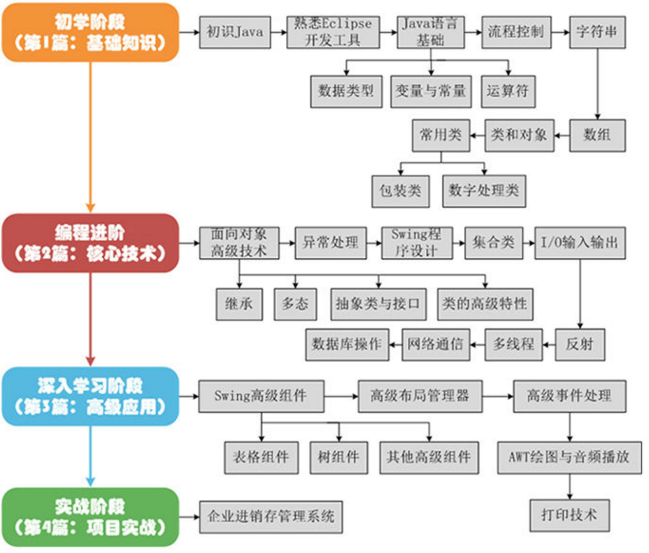
# Java EE笔记



## Java 精通

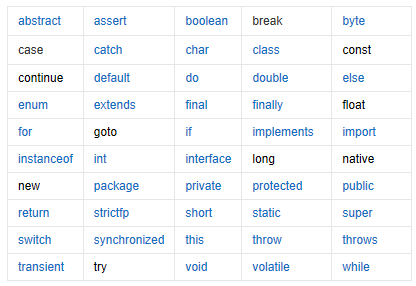
### 第一章java基础

#### 1.1 java基础语法

##### 1.1.1 java 标识符

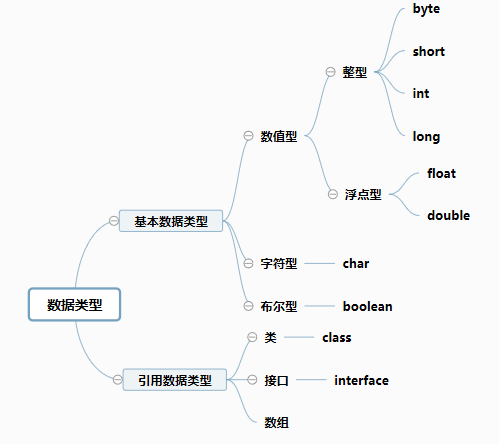
|  |  |  |
| --- | --- | --- |
| Java命名规范 | 示例 | 说明 |
| 包的命名 | scnu.cn.edu.controllor | 全部小写，由域名组成 |
| 类的命名 | GraphicsTest | 单词首字母大写 |
| 方法的命名 | drawImage | 首字母小写，其后单词首字母大写 |
| 常量的命名 | MAX\_VALUE | 全部大写，下划线分词 |
| 参数命名 | fValue | 同方法的命名 |
| 变量命名 | s\_sentence或者strSentence（常用） | 匈牙利命名，类型\_变量名|类型变量名 |

##### 1.1.2 java 关键字



#### 1.2 java数据类型

##### 1.2.1 java 数据类型分类



##### 1.2.2 java数据类型空间占用

|  |  |
| --- | --- |
| 类型 | 占内存大小 |
| byte | 1B |
| short | 2B |
| int | 4B |
| long | 8B |
| float | 4B |
| double | 8B |

#### 1.3 java 控制流程

##### 1.3.1 java运算符

算术运算符，关系运算符，逻辑运算符

|  |  |  |
| --- | --- | --- |
| 算术运算符 | 关系运算符 | 逻辑运算符 |
| + 加法 | == 等于 | && 与（短路） |
| * 减法 | != 不等于 | || 或（短路） |
| * 乘法 | > 大于 | ! 非 |
| / 除法 | < 小于 | & 与（非短路） |
| % 取模 | >= 大于等于 | | 或（非短路） |
| -- 自减 | <= 小于等于 |  |
| ++ 自加 | expression ? true:false(三元) |  |

所谓的“短路”，就是判断一半如果就能确定真假，后面的表达式就不会判断了。比如“&&”操作，如果前面的是假，整个表达式就是假，“&&”后面的表达式就不会判断。

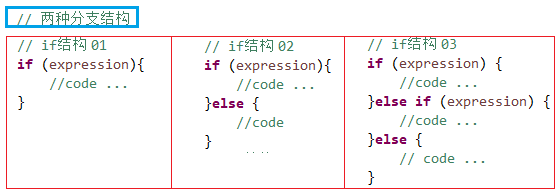
移位运算符：

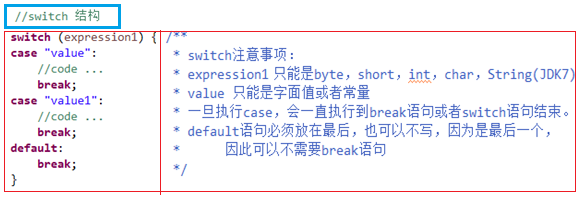
|  |  |
| --- | --- |
| & 按位与 | 1<<3 1\*23 左移 |
| | 按位或 | 8>>3 8/23 右移 |
| ^ 按位异或 | >>> 无符号右移(补0) |
| ~ 按位补(反转) |  |

##### 1.3.2 java 控制语句(6种)

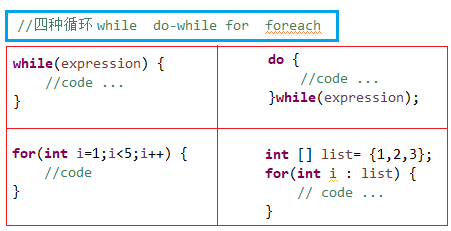
两种分支

两种分支：if语句或者switch语句。而if语句中又有三种结构。如下代码：





四种循环 while do-while for foreach



**注意**： break关键字：结束循环

continue关键字：跳出本次循环

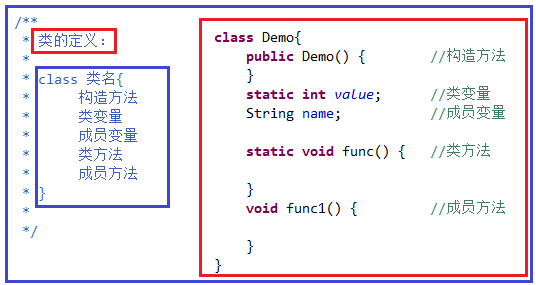
### 第二章 java进阶

#### 2.1 面向对象思想

面向对象，即是通过**消息**传递实现各个对象之间的合作，从而实现程序的功能。因此，一个对象能接收什么样的**消息**是进行面向对象设计的重心，也即是需求分析。

http://www.cnblogs.com/carsonzhu/p/5472159.html

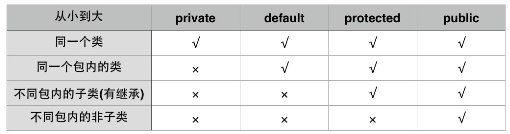
##### 2.1.1 创建类



#### 2.2 封装

##### 2.2.1 权限修饰符

***总之：private同一类内访问，default 同一包内访问，public 同一工程内访问。***

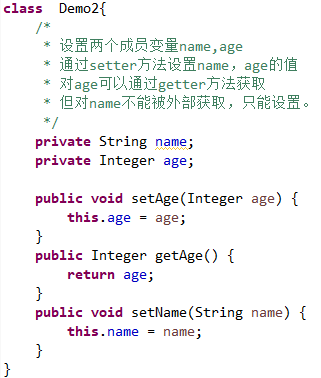


##### 2.2.2 封装性的使用

1）封装的目的：给予属性和方法权限。

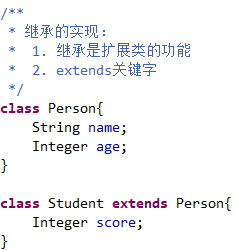
2）封装的实现（示例）：

为属性和方法通过private 进行封装，通过setter和getter方法访问。

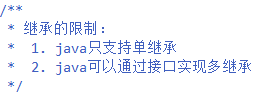


#### 2.3 继承

##### 2.3.1 继承的实现



##### 2.3.2 继承的限制

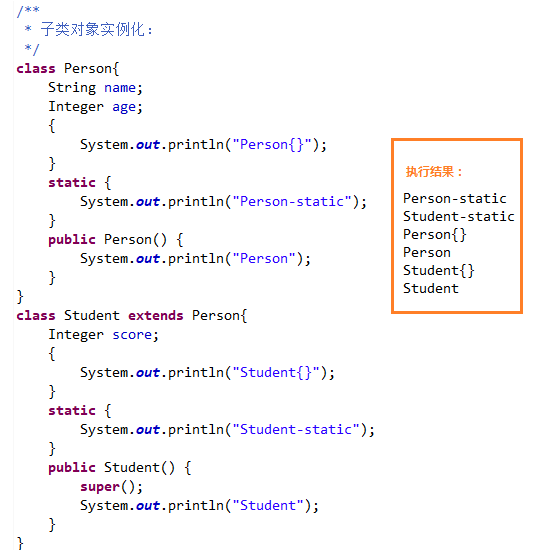


##### 2.3.3 子类对象的实例化

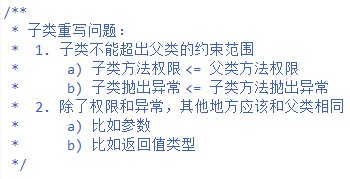
执行顺序：

1. 静态代码块 > 构造代码块 > 构造方法
2. 父类 > 子类
3. 即：

***父类静态代码块 –> 子类静态代码块 –> 父类构造代码块 –> 父类构造方法 –> 子类构造代码块 –> 子类构造方法***

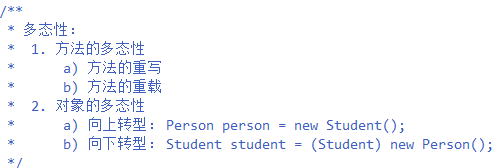


##### 2.3.4 重写



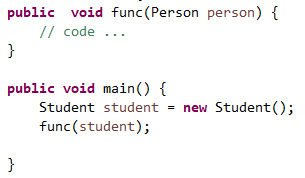
#### 2.4 多态

##### 2.4.1 多态的概念



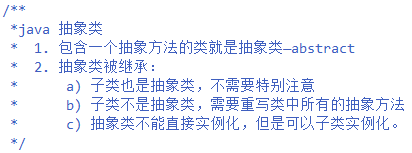
##### 2.4.2 多态的使用

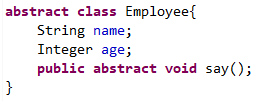
方法传递参数时，可以将参数设置为父类，传入时根据需求传入指定的子类。

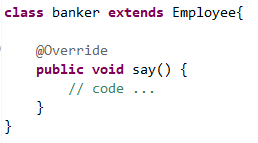


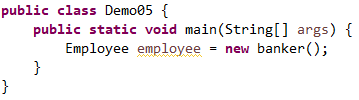
#### 2.5 接口

##### 2.5.1 java抽象类



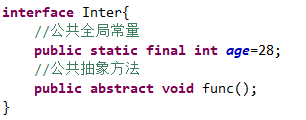




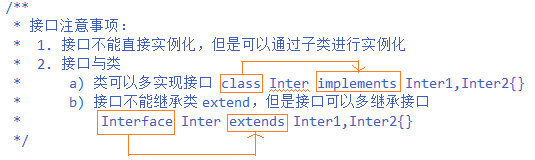


##### 2.5.2 java接口

接口的定义：



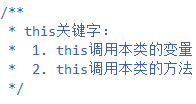
接口的特点：



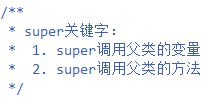
#### 2.6 类中各种关键字

|  |  |  |
| --- | --- | --- |
| 关键字 | 示例 | 说明 |
| this | this . name = name; | 本类 |
| super | super(); | 父类 |
| final | final double PI;  final Person person; | final 在指向基本类型时表示：**该基本类型是常量，比如π**  final 在指向引用类型时表示：  **该引用类型指向不变，但是内容可变。** |
| static | static int age; | **类变量，即只在类第一次实例化时创建赋值，之后的实例化不会重新创建。** |
| instanceof | 对象 instanceof 类 | 判断对象是不是该类的对象 |

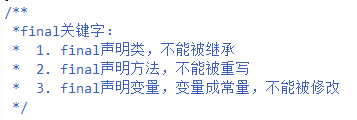
##### 2.6.1 this关键字



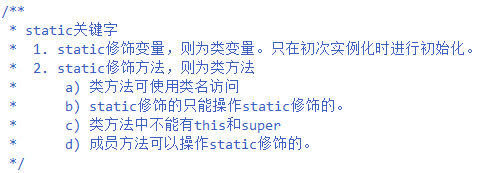
##### 2.6.2 super 关键字



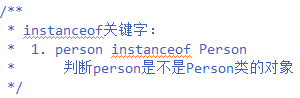
##### 2.6.3 final 关键字



##### 2.6.4 static 关键字



##### 2.6.4 instanceof 关键字



#### 2.7 其他特性与包

##### 2.7.1 匿名对象

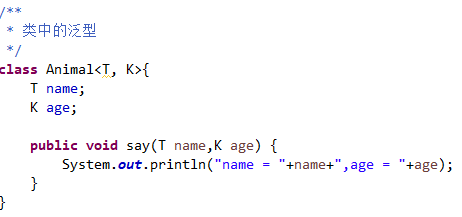


##### 2.7.2 泛型

###### 2.7.2.1 泛型的定义



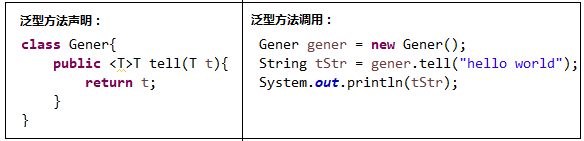
###### 2.7.2.2类/接口中的泛型



**泛型的声明与使用：**



###### 2.7.2.3 泛型方法



###### 2.7.2.4 通配符



##### 2.7.3 包



#### 2.8 常用类

##### 2.8.1 数组

###### 2.8.1.1 数组的声明



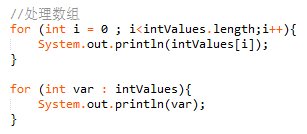
###### 2.8.1.2 数组的创建



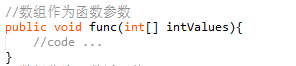
###### 2.8.1.3 数组的初始化



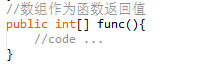
###### 2.8.1.4 数组的处理



###### 2.8.1.5 数组作为函数参数



###### 2.8.1.6 数组作为函数返回值



###### 2.8.1.7 java.util.Arrays类

Array类，其提供的方法都是类方法static

**1. 给数组赋值，fill方法**

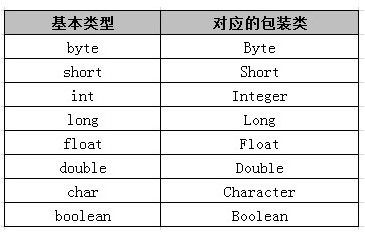
**2. 对数组排序 sort方法**

**3. 比较数组 通过equals比较是否相等**

**4. 查找数组元素：通过binarySearch执行二分法查找**

##### 2.8.2 包装类

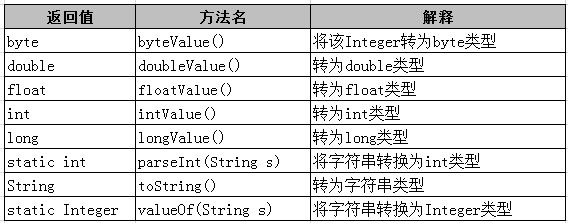
###### 2.8.2.1 基本类型与包装类的对应关系：



其中包装类主要提供了两大类型的方法：

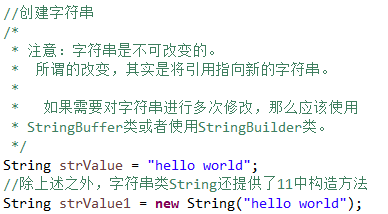
1. **将本类型和其他基本类型进行转换的方法**
2. **将字符串和本类型以及其包装类互相转换的方法**

###### 2.8.2.2 以Integer类型为例，其常用方法如下：



##### 2.8.3 String类

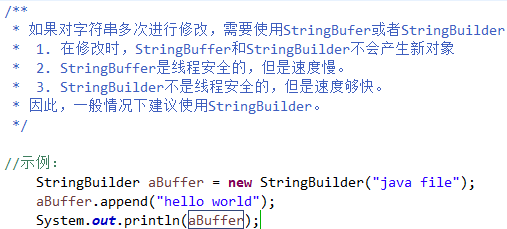
###### 2.8.3.1 String类







###### 2.8.3.2 StringBuffer和StringBuilder类



##### 2.8.4 集合类

##### 2.8.5 日期类

###### 2.8.5.1 日期处理的关键类

**Instant**——它代表的是时间戳

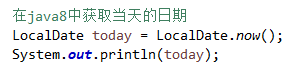
**LocalDate**——不包含具体时间的日期，比如2014-01-14。它可以用来存储生日，周年纪念日，入职日期等。

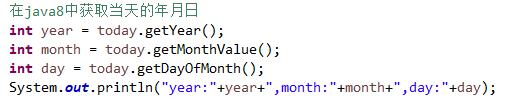
**LocalTime**——它代表的是不含日期的时间

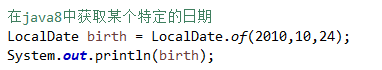
**LocalDateTime**——它包含了日期及时间，不过还是没有偏移信息或者说时区。

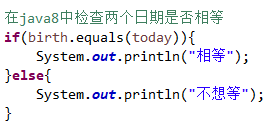
**ZonedDateTime**——这是一个包含时区的完整的日期时间，偏移量是以UTC/格林威治时间为基准的。

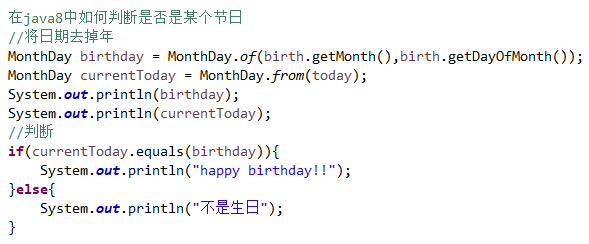
###### 2.8.5.2 日期类的测试：

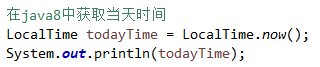


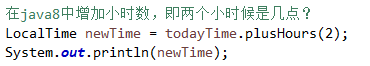


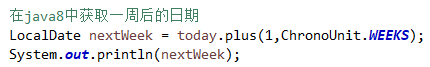


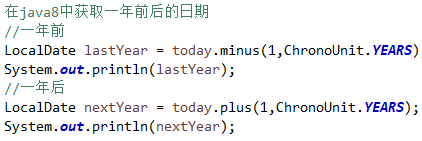












### 第三章 java高级

#### 3.1 多线程

#### 3.2 网络编程

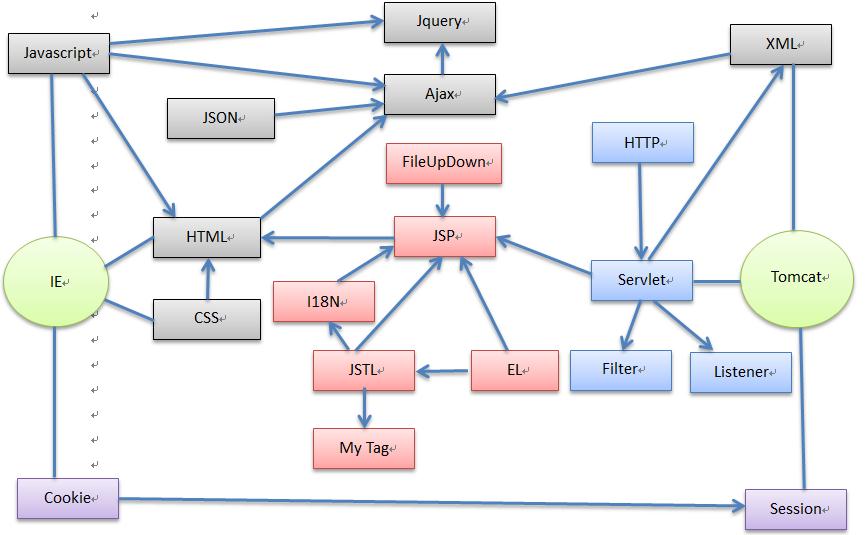
#### 3.3 反射

#### 3.4 设计模式

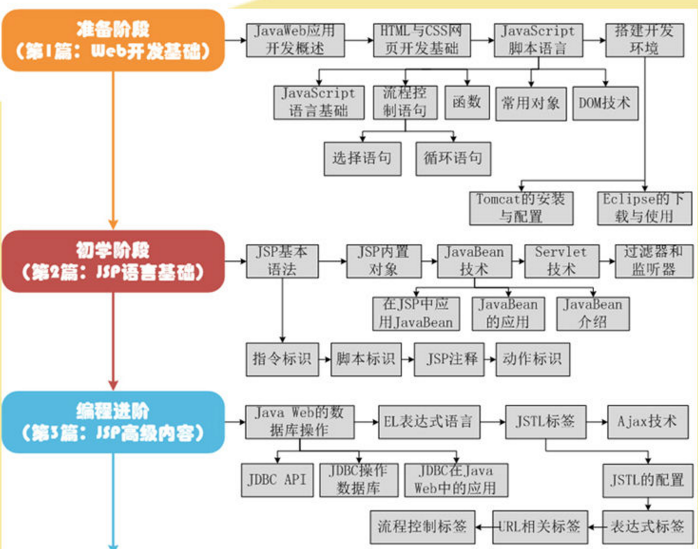
#### 3.5 GUI编程

## JAVA Web精通

JavaWEB 知识结构图



JavaWeb 学习流程图



### HTTP

#### HTTP的主要特点

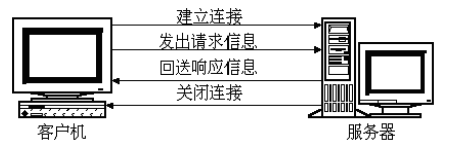
简单，快速，灵活，无连接，无状态

所谓的无连接，就是在交互后关闭连接，下一次交互会发送新的请求

所谓的无状态，就是下一次交互的页面与上一次的无关，每次请求都是独立的，这时如果下一个页面需要上一个页面的信息，就需要重新传送，这样就会导致数据量比较大，但是如果不需要上一个页面的信息，那么会反应比较快。

#### HTTP 的会话方式

##### 1.2.1 浏览器与客户端至少需要四个步骤



##### 1.2.2 HTTP请求方式1.0 与 1.1

在1.0 版本中，如果网页中带有图片之类，会开启一个新的链接下载图片。但是1.1之后都是使用同一个连接

#### HTTP请求

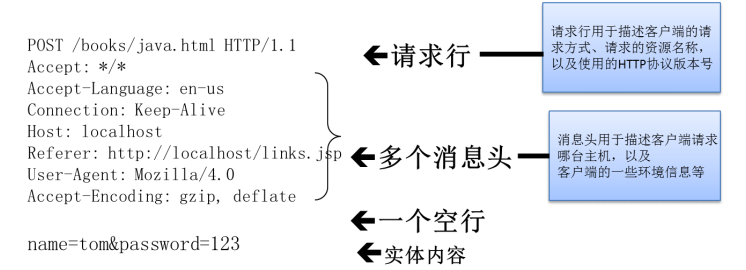
##### 一个完整的HTTP请求

一个完整的HTTP请求包括：

①请求行

②请求头

③请求体



##### 请求方法

①HTTP中定义了7中请求方式：POST，GET，HEAD，OPTIONS，DELETE，TRACE，PUT

②GET请求

GET请求是比较常用的，无论是在浏览器中输入url地址还是超链接等都是以GET的方式进行的

GET对于参数的携带都是在URL地址之后的<http://xxx.htm?name> = abc & pwd=123

从上可知，GET在携带数据上大小是有限制的，因为数据放在请求头中。

GET请求由于将数据放在请求头中，因此本身没有请求体

③POST请求

POST一般用在表单提交或者在Ajax中指定使用POST，由于POST方式是将数据放在请求体中，因此可以携带大量数据，并且大小是没有限制的。

POST请求，是将form表单提交的数据放在请求体中，发送给服务器的，因此会有请求体。

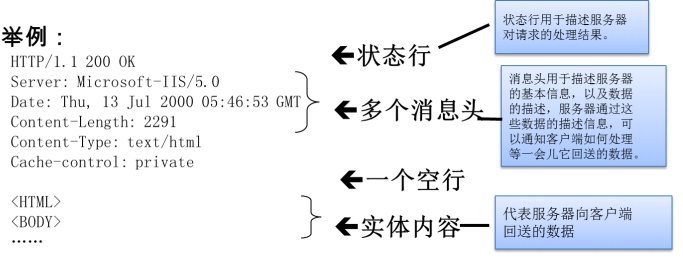
##### HTTP的响应

一个HTTP的响应也是由三部分组成的：

①响应状态行，包括协议版本，状态码等

②响应消息头，响应头

③实体内容：响应体



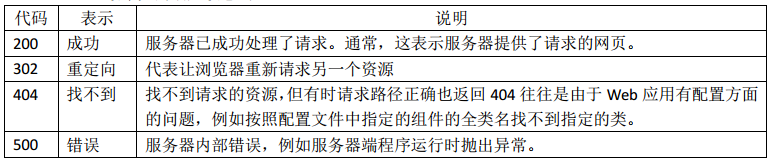
常见的响应状态码：

响应状态码以2开头通常表示成功

响应状态码以3开头通常表示转移，重定向

响应状态码以4开头通常表示无法访问，找不到或者没权限，如404

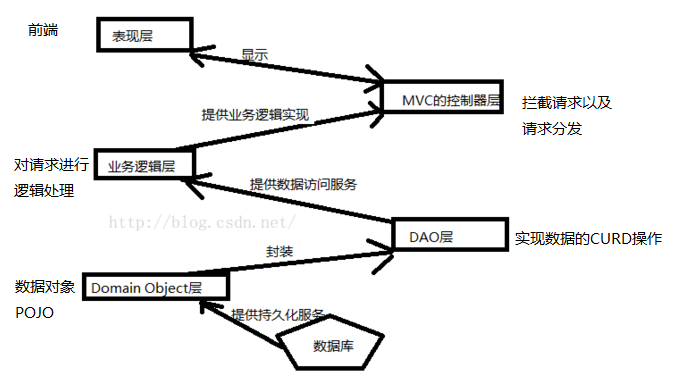
响应状态码以5开头通常表示服务器错误



### 第二章 Tomcat

### 第三章 Servlet

#### 3.1 JavaEE的分层模型



其中Servlet主要实现的是业务逻辑层

#### 3.2 Servlet的概念

①servlet是javaEE 的web组件，组件需要在容器中运行，配置web.xml

②Servlet是一个特殊的java类，是继承了HttpServlet的类

③重写HttpServlet中的方法

#### 3.3 Servlet 的示例：

①创建myServlet类并且继承HttpServlet类，并且重写doPost以及doGet方法。

②部署Servlet，即配置web.xml文件

MyServlet.java

public class MyServlet extends HttpServlet {

private static final long *serialVersionUID* = 1L;

@Override

protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {

//设置编码格式

resp.setContentType("text/html;charset=utf-8");

//获取输出流

PrintWriter out = resp.getWriter();

out.println("HTML");

}

@Override

protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {

this.doPost(req, resp);

}

}

Web.xml

<!-- 设置Servlet类 -->

<servlet>

<servlet-name>myServlet</servlet-name>

<servlet-class>sc.edu.scnu.servlet.MyServlet</servlet-class>

</servlet>

<!-- 部署Servlet，即设置访问URL -->

<servlet-mapping>

<servlet-name>myServlet</servlet-name>

<url-pattern>/first</url-pattern>

</servlet-mapping>

#### 3.4 Servlet 获取表单数据

①String getParameter(String name)，通过name 获取数据

②String[] getParameter(String name)，通过name获取多个数据，例如checkbox等。一个name对应多个数据时。

示例：

Index.jsp

<form action="/myServlet/first" method="post">

username :<input type="text" name="username"><br>

password :<input type="password" name = "pwd"><br>

<input type="checkbox" name = "favor" value="favor1">favor1<br>

<input type="checkbox" name = "favor" value="favor2">favor2<br>

<input type="checkbox" name = "favor" value="favor3">favor3<br>

<input type="submit" value="提交">

</form>

MyServlet.java —— doPost

**protected** **void** doPost(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {

// **TODO** Auto-generated method stub

String username = req.getParameter("username");

String pwd = req.getParameter("pwd");

System.**out**.println(username);

System.**out**.println(pwd);

System.**err**.println("---------------");

String[] strings = req.getParameterValues("favor");

**for**(String str:strings) {

System.**out**.println(str);

}

}

3.5 Servlet 控制跳转页面

① 响应重定向方式，两次请求——客户端行为

例如，我需要设计一个软件，就找到A公司，A公司说不会，然后我去找B公司，完成后，我可以对外说是B公司帮我完成的。

String username = req.getParameter("username");

String pwd = req.getParameter("pwd");

**if**(username.equalsIgnoreCase("admin")&&pwd.equalsIgnoreCase("admin")){

resp.sendRedirect("/myServlet/view/success.jsp");

}**else** {

resp.sendRedirect("/myServlet/view/fail.html");

}

②请求转发，一次请求——服务器端行为。

例如，我需要设计一个软件，就找到A公司，A公司做不了，就外包给B公司，但是对我来说，我会对外称是A公司设计的，因为B公司对我是透明的。也就是说，请求转发时，URL地址并不会发生改变。

**if**(username.equalsIgnoreCase("admin")&&pwd.equalsIgnoreCase("admin")){

//请求转发

//req.getRequestDispatcher("/view/fail.html").forward(req, resp);

//请求重定向

resp.sendRedirect("/myServlet/view/success.jsp");

}**else** {

//请求转发

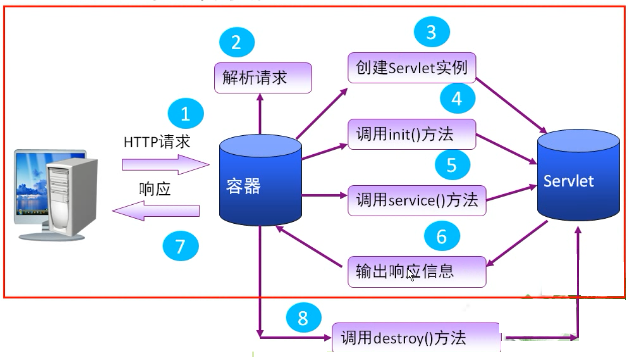
req.getRequestDispatcher("/view/fail.html").forward(req, resp);

//请求重定向

//resp.sendRedirect("/myServlet/view/fail.html");

}

#### 3.5 Servlet 的生命周期



### 第四章 JSP

#### 4.1 JSP中的概念

从本质上说，JSP也是一个Servlet，只是JSP是实现view部分，Servlet是实现逻辑控制部分。

JSP，java Servlet Page。 == HTML+java代码

在JSP中主要有两个元素：

① 脚本元素： <% java 代码%>

②表达式：<%= 需要输出的内容 %>

示例：

<!-- 在jsp中显示10个checkbox，使用jsp中的脚本以及表达式 -->

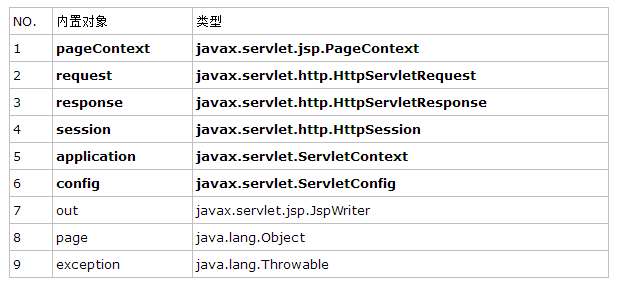
<%**for**(**int** i = 0;i<10;i++){ %>

<input type=*"checkbox"*> <%=i %>

<%} %>

#### 4.2 JSP中的内置对象

##### 4.2.1 认识JSP中的九大内置对象



##### 4.2.2 JSP中的常用内置对象详解

①request，从上表中可以看出，request对象是来自javax.http.HttpServletRequest类的。

②response

③session

④application

⑤pageContext

#### 4.3 JSP与Servlet之间传递数据

#### 4.4 JSP的自定义标签库

#### 4.5 JSP 的EL表达式

#### 4.6 JSP 的JSTL

## 第三节 Spring 精通

### 第一章 Spring框架

### 第二章 spring 核心容器IOC

#### 2.1 bean的实例化

#### 2.2 bean的管理

#### 2.3 bean属性的设置

#### 2.4 使用注解完成IOC配置

### 第三章 spring 的AOP机制

#### 3.1 spring AOP的原理

#### 3.2 spring AOP的几种通知

#### 3.3 AOP 的注解配置

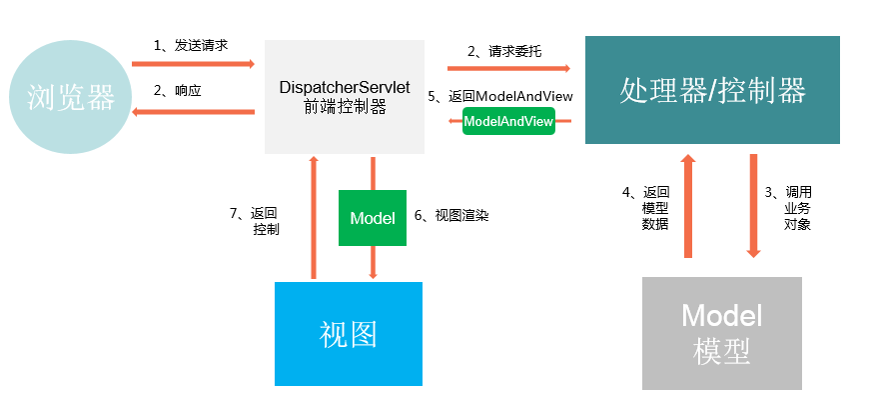
### 第四章 spring的数据库操作

#### 4.1 spring集成jdbc—jdbcTemplet

#### 4.2 spring的事务控制

## 第四节 Spring MVC精通

### spring mvc原理



### 第二章 springMVC的各个组件

#### 2.1 handlerMapping

#### 2.2 控制器

#### 2.3 基于注解的控制器

#### 2.4 ModelAndViewResolver

### 第三章 springMVC 实现功能

#### 3.1 异常处理

#### 3.2 拦截器

#### 3.3 数据传递

#### 3.4 国际化

#### 3.5 类型转换

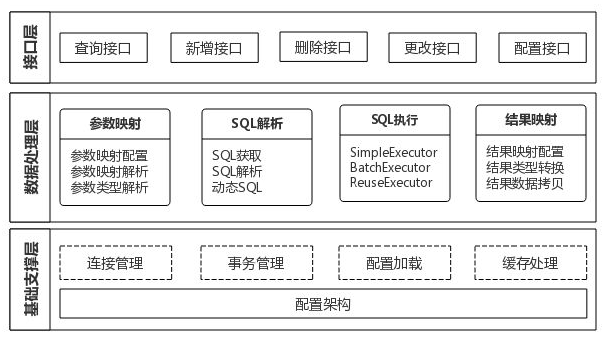
#### 3.6 验证

## 第五节 MyBatis 精通

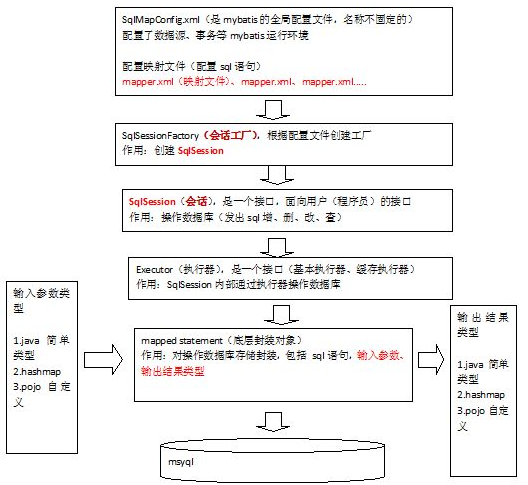
### MyBatis原理

<http://www.mybatis.org/mybatis-3/getting-started.html>

#### 1.1 MyBatis 架构图



#### 1.2 MyBatis运行流程



#### 1.3 MyBatis的增删改查

<http://www.cnblogs.com/woshimrf/p/5651966.html> 示例

##### 1.3.1 需求

* + - 按id查询用户
    - 按name查询用户
    - 增加操作
    - 更新操作
    - 删除操作

##### 1.3.2 环境

* + - Java环境，jdk8 ， IDEA，Maven
    - Mybatis环境，mybatis.jar以及其依赖包
    - Mysql环境，mysql-connect.jar以及其依赖包

##### 1.3.3 Log4j.properties

#开发环境设为DEBUG，生产环境INFO或这ERROR

log4j.rootLogger=DEBUG,stdout

log4j.appender.A1=org.apache.log4j.ConsoleAppender

log4j.appender.A1.layout=org.apache.log4j.PatternLayout

log4j.appender.A1.layout.ConversionPattern=%5p [%t] - %m%n

##### 1.3.4 maven配置

Maven将xml文件编译到classes文件夹下

<build>  
 <resources>  
 <resource>  
 <directory>src/main/java</directory>  
 <includes>  
 <include>\*\*/\*.xml</include>  
 </includes>  
 </resource>  
 </resources>  
</build>

Maven的依赖包

<dependencies>

<!-- https://mvnrepository.com/artifact/junit/junit -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.12</version>

<scope>test</scope>

</dependency>

<-- mysql5与mysql6的区别，com.mysql.jdbc.Driver

Com.mysql.cj.jdbc.Driver

-->

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>5.1.44</version>

</dependency>

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis</artifactId>

<version>3.4.5</version>

</dependency>

<!-- https://mvnrepository.com/artifact/asm/asm -->

<dependency>

<groupId>asm</groupId>

<artifactId>asm</artifactId>

<version>3.3.1</version>

</dependency>

<!-- https://mvnrepository.com/artifact/cglib/cglib -->

<dependency>

<groupId>cglib</groupId>

<artifactId>cglib</artifactId>

<version>3.2.5</version>

</dependency>

<dependency>

<groupId>commons-logging</groupId>

<artifactId>commons-logging</artifactId>

<version>1.2</version>

</dependency>

<dependency>

<groupId>org.javassist</groupId>

<artifactId>javassist</artifactId>

<version>3.21.0-GA</version>

</dependency>

<dependency>

<groupId>org.apache.logging.log4j</groupId>

<artifactId>log4j-core</artifactId>

<version>2.8.2</version>

</dependency>

<dependency>

<groupId>log4j</groupId>

<artifactId>log4j</artifactId>

<version>1.2.17</version>

</dependency>

<dependency>

<groupId>org.apache.logging.log4j</groupId>

<artifactId>log4j-api</artifactId>

<version>2.8.2</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.slf4j/slf4j-api -->

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>1.7.25</version>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-log4j12</artifactId>

<version>1.7.25</version>

<scope>test</scope>

</dependency>

</dependencies>

##### 1.3.5 全局配置文件sqlMaperConfig.xml

<?xml version=*"1.0"* encoding=*"UTF-8"* ?>

<!DOCTYPE configuration

PUBLIC "-//mybatis.org//DTD Config 3.0//EN"

"http://mybatis.org/dtd/mybatis-3-config.dtd">

<configuration>

<!-- 加载java的配置文件或者声明属性信息 -->

<properties resource=*"db.properties"*>

<property name=*"db.username"* value=*"123"* />

</properties>

<!-- <settings></settings> -->

<!-- 自定义别名 -->

<typeAliases>

<!-- 单个别名定义 -->

<!-- <typeAlias type="com.itheima.mybatis.po.User" alias="user"/> -->

<!-- 批量别名定义（推荐） -->

<!-- package：指定包名称来为该包下的po类声明别名，默认的别名就是类名（首字母大小写都可） -->

<package name=*"com.itheima.mybatis.po"* />

</typeAliases>

<!-- 配置mybatis的环境信息，与spring整合，该信息由spring来管理 -->

<environments default=*"development"*>

<environment id=*"development"*>

<!-- 配置JDBC事务控制，由mybatis进行管理 -->

<transactionManager type=*"JDBC"*></transactionManager>

<!-- 配置数据源，采用mybatis连接池 -->

<dataSource type=*"POOLED"*>

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

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

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

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

</dataSource>

</environment>

</environments>

<!-- 加载映射文件 -->

<mappers>

<mapper resource=*"User.xml"* />

<!-- <mapper resource="mapper/UserMapper.xml" /> -->

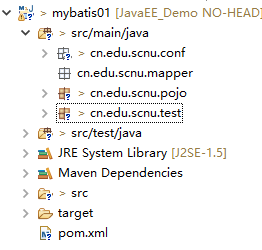
<!-- 批量加载映射文件 -->

<package name=*"com.itheima.mybatis.mapper"* />

</mappers>

</configuration>

##### 1.3.6 工程目录结构



##### 1.3.7 映射文件 UserMapper.xml

<?xml version=*"1.0"* encoding=*"UTF-8"* ?>

<!DOCTYPE mapper

PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"

"http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace=*"test"*>

<!--

其中id是statement的id。

paramterTyper是参数#{id}的类型

resultType:是返回值的类型

-->

<select id="findById" parameterType="int" resultType="cn.edu.scnu.pojo.City">  
 SELECT \* FROM areas WHERE id = #{id}  
</select>

</mapper>

##### 1.3.8 测试文件 \_ 按ID查询用户信息

@Test  
public void findById() throws IOException {  
 //创建会话工厂  
  
 //cn/edu/scnu/conf/mybatis-config.xml  
 String resource = "cn/edu/scnu/conf/mybatis-config.xml";  
 InputStream inputStream = Resources.*getResourceAsStream*(resource);  
  
 SqlSessionFactory sessionFactory = new SqlSessionFactoryBuilder().build(inputStream);  
  
 //创建会话  
 SqlSession session = sessionFactory.openSession();  
  
 City city = session.selectOne("test.findById",2);  
  
 System.*out*.println(city.toString());  
  
 session.close();  
}

##### 1.3.9 按name查询用户信息

Mapper.xml

<!--  
 注意：  
 resultType:返回无论单条还是多条，都指的是数据类型  
 ${} : 表示拼接sql，如果是基本数据类型，{}必须是{value}，否则会出错。  
 由于是拼接sql，因此会出现sql注入，比如value= 'or 1=1 or'，拼接之后：  
 '%' or 1=1 or '%'  
-->  
<select id="findByName" parameterType="java.lang.String" resultType="cn.edu.scnu.pojo.City">  
  
 SELECT \* FROM areas WHERE area LIKE '%${value}%'  
  
</select>

findByName.java

@Test  
public void findByName() throws IOException {  
 //创建会话工厂  
  
 //cn/edu/scnu/conf/mybatis-config.xml  
 String resource = "cn/edu/scnu/conf/mybatis-config.xml";  
 InputStream inputStream = Resources.*getResourceAsStream*(resource);  
  
 SqlSessionFactory sessionFactory = new SqlSessionFactoryBuilder().build(inputStream);  
  
 //创建会话  
 SqlSession session = sessionFactory.openSession();  
  
 List<City> cities = session.selectList("test.findByName","区");  
  
 for(City city:cities){  
 System.*out*.println(city.toString());  
 }  
  
 session.close();  
}

##### 1.3.10 插入用户信息

Mapper.xml

<!--  
 parameterType : 插入的数据对象  
 #{}，可以获取对象中的指定属性，比如id属性。  
-->  
<insert id="insertUser" parameterType="cn.edu.scnu.pojo.User">  
 INSERT INTO users(id,name,age,addr) VALUE(#{id},#{name},#{age},#{addr})  
</insert>

insertObject,java  
@Test  
public void insertObject() throws IOException {  
 User user = new User();  
 user.setId(6);  
 user.setName("ff");  
 user.setAge(24);  
 user.setAddr("shandong");  
  
 System.*out*.println(user.toString());  
 System.*out*.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  
 //创建会话工厂  
  
 //cn/edu/scnu/conf/mybatis-config.xml  
 String resource = "cn/edu/scnu/conf/mybatis-config.xml";  
 InputStream inputStream = Resources.*getResourceAsStream*(resource);  
  
 SqlSessionFactory sessionFactory = new SqlSessionFactoryBuilder().build(inputStream);  
  
 //创建会话  
 SqlSession session = sessionFactory.openSession();  
  
  
 int i = session.insert("test.insertUser",user);  
 session.commit();  
 System.*out*.println(i);  
 session.close();  
  
}

##### 1.3.11 自增主键在插入后的获取

Mapper.xml

<!--  
 其中：  
 SELECT LAST\_INSERT\_ID()：得到刚插入的主键，仅仅适用于自增主键  
 keyProperty:表示的是将返回值，赋值给parameterType指定对象的某个属性  
 order：表示执行顺序在insert语句之前还是之后  
 resultType:指定结果类型  
-->  
<insert id="insertAndGetId" parameterType="cn.edu.scnu.pojo.User">  
  
 <selectKey keyProperty="id" order="AFTER" resultType="java.lang.Integer">  
 SELECT LAST\_INSERT\_ID()  
 </selectKey>  
 INSERT INTO users(name,age,addr) VALUE(#{name},#{age},#{addr})  
</insert>

Insert\_getId.java

@Test  
 public void insert\_getId() throws IOException {   
 User user = new User();  
 user.setName("hh");  
 user.setAge(24);  
 user.setAddr("shandong");  
  
 System.*out*.println(user.toString());  
 System.*out*.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  
 //创建会话工厂  
  
 //cn/edu/scnu/conf/mybatis-config.xml  
 String resource = "cn/edu/scnu/conf/mybatis-config.xml";  
 InputStream inputStream = Resources.*getResourceAsStream*(resource);  
  
 SqlSessionFactory sessionFactory = new SqlSessionFactoryBuilder().build(inputStream);  
  
 //创建会话  
 SqlSession session = sessionFactory.openSession();  
  
  
 session.insert("test.insertAndGetId",user);  
 session.commit();  
 session.close();  
  
 System.*out*.println(user.getId());  
 }

##### 1.3.12 非自增主键，获取uuid值

##### 1.3.13 更新

Mapper.xml

<update id="updateUser" parameterType="cn.edu.scnu.pojo.User">  
 update users set name = #{name},age = #{age},addr = #{addr} WHERE id = #{id}  
</update>

UpdateUser.java

@Test  
public void updateUser() throws IOException {  
 User user = new User();  
 user.setId(1);  
 user.setName("hh");  
 user.setAge(24);  
 user.setAddr("shandong");  
  
 System.*out*.println(user.toString());  
 System.*out*.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  
  
 String resource = "cn/edu/scnu/conf/mybatis-config.xml";  
 InputStream inputStream = Resources.*getResourceAsStream*(resource);  
  
 SqlSessionFactory sessionFactory = new SqlSessionFactoryBuilder().build(inputStream);  
  
 //创建会话  
 SqlSession session = sessionFactory.openSession();  
  
 session.update("test.updateUser",user);  
 session.commit();  
  
 session.close();  
  
}

##### 1.3.14 删除

Mapper.xml

<delete id="deleteUser" parameterType="java.lang.Integer">  
 delete FROM users WHERE id = #{id}  
</delete>

DeleteUser.java

@Test  
public void delteUser() throws IOException {  
 String resource = "cn/edu/scnu/conf/mybatis-config.xml";  
 InputStream inputStream = Resources.*getResourceAsStream*(resource);  
  
 SqlSessionFactory sessionFactory = new SqlSessionFactoryBuilder().build(inputStream);  
  
 //创建会话  
 SqlSession session = sessionFactory.openSession();  
  
 session.delete("test.deleteUser",1);  
 session.commit();  
  
 session.close();  
}

### MyBatis开发Dao方法

#### 2.1 原始dao开发方法

##### 2.1.1 定义dao接口

public interface UserDao {  
 /\*  
 public static final 常量  
 public abstract 方法  
  
 可以省略  
 \*/  
  
 City findById(Integer id);  
 List findByName(String name);  
 void deleteUser(Integer id);  
  
}

##### 2.1.2 实现dao接口，并将sqlSessionFactory 注入

public class UserDaoImpl implements UserDao {  
 private SqlSessionFactory sqlSessionFactory;  
  
 public UserDaoImpl(SqlSessionFactory sqlSessionFactory) {  
 this.sqlSessionFactory = sqlSessionFactory;  
  
 }  
  
 public City findById(Integer id) {  
 SqlSession sqlSession = sqlSessionFactory.openSession();  
 return sqlSession.selectOne("test.findById",1);  
  
 }  
  
 public List findByName(String name) {  
 SqlSession sqlSession = sqlSessionFactory.openSession();  
 return sqlSession.selectList("test.findByName","区");  
 }  
  
 public void deleteUser(Integer id) {  
 SqlSession sqlSession = sqlSessionFactory.openSession();  
 sqlSession.delete("test.deleteUser",3);  
 }  
}

##### 2.1.3 测试文件

public class DaoTest {  
 private SqlSessionFactory sqlSessionFactory;  
  
 @Before  
 public void beforeInfo() throws IOException {  
 String resource = "cn/edu/scnu/conf/mybatis-config.xml";  
 InputStream inputStream = Resources.*getResourceAsStream*(resource);  
 sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);  
 }  
  
  
 @Test  
 public void findById(){  
 UserDaoImpl userDao = new UserDaoImpl(sqlSessionFactory);  
 City city = userDao.findById(1);  
 System.*out*.println(city.toString());  
 }  
  
}

##### 2.1.4 总结

1. dao接口方法中存在大量模板方法（重复代码）；

2. statement的id硬编码 return sqlSession.selectOne("test.findById",1);  
 3. Sqlsession使用泛型，传入类型错误不会报错。

#### 2.2 Mapper代理方法

##### 2.2.1 编写Mapper.xml文件

<mapper namespace="cn.edu.scnu.mapper.CityMapper">  
 <select id="findById" parameterType="int" resultType="cn.edu.scnu.pojo.City">  
 SELECT \* FROM areas WHERE id = #{id}  
 </select>

</mapper>

##### 2.2.2 编写Mapper接口（相当于dao接口）

程序员在白那些mapper接口需要遵循一些开发规范，mybatis会自动生成mapper接口的实现类对象。

开发规范:

1. namespace = mapper接口

<mapper namespace="cn.edu.scnu.mapperInter.UserMapper">

2. mapper.java中的方法名 = mapper.xml中的statement的id

3. mapper.java接口中的方法输入参数类型和mapper.xml中的statement的

parameterType指定的类型一致。

4. mapper.java接口中的方法返回值类型和mapper.xml中的statement的

resultType指定的类型一致。

Mapper.java

public interface CityMapper {  
  
 City findById(int id) throws Exception;  
}

##### 2.2.3 测试

public class MapperTest {  
 private SqlSessionFactory sqlSessionFactory;  
 @Before  
 public void beforeInfo() throws IOException {  
  
 String resource = "cn/edu/scnu/conf/mybatis-config.xml";  
  
 InputStream inputStream = Resources.*getResourceAsStream*(resource);  
  
 sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);  
 }  
  
 @Test  
 public void findById() throws Exception {  
 SqlSession session = sqlSessionFactory.openSession();  
  
 CityMapper cityMapper = session.getMapper(CityMapper.class);  
  
 City city = cityMapper.findById(3);  
 System.*out*.println(city.toString());  
 }  
}

##### 2.2.4 总结

1. 如果返回单独一个对象，mybatis会默认调用selectOne，如果返回一个List类型，mybatis默认调用selectList

2. mapper接口方法参数只有一个，但是也可以使用包装类型的pojo以便满足不同的需求。

3. 注意：持久层dao可以使用包装类型宽展，但是业务层不建议使用包装类型。

### 第三章 全局配置文件—mybatis-config.xml

#### 3.1 properties定义

##### 3.1.1 db.properties

jdbc.driver = com.mysql.jdbc.Driver  
jdbc.url = jdbc:mysql://localhost:3306/city\_test  
jdbc.username=root  
jdbc.password=12345

##### 3.1.2 mybatis-config.xml

<configuration>  
<!--加载属性文件-->  
<properties resource="cn/edu/scnu/conf/db.properties"></properties>  
  
<!--加载属性，建议不要使用-->  
<properties>  
 <property name="" value=""></property>  
</properties>

<!-- 配置mybatis的环境信息，与spring整合，该信息由spring来管理 -->  
 <environments default="development">  
 <environment id="development">  
 <!-- 配置JDBC事务控制，由mybatis进行管理 -->  
 <transactionManager type="JDBC"></transactionManager>  
 <!-- 配置数据源，采用mybatis连接池 -->  
 <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>  
 <mappers>  
 <mapper resource="cn/edu/scnu/sqlMap/CityMapper.xml"></mapper>  
 <mapper resource="cn/edu/scnu/sqlMap/UserMapper.xml"></mapper>  
 </mappers>  
</configuration>

属性读取顺序：

先读取<properties>,再读取resource,最后读取parameterType传递的属性，会覆盖前者。

因此：

parameterType传递的属性最高级

resouce属性次之

<properties>属性最低

#### 3.2 settings全局参数配置

全局参数配置主要配置缓存，懒加载等内容。

<settings>  
 <setting name="" value=""/>  
</settings>

#### 3.3 别名定义

##### 3.3.1 默认别名



##### 3.3.2 自定义别名

在mapper.xml中，定义很多的statement，statement需要parameterType指定输入参数的类型、需要resultType指定输出结果的映射类型。

如果在指定类型时输入类型的全路径，不方便开发，可以针对parameterType和resultType指定的类型定义一些别名。

resultType="cn.edu.scnu.pojo.City"

mybatis-config.xml

<typeAliases>  
 <!--单个类型的别名定义-->  
 <typeAlias type="cn.edu.scnu.pojo.City" alias="City"></typeAlias>  
 <!--批量定义别名，别名就是类型，首字母大小写都行-->  
 <package name="cn.edu.scnu.pojo"></package>  
</typeAliases>

#### 3.4 类型处理器

#### 3.5 mapper的加载

### 第四章 映射文件配置—Mapper.xml

#### 4.1 输入映射—parameterType

#### 4.2 输出映射—resultType/resultMap

### 第五章 高级映射

#### 5.1 一对一映射

#### 5.2 一对多映射

#### 5.3 多对多映射

#### 5.4 延迟加载

### 第六章 查询缓存

#### 6.1 一级缓存

#### 6.2 二级缓存

### **第七章 mybatis和spring的整合**

#### 7.1 整合后的原始Dao开发

#### 7.2 整合后的Mapper代理开发