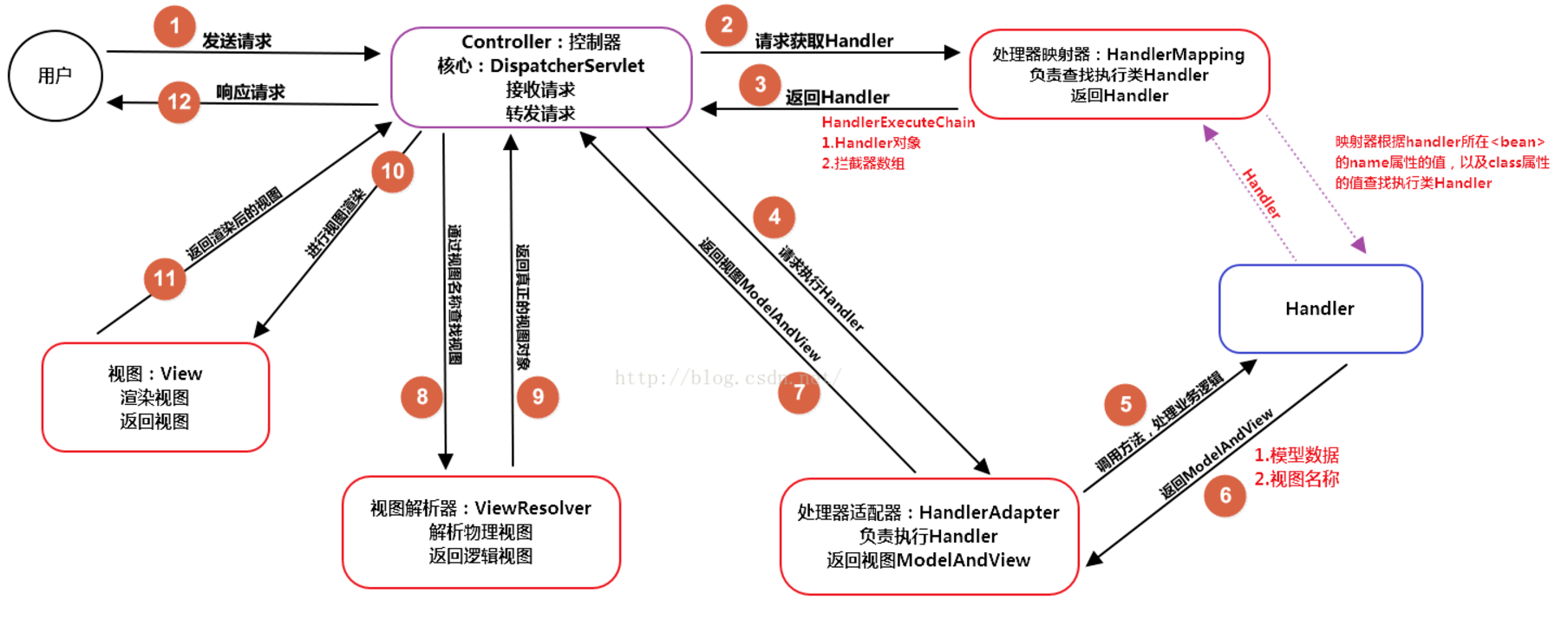
　　　　根据逻辑视图名来解析真正的视图。

第九步：试图解析器向前端控制器返回 view

第十步：前端控制器进行视图渲染

　　　　就是将模型数据（在 ModelAndView 对象中）填充到 request 域

第十一步：前端控制器向用户响应结果



组件解释：

1、前端控制器DispatcherServlet（不需要程序员开发）。

作用：接收请求，响应结果，相当于转发器，中央处理器。有了DispatcherServlet减少了其它组件之间的耦合度。

2、处理器映射器HandlerMapping（不需要程序员开发）。

　　作用：根据请求的url查找Handler。

3、处理器适配器HandlerAdapter（不需要程序员开发）。

　　作用：按照特定规则（HandlerAdapter要求的规则）去执行Handler。

4、处理器Handler（需要程序员开发）。

　　注意：编写Handler时按照HandlerAdapter的要求去做，这样适配器才可以去正确执行Handler

5、视图解析器ViewResolver（不需要程序员开发）。

　　作用：进行视图解析，根据逻辑视图名解析成真正的视图（view）

6、视图View（需要程序员开发jsp）。

　　注意：View是一个接口，实现类支持不同的View类型（jsp、freemarker、pdf…）

ps:不需要程序员开发的，需要程序员自己做一下配置即可。

说明：在springmvc的各个组件中，**处理器映射器、处理器适配器、视图解析器**称为springmvc的三大组件。需要用户开发的组件有：处理器、视图

以下组件通常使用框架提供的实现：

DispatcherServlet：前端控制器

用户请求到达前端控制器，它就相当于mvc模式中的C，dispatcherServlet是整个流程控制的中心，由它调用其它组件处理用户的请求，dispatcherServlet的存在降低了组件之间的耦合性。

HandlerMapping：处理器映射器；向前端控制器返回Handler对象或拦截器数组

HandlerMapping负责根据用户请求找到Handler即处理器，springmvc提供了不同的映射器实现不同的映射方式，例如：配置文件方式，实现接口方式，注解方式等。

Handler：处理器

Handler 是继DispatcherServlet前端控制器的后端控制器，在DispatcherServlet的控制下Handler对具体的用户请求进行处理。

由于Handler涉及到具体的用户业务请求，所以一般情况需要程序员根据业务需求开发Handler。

HandlAdapter：处理器适配器

通过HandlerAdapter对处理器进行执行，这是适配器模式的应用，通过扩展适配器可以对更多类型的处理器进行执行。

View Resolver：视图解析器

View Resolver负责将处理结果生成View视图，View Resolver首先根据逻辑视图名解析成物理视图名即具体的页面地址，再生成View视图对象，最后对View进行渲染将处理结果通过页面展示给用户。

View：视图

springmvc框架提供了很多的View视图类型的支持，包括：jstlView、freemarkerView、pdfView等。我们最常用的视图就是jsp。

一般情况下需要通过页面标签或页面模版技术将模型数据通过页面展示给用户，需要由程序员根据业务需求开发具体的页面。

------- 先有大概印象，知道大概，再去研究-----------

# MyBatis

## 1.jdbc操作数据库

通过上面的例子我们可以分析如下几点：

　　①、问题一：数据库连接，使用时就创建，使用完毕就关闭，这样会对数据库进行频繁的获取连接和关闭连接，造成数据库资源浪费，影响数据库性能。

　　　　设想解决：使用数据库连接池管理数据库连接

　　②、问题二：将 sql 语句硬编码到程序中，如果sql语句修改了，那么需要重新编译 Java 代码，不利于系统维护

　　　　设想解决：将 sql 语句配置到 xml 文件中，即使 sql 语句变化了，我们也不需要对 Java 代码进行修改，重新编译

　　③、问题三：在 PreparedStatement 中设置参数，对占位符设置值都是硬编码在Java代码中，不利于系统维护

　　　　设想解决：将 sql 语句以及占位符和参数都配置到 xml 文件中

　　④、问题四：从 resultset 中遍历结果集时，对表的字段存在硬编码，不利于系统维护

　　　　设想解决：将查询的结果集自动映射为 Java 对象

　　⑤、问题五：重复性代码特别多，频繁的 try-catch

　　　　设想解决：将其整合到一个 try-catch 代码块中

　　⑥、问题六：缓存做的很差，如果存在数据量很大的情况下，这种方式性能特别低

　　　　设想解决：集成缓存框架去操作数据库

　　⑦、问题七：sql 的移植性不好，如果换个数据库，那么sql 语句可能要重写

　　　　设想解决：在 JDBC 和 数据库之间插入第三方框架，用第三方去生成 sql 语句，屏蔽数据库的差异

## 2.MyBatis操作基于XML配置（使用maven构建项目）

数据库建表-》创建maven工程并pom配置-》jdbc.properties和mybatis-configuration.xml文件配置（数据库配置）-》与数据库表对应建立pojo类-》mapper文件，写操作数据库的方法CRUD-》测试类

1).数据库建表

2).创建java工程

3).jdbc.properties

4).mybatis-configuration.xml

加载数据库配置文件

environments -default

environment -id

transactionManager -type

dataSource -type

property -name -value

property

....

5).userMapper.xml

编写动态sql

select -id -parameterType -resultType

update

insert

delete

6).向mybatis-configuration.xml中注册usermapper.xml文件

<mappers>

<mapper resource="com/ys/mapper/userMapper.xml"/>

</mappers>

7).创建测试类

定义mybatis全局配置文件 mybatis-configuration.xml

加载配置文件

构建sqlSession的工厂

根据SqlSessionFactory产生session

增删该查。。。。

## MyBatis源码解读★★★★

### 初始化环境--见2操作

### 构建Configuration对象。

Configuration对象保存了mybatis的配置信息，主要包括mybatis-configuration.xml基础配置文件和mapper.xml映射配置文件。

①读取配置文件

InputStream inputStream = MybatisTest.class.getClassLoader()

.getResourceAsStream("mybatis-configuration.xml");

sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);

获取基础配置文件mybatis-configuration.xml的字节流

将该字节流对象作为build方法的参数传入进去，将mybatis-configuration.xml文件解析org.w3c.dom.Document对象，并将Document对象存储在XPathParser对象中便于后面解析

将Document对象转成Configuration对象

②初始化基础配置

**Properties全局参数**

**typeAliases别名**

<typeAliases>

<typeAlias type="com.ys.po.User" alias="user"/> </typeAliases>

类型别名是为 Java 类型设置一个短的名字。它只和 XML 配置有关，存在的意义仅在于用来减少类完全限定名的冗余。它还有一个 <package name="com.ys.po" /> 标签，在没有注解的情况下，会使用 Bean 的首字母小写的非限定类名来作为它的别名。

**Mapper映射器**

在 mybatis-configuration.xml 配置文件中有两个标签，一个是 <environments/> 用来配置数据源等信息。另一个就是 <mappers />标签了，用来进行 sql 文件映射。也就是说我们需要告诉 MyBatis 到哪里去找到这些语句。 Java 在自动查找这方面没有提供一个很好的方法，所以最佳的方式是告诉 MyBatis 到哪里去找映射文件。可以使用相对于类路径的资源引用， 或完全限定资源定位符（包括 file:/// 的 URL），或类名和包名等。

### 数据源的配置

<!-- 可以配置多个运行环境，但是每个 SqlSessionFactory 实例只能选择一个运行环境常用： 一、development:开发模式 二、work：工作模式 -->

<environments default="development">

<!--id属性必须和上面的default一样 -->

<environment id="development">

<!--使用JDBC的事务管理机制-->

<transactionManager type="JDBC" />

<dataSource type="POOLED">

<property name="driver" value="${jdbc.driver}" />

<property name="url" value="${jdbc.url}" />

<property name="username" value="${jdbc.username}" />

<property name="password" value="${jdbc.password}" />

</dataSource>

</environment>

</environments>

①mybatis三种数据源，

UNPOOLED：(不使用连接池）这个数据源的实现只是每次被请求时打开和关闭连接。虽然有点慢，但对于在数据库连接可用性方面没有太高要求的简单应用程序来说，是一个很好的选择。 不同的数据库在性能方面的表现也是不一样的，对于某些数据库来说，使用连接池并不重要，这个配置就很适合这种情形。

　　POOLED：(使用连接池）这种数据源的实现利用“池”的概念将 JDBC 连接对象组织起来，避免了创建新的连接实例时所必需的初始化和认证时间。

JNDI ：这个数据源的实现是为了能在如 EJB 或应用服务器这类容器中使用，容器可以集中或在外部配置数据源，然后放置一个 JNDI 上下文的引用

数据库连接池的理解：初始化一个池子，里面预先存放一定数量的资源。当需要使用该资源的时候，将该资源标记为忙状态；当该资源使用完毕后，资源池把相关的资源的忙标示清除掉，以示该资源可以再被下一个请求使用。

对应到上面数据库连接的问题，我们可以这样解决：先建立一个池子，里面存放一定数量的数据库连接。当需要数据库连接时，只需从“连接池”中取出一个，使用完毕之后再放回去。我们可以通过设定连接池最大连接数来防止系统无尽的与数据库连接。这样就能避免频繁的进行数据库连接和断开耗资源操作。

### 事务的配置

①<transactionManager type=”JDBC”/>

这个配置就是直接使用了 JDBC 的提交和回滚设置，它依赖于从数据源得到的连接来管理事务作用域。

②<transactionManager type=”MANAGED”/>

这个配置几乎没做什么。它从来不提交或回滚一个连接，而是让容器来管理事务的整个生命周期（比如 JEE 应用服务器的上下文）

**注意：和数据源配置一样，通常项目中我们不会单独使用 mybatis 来管理事务。比如选择框架 Spring +mybatis，这时候没有必要配置事务管理器， 因为 Spring 模块会使用自带的管理器来覆盖前面的配置。**

③初始化事务管理器

我们说事务（Transaction），一般是指要做的或所做的事情。在数据库中，事务具有如下四个属性：

　　原子性（atomicity）：一个事务是一个不可分割的工作单位，事务中包括的诸操作要么都做，要么都不做。

　　一致性（consistency）：事务必须是使数据库从一个一致性状态变到另一个一致性状态。一致性与原子性是密切相关的。

　　隔离性（isolation）：一个事务的执行不能被其他事务干扰。即一个事务内部的操作及使用的数据对并发的其他事务是隔离的，并发执行的各个事务之间不能互相干扰。

　　持久性（durability）：持久性也称永久性（permanence），指一个事务一旦提交，它对数据库中数据的改变就应该是永久性的。接下来的其他操作或故障不应该对其有任何影响。

　　这也就是常说的事务 ACID 特性。而在程序中，对于事务的操作通常是：

　　创建事务（create）

　　提交事务（commit）

　　回滚事务（rollback）

关闭事务（close）

MyBatis为了获取事务采用了工厂模式，定义了一个工厂接口，TransactionFactory，通过实现该工厂接口，mybatis提供了了两种不同的事务管理器，JDBC和MANAGED 这两种事务管理器的获取也采用了工厂模式。

### sql语句的执行流程

加载配置文件，产生SqlSessionFactory，通过SqlSessionFactory获取SqlSession对象

## 4.MyBatis操作基于注解

1)、创建MySQL数据库：mybatisDemo和表：user

2)、建立一个Java工程，并导入相应的jar包，具体目录如下

3)、在 MyBatisTest 工程中添加数据库配置文件 mybatis-configuration.xml

4)、定义表所对应的实体类

5)、定义操作 user 表的注解接口 UserMapper.java

6)、向 mybatis-configuration.xml 配置文件中注册 UserMapper.java 文件

7)、创建测试类

不需要userMapper.xml文件，创建定义操作user表的注解接口userMApper.java。

分别创建CRUD的抽象方法，通过注解编写SQL；测试时，先定义myBatis全局配置文件(mybatis-configuration.xml)并加载，构建SqlSessionFactory工厂，根据SqlSessionFactory工厂产生session；通过session获取userMapper接口，然后调用定义的CRUD方法，当需要插入数据时，先要创建POJO对象，然后调用其set方法，最后将该对象当做参数传入userMapper接口定义的方法中。

## 5.properties以及别名定义

1).将 数据库的配置语句写在 db.properties 文件中

2). mybatis-configuration.xml 中加载db.properties文件并读取

<properties resource="db.properties">

<property name="username" value="aaa"/>

</properties>

读取优先级：

1. 在 properties 内部自定义的属性值第一个被读取

② 然后读取 resource 路径表示文件中的属性，如果有它会覆盖已经读取的属性；如果 resource 路径不存在，那么读取 url 表示路径文件中的属性，如果有它会覆盖第一步读取的属性值

　③ 最后读取 parameterType 传递的属性值，它会覆盖已读取的同名的属性

3).定义别名

<typeAliases>

<typeAlias type="com.ys.po.User" alias="user"/>

</typeAliases>

## 动态SQL

需要用userMapper.xml来写动态SQL。

trim标记是一个格式化的标记，可以完成set或者是where标记的功能

动态SQL:if+set 语句

Choose(when,otherwise)

trim语句

<select id="selectUserByUsernameAndSex"resultType="user" parameterType="com.ys.po.User">

        select \* from user

        <!-- <where>

            <if test="username != null">

               username=#{username}

            </if>

            <if test="username != null">

               and sex=#{sex}

            </if>

        </where>  -->

        <trim prefix="where" prefixOverrides="and | or">

            <if test="username != null">

               and username=#{username}

            </if>

            <if test="sex != null">

               and sex=#{sex}

            </if>

        </trim>

    </select>

prefix：前缀

prefixoverride：去掉第一个and或者是or

suffix：后缀

suffixoverride：去掉最后一个逗号（也可以是其他的标记，就像是上面前缀中的and一样）

动态SQL: SQL 片段

## Mybatis缓存

mybatis的一级缓存:

　　MyBatis会在表示会话的SqlSession对象中建立一个简单的缓存，将每次查询到的结果结果缓存起来，

当下次查询的时候，如果判断先前有个完全一样的查询，会直接从缓存中直接将结果取出，返回给用户，

不需要再进行一次数据库查询了。

　　MyBatis会在一次会话的表示----一个SqlSession对象中创建一个本地缓存(local cache)，

对于每一次查询，都会尝试根据查询的条件去本地缓存中查找是否在缓存中，如果在缓存中，

就直接从缓存中取出，然后返回给用户；否则，从数据库读取数据，将查询结果存入缓存并返回给用户。

　　一级缓存是SqlSession级别的缓存。在操作数据库时需要构造 sqlSession对象，在对象中有一个(内存区域)数据结构（HashMap）用于存储缓存数据。不同的sqlSession之间的缓存数据区域（HashMap）是互相不影响的。

　　一级缓存的作用域是同一个SqlSession，在同一个sqlSession中两次执行相同的sql语句，第一次执行完毕会将数据库中查询的数据写到缓存（内存），第二次会从缓存中获取数据将不再从数据库查询，从而提高查询效率。当一个sqlSession结束后该sqlSession中的一级缓存也就不存在了。Mybatis默认开启一级缓存。

　　二级缓存是mapper级别的缓存，多个SqlSession去操作同一个Mapper的sql语句，多个SqlSession去操作数据库得到数据会存在二级缓存区域，多个SqlSession可以共用二级缓存，二级缓存是跨SqlSession的。

　　二级缓存是多个SqlSession共享的，其作用域是mapper的同一个namespace，不同的sqlSession两次执行相同namespace下的sql语句且向sql中传递参数也相同即最终执行相同的sql语句，第一次执行完毕会将数据库中查询的数据写到缓存（内存），第二次会从缓存中获取数据将不再从数据库查询，从而提高查询效率。Mybatis默认没有开启二级缓存需要在setting全局参数中配置开启二级缓存。

如果缓存中有数据就不用从数据库中获取，大大提高系统性能。

## Statement和PreparedStatement

Statement执行不带参数的简单SQL语句,并返回它所生成结果的对象,每次执行SQL语句时,数据库都要编译该sql语句

PreparedStatement用来执行**带参数**的预编译的SQL语句

PreparedStatement的优点:

①效率高.

使用PreparedStatement执行SQL命令时,命令会被数据库编译和解析,并放到命令缓冲区.以后每当执行同一个PreparedStatement对象时,预编译的命令就可以重复使用

②代码可读性和可维护性好

③安全性好.

使用PreparedStatement可以防止SQL注入.

SQL注入如:

select \* from user where name = ‘aa’ and password = ‘bb’ or 1=1

## SqlSessionFactory

在使用Mybatis框架时，首先 就是要产生SqlSessionFactory类的实例(相当于产生连接池)，通过调用SqlSessionFactoryBuilder类的实例的builder方法来完成。

## ------------------------------------------------------mybatis详解7

# SpringBoot

## 介绍

先编译后运行就是静态，不需要编译即可运行即是动态语言。动态语言Ruby,Node.js,java繁多的配置增加了工作量，SpringBoot习惯优于配置，

## 配置

@SpringBootApplication：Spring Boot项目的核心注解，主要目的是开启自动配置。它其实是一个组合注解，主要包括：①SpringBootConfiguration，这是springboot项目的配置注解，也是一个组合注解，在springboot项目中，推荐使用SpringBootConfiguration替代Configuration②EnableAutoCOnfiguration,启动自动配置，该注解会使springboot根据项目中依赖的jar包自动配置项目的配置项，③ComponentScan默认扫描SpringBootApplication所在类的同级目录以及它的子目录；

@Configuration：这是一个配置Spring的配置类；

@Controller：标明这是一个SpringMVC的Controller控制器

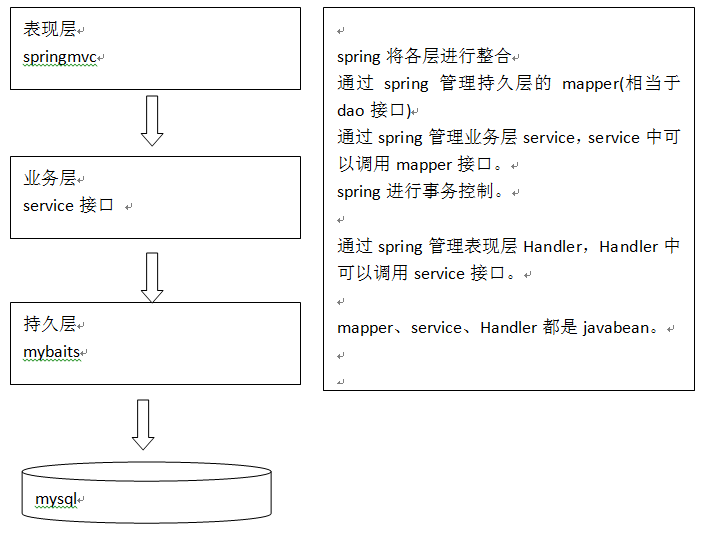
如果我们不需要Spring Boot自动配置，想关闭某一项的自动配置，该如何设置呢？

比如：我们不想自动配置Redis，想手动配置

@SpringBootApplication(exclude={RedisAutoConfiguration.class})

### SSM

## 1.ssm整合



①、表现层，也就是 Controller,由 SpringMVC 来控制，而SpringMVC 是Spring 的一个模块，故不需要整合。

　　②、业务层，也就是 service，通常由 Spring 来管理 service 接口，我们会使用 xml 配置的方式来将 service 接口配置到 spring 配置文件中。而且事务控制一般也是在 service 层进行配置。

　　③、持久层，也就是 dao 层，而且包括实体类，由 MyBatis 来管理，通过 spring 来管理 mapper 接口，使用mapper的扫描器自动扫描mapper接口在spring中进行注册。

很明显，spring 在三大框架的整合中占据至关重要的地位，类似于一个大管家，将 MyBatis 和 SpringMVC 揉合在一起。

## 整合dao层

1. db.properties
2. mybatis全局配置文件 mybatis-configuration.xml
3. 配置 Spring 文件

Sprig-dao.xml

第一步 配置数据源

加载db.properties

1. 创建SqlSessionFactory

数据库连接池,加载mybatis全局配置文件

1. 配置mapper扫描器，

\*接口类名和映射文件必须同名

\*接口类和映射文件必须在同一目录下

\*映射文件namespace名称必须是接口的全类路径名

\*接口的方法名必须和映射statement的id一致

1. 根据逆向工程生成 po 类以及 mapper 文件

在实际开发中，我们基本上能对mybatis应用自如了，但是我们发现了一个问题，所有操作都是围绕着po类，xxxMapper.xml文件，xxxMapper接口等文件来进行的。如果实际开发中数据库的表特别多，那么我们需要手动去写每一张表的po类，xxxMapper.xml，xxxMapper.java文件，这显然需要花费巨大的精力，而且可能由于表字段太多，写错了而不知道也是可能的.

所以我们在实际开发中，一般使用逆向工程方式来自动生成所需的文件。

3.整合service

### 工作

## 1.工具使用技巧

1. alt+《》回到上一个操作文件，下一个操作文件
2. Ctrl+L 搜索列号
3. 注意：根据阿里巴巴的最新编码规范，实体类中的属性不要写基本数据类型，必须使用包装类型。比如 int 类型的 id，我们应该写成其包装类 Integer 类型。

原因：比如显示成交总额涨跌情况，即正负 x %， x 为基本数据类型，调用的 RPC 服务，调用不成功时，返回的是默认值，页面显示为 0%，这是不合理的，应该显示成中划线。所以包装数据类型的 null 值，能够表示额外的信息，如：远程调用失败，异常退出。因为基本数据类型有默认值，会影响调用。。。

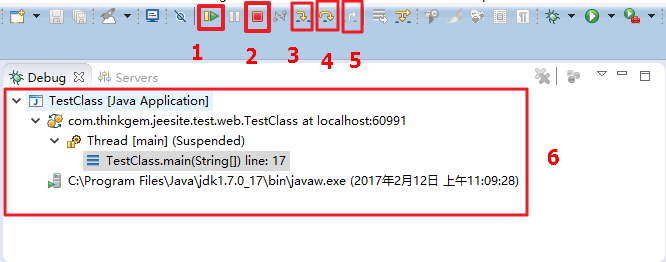
## 2.build path和java compiler level

build path的JDK版本是你开发的时候编译器需要使用到的，就是你在eclipse中开发代码，给你提示报错的，编译的过程；java compiler compliance level中配置的编译版本号，这个编译版本号的作用是，你这个项目将来开发完毕之后，要放到服务器上运行，那个服务器上JDK的运行版本。同事的问题就是，build path中配置1.7的JDK，java compiler compliance level中配置的1.7，但是服务器上是1.6的JDK，就报了那个错误，说是编译所用的jdk（1.7）比运行所用的jdk（1.6）高了，这是错误的。【不满足于问题的解决，要追根知底】

## 3.Debug

F8 运行到当前方法的下一个断点,或者当前方法调用的子孙方法的下一个断点或如果没有下一个断点则程序直接运行完毕;

F5 进入当前断点所在子方法(不常用)F6 进入当前断点所在方法的下一步(常用)



1.表示当前实现继续运行直到下一个断点，快捷键为F8。

2.表示打断整个进程

3.表示进入当前方法，快捷键为F5。

4.表示运行下一行代码，快捷键为F6。

5.表示退出当前方法，返回到调用层，快捷键为F7。

6.表示当前线程的堆栈，从中可以看出在运行哪些代码，并且整个调用过程，以及代码行号

## 4.路径问题

"./"：代表目前所在的目录。

"../"：代表上一层目录。可叠加使用

以"/"开头：代表根目录。

# Maven

## 1.创建

groupid和artifactId被统称为“坐标”是为了保证项目唯一性而提出的，如果你要把你项目弄到maven本地仓库去，你想要找到你的项目就必须根据这两个id去查找。

groupId一般分为多个段，这里只说两段，第一段为域，第二段为公司名称。com.ys

ArtifactID就是项目的唯一的标识符，实际对应项目的名称，就是项目根目录的名称。Hellomaven

依照这个设置，你的包结构最好是com.ys.hellomaven打头的，如果有个StudentDao，它的全路径就是com.ys.hellomaven.dao.StudentDao

## 2.常用命令

Mvn compile 编译，将源程序编译成class字节码文件

Mvn test 测试并生成此时报告

Mvn clean 将以前编译得到的旧的class字节码文件删除

Mvn package 打包，动态web工程打war包，java工程打jar包

Mvn install 将项目生成jar包放在仓库中，以便别的模块调用

## 3.maven坐标

俗称 gav：使用下面三个向量子仓库中唯一定位一个 Maven 工程

　在项目中的 pom.xml 文件中，我们可以看到下面gav的定义：

　　　　1、groupid:公司或组织域名倒序

　　　　　　<groupid>com.ys.maven</groupid>

　　　　2、artifactid:模块名，也是实际项目的名称

　　　　　　<artifactid>Maven\_05</artifactid>

　　　　3、version:当前项目的版本

　　　　　　<version>0.0.1-SNAPSHOT</version>

## 4.依赖配置

　 ①、dependencies：一个 pom.xml 文件中只能存在一个这样的标签。用来管理依赖的总标签。

　　②、dependency:包含在dependencies标签中，可以有无数个，每一个表示一个依赖

　　③、groupId,artifactId和version：依赖的基本坐标，对于任何一个依赖来说，基本坐标是最重要的，Maven根据坐标才能找到需要的依赖。

　　④、type：依赖的类型，对应于项目坐标定义的packaging。大部分情况下，该元素不必声明，其默认值是jar。

　　⑤、scope：依赖的范围，默认值是 compile。后面会进行详解。

　　⑥、optional：标记依赖是否可选。

⑦、exclusions：用来排除传递性依赖，后面会进行详细介绍。

1. 依赖范围scope



1. 编译依赖范围（compile），该范围就是默认依赖范围，此依赖范围对 于编译、测试、运行三种classpath都有效，举个简单的例子，假如项目中有spring-core的依赖，那么spring-core不管是在编译，测试，还是运行都会被用到，因此spring-core必须是编译范围（构件默认的是编译范围，所以依赖范围是编译范围的无须显示指定）
2. 测试依赖范围(test)，顾名思义就是针对于测试的，使用此依赖范围的依赖，只对测试classpath有效，在编译主代码和项目运行时，都将无法使用该依赖，最典型的例子就是 Junit, 构件在测试时才需要，所以它的依赖范围是测试，因此它的依赖范围需要显示指定为<scope>test</scope> ,当然不显示指定依赖范围也不会报错，但是该依赖会被加入到编译和运行的classpath中,造成不必要的浪费 。
3. 已提供依赖范围(provided),使用该依赖范围的maven依赖，只对编译和测试的classpath有效，对运行的classpath无效，典型的例子就是servlet-api， 编译和测试该项目的时候需要该依赖，但是在运行时，web容器已经提供的该依赖，所以运行时就不再需要此依赖，如果不显示指定该依赖范围，并且容器依赖的版本和maven依赖的版本不一致的话，可能会引起版本冲突，造成不良影响。
4. 运行时依赖范围(runtime),使用该依赖范围的maven依赖，只对测试和运行的classpath有效，对编译的classpath无效，典型例子就是JDBC的驱动实现，项目主代码编译的时候只需要JDK提供的JDBC接口，只有在测试和运行的时候才需要实现上述接口的具体JDBC驱动。
5. 系统依赖范围（system）,该依赖与classpath的关系与 provided依赖范围完全一致，但是系统依赖范围必须通过配置systemPath元素来显示指定依赖文件的路径，此类依赖不是由maven仓库解析的，而且往往与本机系统绑定，可能造成构件的不可移植，因此谨慎使用
6. 导入依赖范围(import),该依赖范围不会对三种classpath产生影响，该依赖范围只能与dependencyManagement元素配合使用，其功能为将目标pom文件中dependencyManagement的配置导入合并到当前pom的dependencyManagement中。有关dependencyManagement的功能请了解maven继承特性。

## 生命周期

　①、Clean Lifecycle:在进行真正的构建之前进行一些清理工作

　②、Default Lifecycle：构建的核心部分，编译、测试、打包、安装、部署等等。

　③、Site Lifecycle：生成项目报告，站点，发布站点。

　这三个都是相互独立的。你可以仅仅调用 clean 来清理工作目录，仅仅调用 site 来生成站点。当然，也可以直接运行 mvn claen install site 运行所有这三套生命周期。mvn 的各个生命周期步骤都是依赖插件来完成的，

## 创建Web工程以及插件原理

1. .配置编译插件

　一般我们创建一个 Maven 工程，就算指定了 JDK 的版本，但是你执行 update project 操作，一般 Maven 工程会自动恢复到默认的 JDK 版本，有可能是1.4，有可能是1.5（和 Maven 版本有关）。指定jdk版本的配置

<build>

    <plugins>

        <!-- 编译插件，指定 JDK 的版本为1.7 -->

        <plugin>

            <groupId>org.apache.maven.plugins</groupId>

            <artifactId>maven-compiler-plugin</artifactId>

            <configuration>

                <source>1.7</source>

                <target>1.7</target>

                <encoding>UTF-8</encoding>

            </configuration>

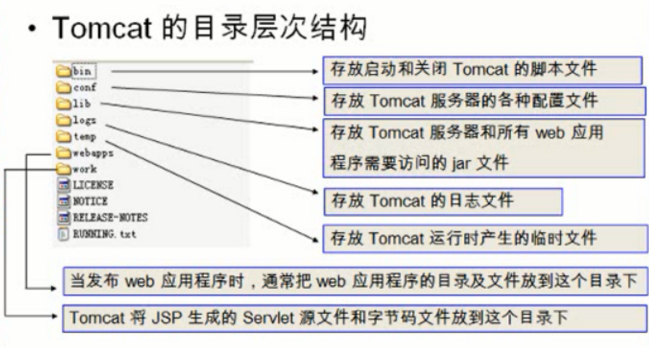
        </plugin>

    </plugins>

</build>

1. .先创建mavenJAVA工程，然后Project Facts转换成web工程，注意webapps路径修改，在src/main/webapps目录下

## Tomcat

1. tomcat目录
2. 部署项目的第一种方法

将war包直接放在webapps目录下，双击startup.bat启动，tomcat会自动解压war

1. 部署项目的第二种方法（修改 conf/server.xml 文件 ）

①、打开tomcat下conf/server.xml，在<Host> </Host>标签之间输入项目配置信息

<Context path="/WebProject" docBase="D:/WebProject" reloadable="true" />

　　path:浏览器访问时的路径名

　　docBase:web项目的WebRoot所在的路径，注意是WebRoot的路径，不是项目的路径。其实也就是编译后的项目

　　reloadble:设定项目有改动时，tomcat是否重新加载该项目

②、双击 startup.bat，启动 tomcat 服务器，然后在浏览器输入访问的项目名称路径

1. 部署项目的第三种方法
2. 进入到 apache-tomcat-7.0.52\conf\Catalina\localhost 目录，新建一个 项目名.xml 文件
3. 在那个新建的 xml 文件中，增加下面配置语句（和上面的是一样的,但是不需要 path 配置，加上也没什么用）

<Context docBase="D:/WebProject" reloadable="true" />

# XML

XML即可扩展标记语言（eXtensible Markup Language）。标记是指计算机所能理解的信息符号，通过此种标记，计算机之间可以处理包含各种信息的文章等。

通俗的讲，XML文件一般用来保存有关系的数据。除此之外，在程序开发中，我们通常用来做各种框架的配置文件。

1. XML 文件由以下几个部分组成：
2. 文档声明 ②、元素 ③、属性 ④、注释 ⑤、CDATA区、特殊字符 ⑥、处理指令（processing intruction)

## 文档说明

<?xml version="1.0" encoding="utf-8" standalone="no" ?>

元素

对于 XML 标签中出现的所有空格和换行，XML 解析都会当做标签内容进行处理。比如下面两个是不一样的

注意：由于在 XML 中，空格和换行都会作为原始内容被处理，所以在编写 XML 文件的时候使用换行和缩进等方式来让原文件中的内容清晰可读的良好习惯可能要被迫改变。这和 HTML 标签的书写是有点区别的。

　⑤、元素的命名规范：

　　　　1、区分大小写，例如：<P>和<p>是不一样的

　　　　2、不能以数字或下划线“-”开头

　　　　3、不能以 xml（或XML,Xml）作为开头

　　　　4、不能包含空格

　　　　5、名称中间不能包含冒号“：”

三、属性

　　①、一个元素可以有多个属性，每个属性都有它自己的名称和取值，比如：

　　　　<input name="tom" />

　　②、属性值一定要用双引号或单引号引起来

　　③、属性的命名规范和元素的命名规范一样

　　④、XML 文件中，元素属性所代表的信息，也可以改为用子元素来表示，比如：

　　　　<input name="tom" />

　　　　可以写为：

　　　　<input>

　　　　　　<name>tom</name>

　　　　</input>

## XMl的解析方式

解析 XML 文件一般来说有两种底层形式，一种是基于树的结构来解析的称为DOM；另一种是基于事件流的形式称为Sax。而在这两种解析方式的基础上，基于底层api的更高级封装解析器也应用而生，比如面向Java的 JDom和 Dom4J。

1).DOM(Document Object Model)

　　DOM 是用与平台和语言无关的方式表示 XML 文档的官方 W3C 标准。DOM 是以层次结构组织的节点或信息片断的集合。这个层次结构允许开发人员在树中寻找特定信息。分析该结构通常需要加载整个文档和构造层次结构，然后才能做任何工作。由于它是基于信息层次的，因而 DOM 被认为是基于树或基于对象的。

　　优点：

　　　　①、整个 Dom 树都加载到内存中了，所以允许随机读取访问数据。

　　　　②、允许随机的对文档结构进行增删。

　　缺点：

　　　　①、整个 XML 文档必须一次性解析完，耗时。

　　　　②、整个 Dom 树都要加载到内存中，占内存。

适用于：文档较小，且需要修改文档内容

1. .Sax(Simple API for XML)

SAX处理的特点是基于事件流的。分析能够立即开始，而不是等待所有的数据被处理。而且，由于应用程序只是在读取数据时检查数据，因此不需要将数据存储在内存中。这对于大型文档来说是个巨大的优点。事实上，应用程序甚至不必解析整个文档；它可以在某个条件得到满足时停止解析。sax分析器在对xml文档进行分析时,触发一系列的事件,应用程序通过事件处理函数实现对xml文档的访问，因为事件触发是有时序性的,所以sax分析器提供的是一种对xml文档的顺序访问机制,对于已经分析过的部分,不能再重新倒回去处理.此外，它也不能同时访问处理2个tag，sax分析器在实现时,只是顺序地检查xml文档中的字节流,判断当前字节是xml语法中的哪一部分,检查是否符合xml语法并且触发相应的事件.对于事件处理函数的本身,要由应用程序自己来实现. SAX解析器采用了基于事件的模型，它在解析XML文档的时候可以触发一系列的事件，当发现给定的tag的时候，它可以激活一个回调方法，告诉该方法制定的标签已经找到。

　　优点：

　　　　①、访问能够立即进行，不需要等待所有数据被加载。

　　　　②、只在读取数据时检查数据，不需要保存在内存中

　　　　③、不需要将整个数据都加载到内存中，占用内存少

　　　　④、允许注册多个Handler,可以用来解析文档内容，DTD约束等等。

　　缺点：

　　　　①、需要应用程序自己负责TAG的处理逻辑（例如维护父/子关系等），文档越复杂程序就越复杂。

　　　　②、单向导航，无法定位文档层次，很难同时访问同一文档的不同部分数据，不支持XPath。

　　　　③、不能随机访问 xml 文档，不支持原地修改xml。

　　适用于：文档较大，只需要读取文档数据。

1. JDOM

JDOM是处理xml的纯java api.使用具体类而不是接口。JDOM具有树的遍历,又有SAX的java规则。

　　JDOM与DOM主要有两方面不同。首先，JDOM仅使用具体类而不使用接口。这在某些方面简化了API，但是也限制了灵活性。第二，API大量使用了Collections类，简化了那些已经熟悉这些类的Java开发者的使用。

　　JDOM自身不包含解析器。它通常使用SAX2解析器来解析和验证输入XML文档（尽管它还可以将以前构造的DOM表示作为输入）。它包含一些转换器以将JDOM表示输出成SAX2事件流、DOM模型或XML文本文档。JDOM是在Apache许可证变体下发布的开放源码。

　　优点：

　　　　①、使用具体类而不是接口，简化了DOM的API。

　　　　②、大量使用了Java集合类，方便了Java开发人员。

　　缺点：

　　　　①、不能处理大于内存的文档.

　　　　②、API 简单，没有较好的灵活性

1. DOM4J

虽然DOM4J代表了完全独立的开发结果，但最初，它是JDOM的一种智能分支。它合并了许多超出基本XML文档表示的功能，包括集成的XPath，支持XML Schema支持以及用于大文档或流化文档的基于事件的处理。它还提供了构建文档表示的选项，它通过DOM4J API和标准DOM接口具有并行访问功能。从2000下半年开始，它就一直处于开发之中。

为支持所有这些功能，DOM4J使用接口和抽象基本类方法。DOM4J大量使用了API中的Collections类，但是在许多情况下，它还提供一些替代方法以允许更好的性能或更直接的编码方法。直接好处是，虽然DOM4J付出了更复杂的API的代价，但是它提供了比JDOM大得多的灵活性。

在添加灵活性、XPath集成和对大文档处理的目标时，DOM4J的目标与JDOM是一样的：针对Java开发者的易用性和直观操作。它还致力于成为比JDOM更完整的解决方案，实现在本质上处理所有Java/XML问题的目标。在完成该目标时，它比JDOM更少强调防止不正确的应用程序行为。

DOM4J是一个非常非常优秀的Java XML API，具有性能优异、功能强大和极端易用使用的特点，同时它也是一个开放源代码的软件。如今你可以看到越来越多Java软件都在使用DOM4J来读写XML，特别值得一提的是连Sun的JAXM也在用DOM4J。

　　优点：

　　　　①、大量使用了Java集合类，方便Java开发人员，同时提供一些提高性能的替代方法。

　　　　②、支持XPath。查找节点特别快

　　　　③、灵活性高。

　　缺点：

　　　　①、大量的使用了接口，API复杂，理解难。

　　　　②、移植性差。

注：XPath是一门在 XML 文档中查找信息的语言。

**比较： 1、 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。**

**2、JDOM和DOM在性能测试时表现不佳，在测试10M文档时内存溢出，但可移植。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）。**

**3、SAX表现较好，这要依赖于它特定的解析方式－事件驱动。一个SAX检测即将到来的XML流，但并没有载入到内存（当然当XML流被读入时，会有部分文档暂时隐藏在内存中）。**

**综上所述：如果XML文档较大且不考虑移植性问题建议采用DOM4J；如果XML文档较小则建议采用JDOM；如果需要及时处理而不需要保存数据则考虑SAX。**

# 报错

1. The input type of the launch configuration does not exist。

test下的包全部变成了文件夹

解决：选中test，右键，选择Build Path---User as Source Folder~

1. config配置文件夹要buildpath。**在build path下的类可以被编译运行，在build path下的配置文件可以被类以相对路径直接读**写。

# 项目组分享交流

## 线程安全

### 基本概念(线程、进程。。。。)

**进程**：进程是具有一定独立功能的程序关于某个数据集合上的一次运行活动,

进程是系统进行资源分配和调度的一个独立单位.

**线程**：是进程的一个实体,是CPU调度和分派的基本单位,它是比进程更小的能独立运行的基本单位.进程具有独立的内存单元，线程共享内存。

**线程安全**：当多个线程访问某个资源时，不管通过怎样的调用方式，或者这些线程如何交替执行，我们在主程序中不需要去做任何的同步，这个类的结果行为都是我们设想的正确行为，那么我们就可以说这个类是线程安全的。

线程安全问题都是由局部变量及静态变量引起的。若线程对资源只有读操作都是安全的。

### 以下是线程安全的：

1)常量是线程安全的，因为只有读操作2)每次调用方法前都新建一个实例是线程安全的，因为不会访问共享的资源3).局部变量是线程安全的，因为每执行一个方法，都会在独立的空间创建局部变量，他不是共享的资源，局部变量包括参数变量和方法内变量。

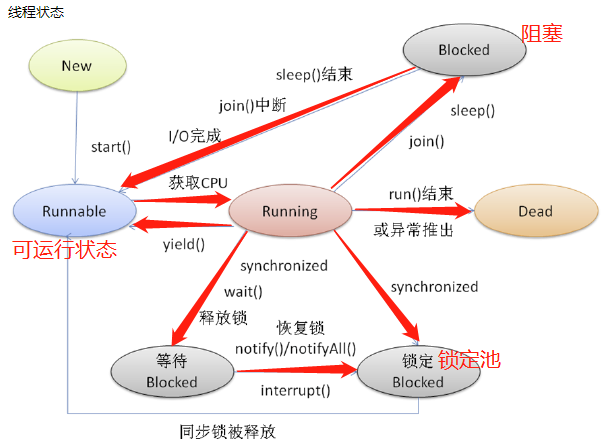
### 3.有无状态对象

有状态就是有数据存储功能。有状态对象(Stateful Bean)就是有实例变量的对象，可以保存数据库，是非线程安全的。

无状态对象：就是一次操作，不能保存数据，无状态对象(Stateless Bean)，就是没有实例变量的对象 .不能保存数据，是不变类，是线程安全的。

Spring中的状态和无状态。无状态的Bean适合用不变模式，技术就是单例模式(**只有一个实例**)，这样可以共享实例，提高性能。有状态的Bean，多线程环境下不安全，那么适合用Prototype原型模式(**访问一次创建一个实例**)。默认情况下，从Spring bean工厂所取得的实例为singleton（scope属性为singleton）,容器只存在一个共享的bean实例。**有状态的bean都使用prototype作用域，而对无状态的bean则应该使用singleton作用域**。

### 4.线程状态



Sleep()方法：在指定时间内让当前正在执行的线程暂停执行，进入阻塞状态，但不会释放“锁标志”。

Wait()方法：Notify()和notifyAll()这三个方法用于协调多个线程对共享数据的存取，所以必须在synchronized语句块内使用。synchronized关键字用于保护共享数据，阻止其他线程对共享数据的存取，但是这样程序的流程就很不灵活了，如何才能在当前线程还没退出synchronized数据块时让其他线程也有机会访问共享数据呢？wait()方法使当前线程暂停执行并释放对象锁标示，让其他线程可以进入synchronized数据块，当前线程被放入对象等待池中。当调用notify()方法后，将从对象的等待池中移走一个任意的线程并放到锁标志等待池中，只有锁标志等待池中线程能够获取锁标志；如果锁标志等待池中没有线程，则notify()不起作用。

notifyAll()则从对象等待池中移走所有等待那个对象的线程并放到锁标志等待池中。这三个方法都是java.lang.Object的方法。

yield()方法：yield意味着放手，放弃，投降。一个调用yield()方法的线程告诉虚拟机它乐意让其他线程占用自己的位置。这表明该线程没有在做一些紧急的事情。注意，这仅是一个暗示，并不能保证不会产生任何影响。

Join()方法：方法使调用该方法的线程在此之前执行完毕，也就是等待该方法的线程执行完毕后再往下继续执行。在很多情况下，主线程生成并起动了子线程，如果子线程里要进行大量的耗时的运算，主线程往往将于子线程之前结束，但是如果主线程处理完其他的事务后，需要用到子线程的处理结果，也就是主线程需要等待子线程执行完成之后再结束，这个时候就要用到join()方法了。

### 5.线程安全的几种方式

1)synchronized

2)lock 是jdk1.6以后引入，属于可操作性锁，可以手动获取和释放锁，

lock.lock();//获取锁对象

lock.unlock();//释放锁对象

lock.tryLock();//有一个boolean返回值

Lock在获取锁的时候，如果拿不到锁，就一直处于等待状态，直到拿到锁，但是tryLock()却不是这样的，tryLock是有一个Boolean的返回值的，如果没有拿到锁，直接返回false，停止等待，它不会像Lock()那样去一直等待获取锁。

### 6.线程创建

6.1继承Thread

6.2实现Runnable接口。

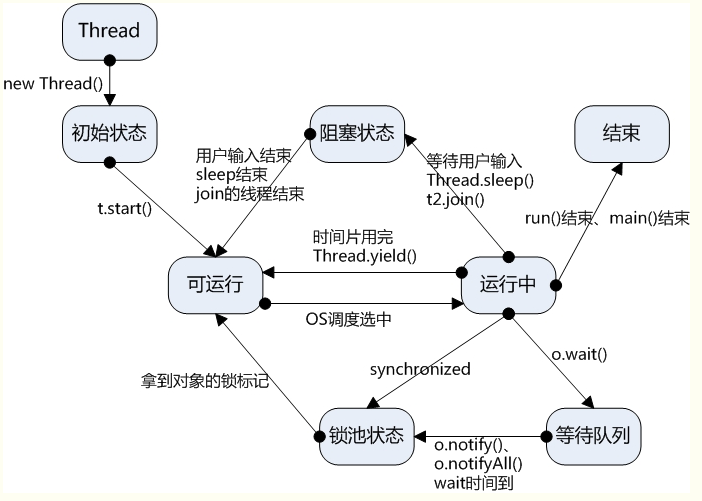
实现Runnable接口比继承Thread类所具有的优势：

1）：适合多个相同的程序代码的线程去处理同一个资源

2）：可以避免java中的单继承的限制

3）：增加程序的健壮性，代码可以被多个线程共享，代码和数据独立

4）：线程池只能放入实现Runable或callable类线程，不能直接放入继承Thread的类



## B、数据库设计

### 1.数据库设计三大范式(3NF)

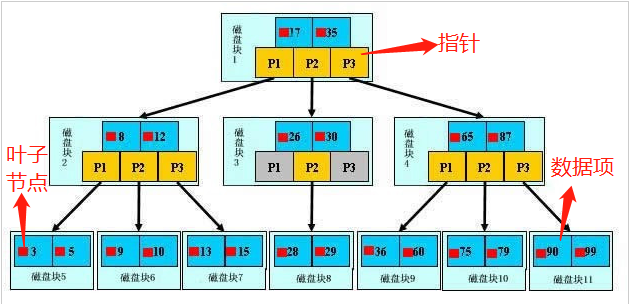
1NF:**确保每列都不可再分，每列都具备原子特性**；例如地址要拆分为：国家、省、市和详细地址。

2NF：**一行数据只做一件事，每一行的数据只能与其中一列相关**；例如一个表有订单编号、房间号、联系人、联系电话、身份证号，这样设计很不合理，不符合2NF，因为联系电话和身份证与联系人相关，房间号、联系人和订单编号相关！若一个人同时定多个房间，会造成数据冗余。应该拆分为两个表：订单表(订单编号、房间号、联系人编号)和用户表(身份证、电话和姓名等)。

3NF：**数据不能存在传递关系，每个属性都跟主键有直接关系而不是间接关系**。例如Student表(学号、姓名、性别、年龄、所在院校、院校地址、院校电话)，这种表结构不符合3NF和2NF，应该拆分为学生表(学号、性别、年龄、所在院校)和院校表(院校名字、院校地址和院校电话)

### 2.索引以及聚集索引和非聚集索引

索引的数据结构：在读取数据时，磁盘IO是非常昂贵的操作，寻找一种数据结构把磁盘IO次数控制在一个很小的数量级最好是常数数量级-》高度可控的多路搜索树-》B+树



**解释**：如磁盘块1包含数据项17和35，包含指针P1、P2、P3，P1表示小于17的磁盘块，P2表示在17和35之间的磁盘块，P3表示大于35的磁盘块。真实的数据存在于叶子节点即3、5、9、10、13、15、28、29、36、60、75、79、90、99。非叶子节点只不存储真实的数据，只存储指引搜索方向的数据项，如17、35并不真实存在于数据表中。

**例子**：如图所示，如果要查找数据项29，那么首先会把磁盘块1由磁盘加载到内存，此时发生一次IO，在内存中用二分查找确定29在17和35之间，锁定磁盘块1的P2指针，内存时间因为非常短（相比磁盘的IO）可以忽略不计，通过磁盘块1的P2指针的磁盘地址把磁盘块3由磁盘加载到内存，发生第二次IO，29在26和30之间，锁定磁盘块3的P2指针，通过指针加载磁盘块8到内存，发生第三次IO，同时内存中做二分查找找到29，结束查询，总计三次IO。真实的情况是，3层的b+树可以表示上百万的数据，如果上百万的数据查找只需要三次IO，性能提高将是巨大的，如果没有索引，每个数据项都要发生一次IO，那么总共需要百万次的IO，显然成本非常非常高。

在数据库中，B+树的高度一般都在2~4层。

叶子节点存放的即为整张表的记录数据，余江聚簇索引的叶子结点成为数据页，

**聚集索引**：指索引项的排序方式和表中数据记录排序方式一致的索引也就是说聚集索引的顺序就是数据的物理存储顺序。比如字典的“拼音”查汉字就是聚集索引，“偏旁部首”查汉字就是非聚集索引，因为正文中的字都是按照拼音排序的。聚集索引会降低inert、update操作的性能，就像数组的插入一样。

**非聚集索引**：索引吮吸与物理存储顺序不同。

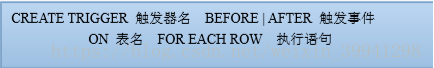
**聚簇索引**：聚簇索引并不是一种单独的索引类型，而是一种数据存储方法，“聚簇”表示数据行和相邻的键值紧凑的存储在一起，因为无法同时把数据行放在两个不同的地方，所以一个表只有一个聚簇索引。可以把相关数据保存在一起，就好像在操场上战队，一个院系一个院系的站在一起，这样要找到一个人，就先找到他的院系，然后在他的院系里找到他就行了，而不是把学校里的所有人都遍历一遍；数据访问更快。

### 3.B树、B-树和B+树

### 4.视图和表

数据库中的数据都是存储在表中的，视图只是一个或多个表依照某个条件组合而成的结果集，一般可以进行update/insert/delete等修改表中的数据，而事务只能进行select操作；表示物理存在的，视图是虚拟的内存表。表是内模式，视图是外模式

1. 触发器



触发器中不能进行任何事务操作，任何对被触发表进行操作的事务都将失败。

### 数据库设计技巧

1. 对象名不要用驼峰规则，全部大写用下划线连接
2. 不要使用社会保障号码（SSN）或身份证号码（ID）选作主键，第一是隐私(不合法)，第二是不要使用手工输入的键作为主键，因为一旦出错，只能删除整个记录2)3)4)5)6)7)8)9)

### MarkDown

## ApacheCommons库

## 类设计

## 常用日志框架

## 设计模式

## 算法设计

## JVM

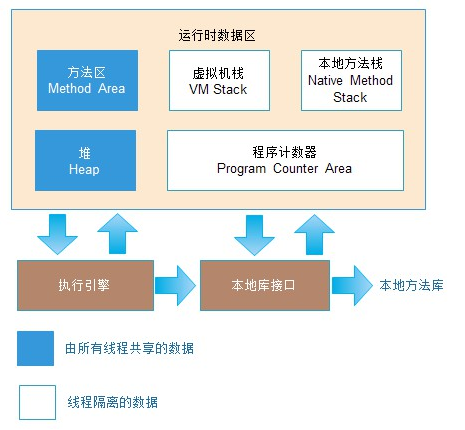
### 1.基本概念

**内存泄漏(memory leak)**：是指程序在申请内存后，无法释放已经申请的内存空间，一次内存泄露危害可以忽略，但内存泄露堆积后果很严重，无论多少内存,迟早会被占光。

**内存溢出(out of memory)**:是指程序在申请内存时，没有足够的空间供其使用；比如申请了一个integer,但给它存了long才能存下的数，那就是内存溢出。

**栈**：栈满时再做进栈必定产生空间溢出，叫上溢，栈空时再做退栈也产生空间溢出，称为下溢。就是分配的内存不足以放下数据项序列,称为内存溢出.

### 2.运行时数据区域(线程隔离的数据也就是线程私有的)



**2.1程序计数器**：是一个较小的内存空间，可以看做是当前线程所执行的字节码的行号指示器。字节码解释器工作时通过改变这个计数器的值来选取下一条需要执行的字节码指令，分支、循环、跳转、异常处理、线程恢复等功能都需要依赖这个计数器来完成。另外，为了线程切换后可以恢复到正常的执行位置，每条线程都需要一个独立的程序计数器，各线程之间计数器互不影响，独立存储，称这类内存区域为“线程私有”的内存。

**作用**：

1).字节码接收器通过改变程序计数器来依次读取指令，从而实现代码的流程控制，如顺序执行，选择，循环和异常处理。

2).在多线程的情况下，程序计数器用于记录当前线程执行的位置，从而当线程被切换回来的时候能够知道该线程上次运行到哪儿了。程序计数器是唯一不会出现 OutOfMemoryError 的内存区域，它的生命周期随着线程的创建而创建，随着线程的结束而死亡。

**2.2JAVA虚拟机栈**：Java 内存可以粗糙的区分为堆内存（Heap）和栈内存（Stack）其中栈就是现在说的虚拟机栈，或者说是虚拟机栈中局部变量表部分。 （实际上，Java虚拟机栈是由一个个栈帧组成，而每个栈帧中都拥有局部变量表、操作数栈、动态链接、方法出口信息）

局部变量表主要存放了编译器可知的各种数据类型（boolean、byte、char、short、int、float、long、double）、对象引用（reference类型，它不同于对象本身，可能是一个指向对象起始地址的引用指针，也可能是指向一个代表对象的句柄或其他与此对象相关的位置）。

Java 虚拟机栈会出现两种异常：StackOverFlowError 和 OutOfMemoryError。

StackOverFlowError： 若Java虚拟机栈的内存大小不允许动态扩展，那么当线程请求栈的深度超过当前Java虚拟机栈的最大深度的时候，就StackOverFlowError异常。OutOfMemoryError： 若 Java 虚拟机栈的内存大小允许动态扩展，且当线程请求栈时内存用完了，无法再动态扩展了，此时抛出OutOfMemoryError异常。

Java 虚拟机栈也是线程私有的，每个线程都有各自的Java虚拟机栈，而且随着线程的创建而创建，随着线程的死亡而死亡。

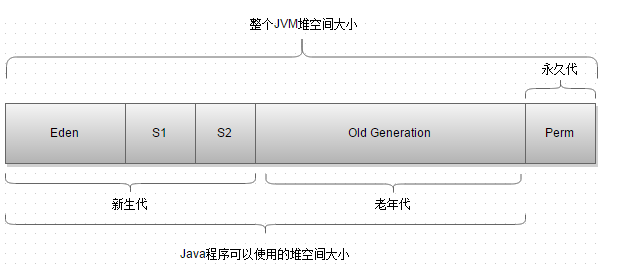
2.3本地方法栈

和虚拟机栈所发挥的作用非常相似，区别是： 虚拟机栈为虚拟机执行 Java 方法 （也就是字节码）服务，而本地方法栈则为虚拟机使用到的 Native 方法服务。 在 HotSpot 虚拟机中和 Java 虚拟机栈合二为一。

本地方法被执行的时候，在本地方法栈也会创建一个栈帧，用于存放该本地方法的局部变量表、操作数栈、动态链接、出口信息。方法执行完毕后相应的栈帧也会出栈并释放内存空间，也会出现 StackOverFlowError 和 OutOfMemoryError 两种异常。

2.4堆

Java虚拟机所管理的内存中最大的一块，java堆是所有线程共享的一块内存区域，在虚拟机启动的时候创建，此内存区域的唯一目的就是存放对象实例，几乎所有的对象实例以及数组都在这里分配内存。Java堆是垃圾收集管理的主要区域，以你为也被称为GC堆(Garbage Collected Heap)从垃圾回收的角度，由于现在的收集器基本都采用分代垃圾收集算法，所以java堆还可以细分为：新生代和老年代，再细致一点有Eden空间、From survivor、To Survicor空间等，进一步划分的目的是为了更好的回收内存，或者更快的分配内存。



Jdk1.8以后移除了整个永久代(PernGen)，取而代之的是一个叫元空间(Metaspace)的区域(永久代使用的是jvm堆内存空间，而元空间使用的是物理内存，直接受到本机的物理内存的限制)

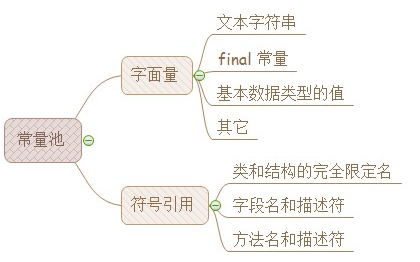
2.5方法区

方法区与java堆一样，是各个线程共享的内存区域，他用于存储已被虚拟机加载的类信息、常量、静态变量、即时编译器编译后的代码等数据。虽然java虚拟机规范把方法区描述为堆的一个逻辑部分，但他却又一个别名叫做 Non-Heap（非堆），目的应该是与 Java 堆区分开来。HotSpot 虚拟机中方法区也常被称为 “永久代”，本质上两者并不等价。仅仅是因为 HotSpot 虚拟机设计团队用永久代来实现方法区而已，这样 HotSpot 虚拟机的垃圾收集器就可以像管理 Java 堆一样管理这部分内存了。但是这并不是一个好主意，因为这样更容易遇到内存溢出问题。

2.6运行时常量

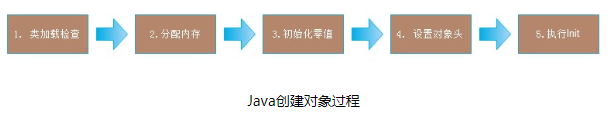
运行时常量池是方法区的一部分。Class 文件中除了有类的版本、字段、方法、接口等描述信息外，还有常量池信息（用于存放编译期生成的各种字面量和符号引用）既然运行时常量池时方法区的一部分，自然受到方法区内存的限制，当常量池无法再申请到内存时会抛出 OutOfMemoryError 异常。

JDK1.7及之后版本的 JVM 已经将运行时常量池从方法区中移了出来，在 Java 堆（Heap）中开辟了一块区域存放运行时常量池。



### HotSpot虚拟机对象

3.1对象的创建



1).**类加载检查**。虚拟机遇到一条 new 指令时，首先将去检查这个指令的参数是否能在常量池中定位到这个类的符号引用，并且检查这个符号引用代表的类是否已被加载过、解析和初始化过。如果没有，那必须先执行相应的类加载过程。

2).**分配内存**。在类加载检查通过后，接下来虚拟机将为新生对象分配内存。对象所需的内存大小在类加载完成后便可确定，为对象分配空间的任务等同于把一块确定大小的内存从 Java 堆中划分出来。分配方式有 “指针碰撞” 和 “空闲列表” 两种。选择以上两种方式中的哪一种，取决于 Java 堆内存是否规整。而 Java 堆内存是否规整，取决于 GC 收集器的算法是”标记-清除”，还是”标记-整理”（也称作”标记-压缩”）。补充内容，虚拟机再创建对象时要保证线程是安全的，有两种方式：CAS+失败重试和TLAB。

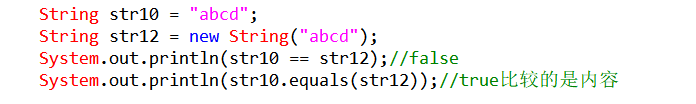
3).**初始化零值**：内存分配完成后，虚拟机需要将分配到的内存空间都初始化为零值（不包括对象头），这一步操作保证了对象的实例字段在 Java 代码中可以不赋初始值就直接使用，程序能访问到这些字段的数据类型所对应的零值。

4).**设置对象头**：初始化零值完成之后，虚拟机要对对象进行必要的设置，例如这个对象是那个类的实例、如何才能找到类的元数据信息、对象的哈希吗、对象的 GC 分代年龄等信息。 这些信息存放在对象头中。 另外，根据虚拟机当前运行状态的不同，如是否启用偏向锁等，对象头会有不同的设置方式。

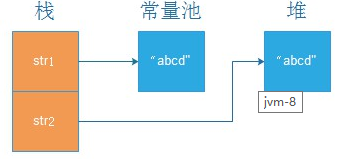
5).**执行init()**:在上面工作都完成之后，从虚拟机的视角来看，一个新的对象已经产生了，但从 Java 程序的视角来看，对象创建才刚开始，<init> 方法还没有执行，所有的字段都还为零。所以一般来说，执行 new 指令之后会接着执行 <init> 方法，把对象按照程序员的意愿进行初始化，这样一个真正可用的对象才算完全产生出来。

### 4.重点补充内容

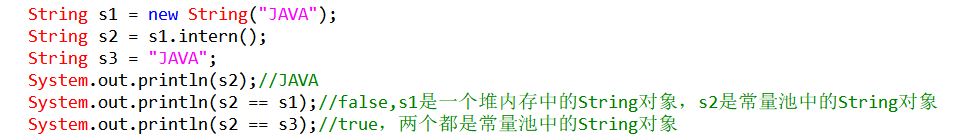
4.1String类和常量池

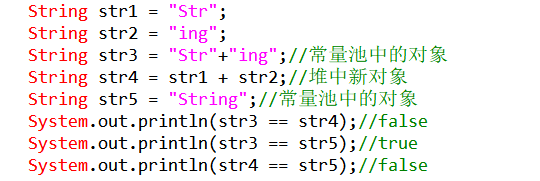


第一种创建方式是直接从常量池中拿对象，第二种方式是直接再堆内存空间中创建一个新的对象。



直接使用双引号声明出来的 String 对象会直接存储在常量池中。如果不是用双引号声明的 String 对象，可以使用 String 提供的 intern 方法。String.intern() 是一个 Native 方法，它的作用是：如果运行时常量池中已经包含一个等于此 String 对象内容的字符串，则返回常量池中该字符串的引用；如果没有，则在常量池中创建与此 String 内容相同的字符串，并返回常量池中创建的字符串的引用。Intern()方法会在常量池中建立。





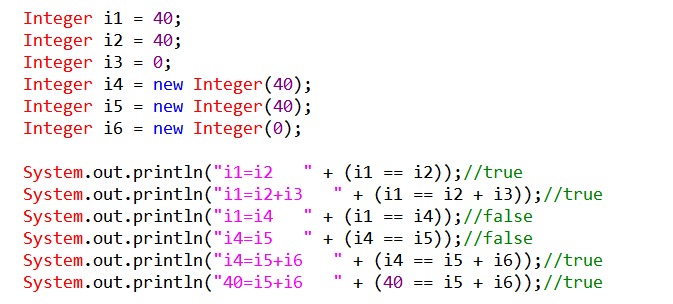


两个，先有字符串 “abc” 放入常量池，然后 new 了一份字符串 “abc” 放入 Java 堆（字符串常量 “abc” 在编译期就已经确定放入常量池，而 Java 堆上的 “abc” 是在运行期初始化阶段才确定），然后 Java 栈的 str1 指向 Java 堆上的 “abc”。

4.2八种基本类型的包装类和常量池

Java 基本类型的包装类的大部分都实现了常量池技术，即 Byte、Short、Integer、Long、Character、Boolean；这5种包装类默认创建了数值 [-128，127] 的相应类型的缓存数据，但是超出此范围仍然会去创建新的对象。

两种浮点数类型的包装类 Float、Double 并没有实现常量池技术。



语句 i4 == i5 + i6，因为 + 这个操作符不适用于 Integer 对象，首先 i5 和 i6 进行自动拆箱操作，进行数值相加，即 i4 == 40。然后Integer对象无法与数值进行直接比较，所以i4自动拆箱转为int值40，最终这条语句转为40 == 40进行数值比较。

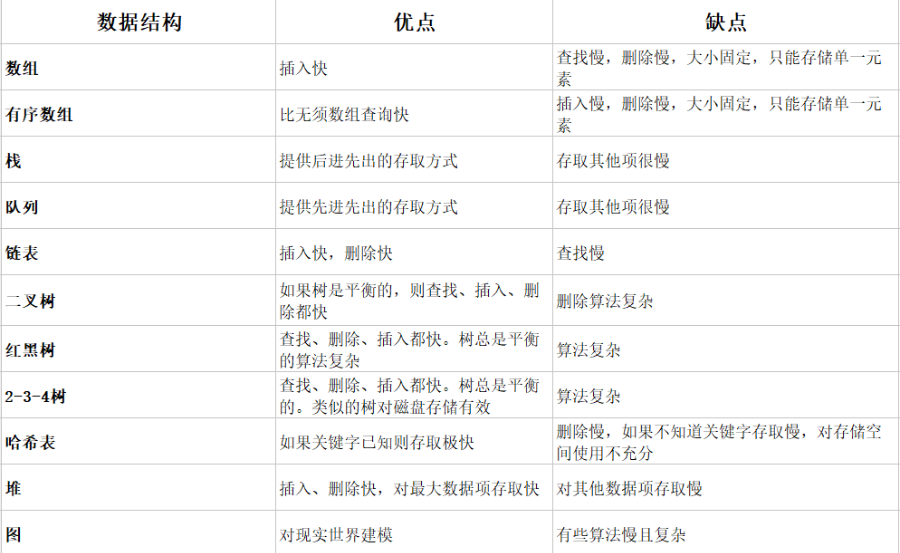
# JAVA数据结构和算法

## 简介

1.1数据结构



几种数据结构的概述



1.2算法

算法的五种特征：有穷性、确定性、可行性、有输入、有输出。

算法的设计原则：

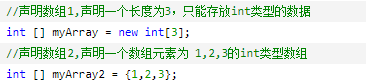
正确性、可读性、健壮性、高效率与低存储量需求。

关于算法的效率，两种算法比较一定要和数据项的个数联系起来，也就是大O，有的算法在数据量特别大的时候性能才能显现出来。

算法是解决问题的步骤，而数据结构的实现离不开算法。

## 数组

2.1数组的两种声明方式



数组的局限性：

数组的局限性分析：

　 ①、插入快，对于无序数组，上面我们实现的数组就是无序的，即元素没有按照从大到小或者某个特定的顺序排列，只是按照插入的顺序排列。无序数组增加一个元素很简单，只需要在数组末尾添加元素即可，但是有序数组却不一定了，它需要在指定的位置插入。

　　②、查找慢，当然如果根据下标来查找是很快的。但是通常我们都是根据元素值来查找，给定一个元素值，对于无序数组，我们需要从数组第一个元素开始遍历，直到找到那个元素。有序数组通过特定的算法查找的速度会比无需数组快，后面我们会讲各种排序算法。

　　③、删除慢，根据元素值删除，我们要先找到该元素所处的位置，然后将元素后面的值整体向前面移动一个位置。也需要比较多的时间。

　　④、数组一旦创建后，大小就固定了，不能动态扩展数组的元素个数。如果初始化你给一个很大的数组大小，那会白白浪费内存空间，如果给小了，后面数据个数增加了又添加不进去了。

　　很显然，数组虽然插入快，但是查找和删除都比较慢，而且扩展性差，所以我们一般不会用数组来存储数据，那有没有什么数据结构插入、查找、删除都很快，而且还能动态扩展存储个数大小呢，答案是有的，但是这是建立在很复杂的算法基础上，后面我们也会详细讲解。

3.栈

## 4.队列

5.前缀、中缀和后缀表达式

6.链表

7.哈希表

8.堆

9.无权无向表

11.二叉树

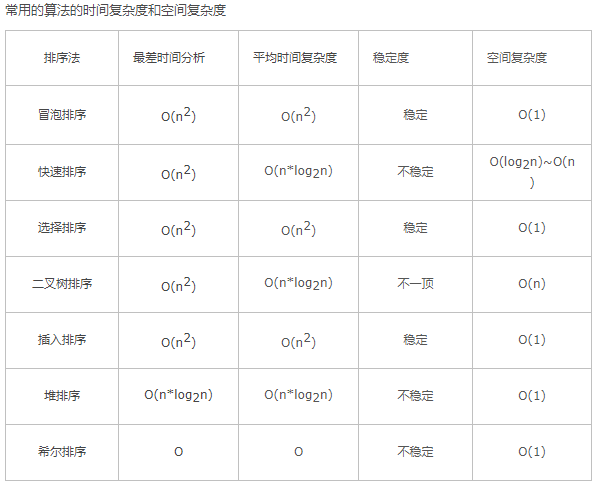
12.红黑树

13. 2-3-4树

14.递归

15.几种常见的排序算法

15.1时间复杂度和空间复杂度



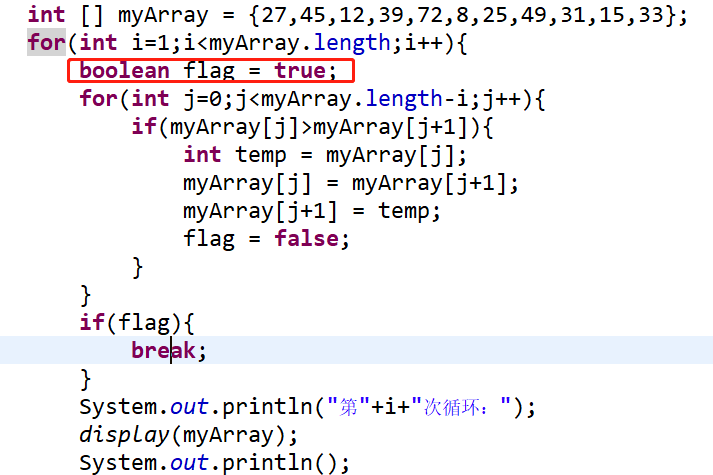
**时间复杂度**：

(1)时间频度:一个算法执行所消耗的时间，理论上是没法估计的，但我们可以理论上比较两个算法哪个花费的时间少，并且一个算法花费的时间与算法中语句的执行次数成正比例，哪个算法中语句执行次数多，它花费时间就多。一个算法中的语句执行次数称为语句频度或时间频度。记为T(n)。

(2)时间复杂度：若有某个辅助函数f(n),使得当n趋近于无穷大时，T（n)/f(n)的极限值为不等于零的常数，则称f(n)是T(n)的同数量级函数。记作T(n)=Ｏ(f(n)),称Ｏ(f(n)) 为算法的渐进时间复杂度，简称时间复杂度。若算法中语句执行次数为一个常数，则时间复杂度为O(1),另外，在**时间频度不相同时，时间复杂度有可能相同**，如T(n)=n2+3n+4与T(n)=4n2+2n+1它们的频度不同，但时间复杂度相同，都为O(n2)。 按数量级递增排列，常见的时间复杂度有：常数阶O(1),对数阶O(log2n),线性阶O(n), 线性对数阶O(nlog2n),平方阶O(n2)，立方阶O(n3),...， k次方阶O(nk),指数阶O(2n)。

**空间复杂度**：是指算法在计算机内执行时所需存储空间的度量。一个算法在计算机存储器上所占用的存储空间，包括存储算法本身所占用的存储空间，算法的输入输出数据所占用的存储空间和算法在运行过程中临时占用的存储空间这三个方面。

15.2冒泡排序



记住一个句话，每次循环的结果是最大的排在最右边，所以每次内循环都少一次；外循环从1开始，内循环从0开始，且每次减一。

另外记得设置标志位：没有改变次序的不计数。

**冒泡排序性能分析**：假设参与比较的数组元素个数为 N，则第一轮排序有 N-1 次比较，第二轮有 N-2 次，如此类推，这种序列的求和公式为：

　　（N-1）+（N-2）+...+1 = N\*（N-1）/2

　当 N 的值很大时，算法比较次数约为 N2/2次比较，忽略减1。

　　假设数据是随机的，那么每次比较可能要交换位置，可能不会交换，假设概率为50%，那么交换次数为 N2/4。不过如果是最坏的情况，初始数据是逆序的，那么每次比较都要交换位置。

　　交换和比较次数都和N2 成正比。由于常数不算大 O 表示法中，忽略 2 和 4，那么冒泡排序运行都需要 O(N2) 时间级别。

　　其实无论何时，只要看见一个循环嵌套在另一个循环中，我们都可以怀疑这个算法的运行时间为 O(N2)级，外层循环执行 N 次，内层循环对每一次外层循环都执行N次（或者几分之N次）。这就意味着大约需要执行N2次某个基本操作。

15.高级排序