# Xml文档结构

## xmlns:名称空间

名称空间存在的意义：为了使两个不同的文档在使用相同的属性去描述不同的内容的时候发生命名冲突；

问题的解决：对于不同的DTD(校验文件 schema:样板)给她一个独一无二的标志来代表，在xml中使用的时候，须将这个DTD元素的标注和元素名，属性连在一起使用，这样就可以指明这个元素的来源了；在xml中使用url作为dtd文件的唯一标识；

<bean

xmlns:context="http://www.springframework.org/schema/context"

(context代表别名，url是真正的名称空间)

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

(这是schema名称空间（模式处理器位置），xml schema实例的名称空间通常前缀是xsi）

xsi:schemaLocation="http://www.springframework.org/schema/beans

<http://www.springframework.org/schema/beans/spring-beans.xsd>

(第一个url表示名称空间，第二个url表示模式位置)

>

</bean>

# eclipse中的文件

## .classpath文件

此文件描述了项目的运行时的依赖；

1. 项目的源文件：kind=”src”
2. 项目的运行环境:kind=”con”
3. 项目的输出位置：kind=”output” //web项目在WEB-INF/classes 下（classpath：就是指的时此路径）
4. 工程library的位置：kind=”lib”

## .project文件

用来描述工程的结构定义，像src，lib,classese的位置

<name></name>工程名称

<natures></natures>运行时需要的插件和具体加载的方式（<buildSpec></buildSpec>）

## .setting文件

都是一些项目的配置文件，可以在eclipse中进行修改的；

# 、aop切面编程

## juint4测试步骤

1. 导入jutint4 ,spring-text 的jar

@runwith(springJuint4ClassRunner.class)

@ContextConfiguration(“classpath:...xml，....xml”)(多个xml用逗号分割)

1. 常用的测试注解

@BeforeClass（只会运行一次，static方法） -> @Before （每个方法之前都会运行）-> @Test（测试方法） -> @After(同before)-> @AfterClass;

String[] args用来用来接受命令行参数，设计主要是为了接受用户的输出参数

// main方法执行之前会去将命令行参数传入到String[] 中

// 在 cmd 中以 类名 空格 参数 形式 传入 如 PsesonAction aaa bb cc

## AOP

1.xml 中开启注解扫描 <context:component-scan base-package="com.ss"></context:component-scan>

打开可以使用@aspect <aop:aspect-autoproxy proxy-target-class="true"/>

# HashMap xss

1. hashMap为数组+单向列表（每一个数组元素上都是一个单向列表）
   1. 单向列表中的每一个元素都是一个Entry(hashmap中的一个静态内部类)，包括四个属性（Object key ,Object value , final int hash, Entry next）；
   2. 内部实际为一个entry数组；初始最大长度为16，加载因子为0.75，阈值为12；初始化一个hashma'p时，所有数组的值都为null;
   3. 调用put(“key”，”value”);方法时；首先通过2方法查找到索引值，若当前索引没有值，则直接将当前的entry存进去；若有值，则会比较通过key的equals()进行比较，判断是否进行覆盖，通常新加入的entry在当前这个列表的头部；
2. HashMap是怎么通过Hash查找数组的索引的？

return h & (length-1);

每次扩容的时候最好是2的几次幂；这样最大程度的保存hash的一致性，尽量避免hash冲突；

1. Object 中的finalized方法： 没有被引用的对象会变成垃圾，垃圾回收器会在处理这个对象之前去调用finalized方法；（方法本身没有内容，需要被重写）；主要用于对象被回收之前的资源释放工作；
2. toString 方法：返回 this.getClass().getName()+“@”+该对象的hashCode值的16进制

# 反射

1.3个类

Java.lang.Class; （类）

java.lang.reflect.Method（方法）;

java.lang.reflect.Feild(属性)；

java.lang.reflect.Modifier(修饰符)；

Java.lang.reflect.Constructor;(构造方法)

1. Properties文件

Properties p=new Properties();

InputStream in=this.getClass().getScourseAsStream(“/init.properties”);

InputStream in=this.getClass().getClassLoader().getScourseAsStream(“init.properties”);

p.load(in);

String value=p.getProperty(key);

1. 构造方法

//调用User的带String和int类型的构造方法

Constructor c=User.getClass().getConstructor(String.class,int.class);

User u=c.newInstance();

1. 属性

Field[] fs=RefectClass.t().getDeclaredFields(); //得到所有的属性（包括私有）

Modifier.toString(fs.getModifiders());得到修饰符

fs.getType().getName();//得到类型

fs.getName()；//得到名称

fs.setAccessable(true);

User u=c.newInstance(); f.set(u, "wangwu"); //给属性赋值

1. wait()/notityAll()

都是object中的方法，都建立在synchronized 之下，前者时线程等待释放对象锁，后者唤醒所有等待的线程；

# Liunx常用命令

ls 查看

cd 跳转

cd /app/tomcat7/logs tail -f catalina.out 查看日志

ps -ef|grep tomcat7 搜索tomcat7这个进程

//重启tomcat过程

kill -9 XXXX 杀死XXXX进程

cd /app/tomcat7/bin/startup.sh 开启tomcat

clear 清屏

find . -name '\*elm\*' 查找中包含 elm字符的文件

Find -name test.txt 查找文件test.txt;

# spring注入和小结

## 7.1注入方式

### 7.1.1 set方法

List,map,String,java.util.Properties;注入；properties 注入和map区别是properties只能为String类型

1. <bean id="userVo" class="com.ss.test.di.UserVo">
2. <property name="name" value="张三丰"></property>
3. <property name="list">
4. <list>
5. <value>12</value>
6. <value>23</value>
7. <value>nihao</value>
8. <ref bean="user"/>
9. </list>
10. </property>
11. <property name="map">
12. <map>
13. <entry key="name" value="zhangsan"></entry>
14. <entry key="age" value="12"></entry>
15. <entry key="user" value-ref="user"></entry>
16. </map>
17. </property>
18. <property name="pro">
19. <props>
20. <prop key="username">root</prop>
21. <prop key="password">root</prop>
22. <prop key="url">jdbc:mysql://localhost:3306/test/</prop>
23. <prop key="driver">com.mysql.jdbc.Driver</prop>
24. </props>
25. </property>
26. </bean>
27. <bean id="user" class="com.ss.test.di.User"></bean>

### 7.1.2构造器注入

<constructor-arg index="1" value="1" type="int"></constructor-arg>

<constructor-arg index="0" value="zs" type="java.lang.String"></constructor-arg>

7.2

<![CDATA[<<可以写特殊字符><>]]>

## 7.2 bean 中的属性

Scope: 默认是单例的，内存中只有一个对象；

# 代码中发送一个http post请求

**//发送请求**

public static String sendStrPost(String url,JSONObject json){

//创建一个客户端

HttpClient client = new DefaultHttpClient();

//post请求

HttpPost post = new HttpPost(url);

String retSrc = null;

try {

//参数

StringEntity s = new StringEntity(json.toString(),"UTF-8");

//s.setContentEncoding("UTF-8");

//json格式

s.setContentType("application/json");

post.setEntity(s);

//发送请求，得到应答对象

HttpResponse res = client.execute(post);

//进行状态码比对

if(res.getStatusLine().getStatusCode() == HttpStatus.OK.value()){

HttpEntity entity = res.getEntity();

// 得到应答的字符串，这也是一个 JSON 格式保存的数据

retSrc = EntityUtils.toString(entity);

}

} catch (Exception e) {

throw new RuntimeException(e);

}

return retSrc;

}

**//接受请求**

public void TestPostRequest(){

HttpRequest req=getRequest();

//拿到请求对象中的字符流

BufferedReader reader=res.getReader();

StringBuffer sb=new StringBuffer();

//读取流中的数据

char[] ch=new char[1024];

int temp=0;

while((temp=sb.read(ch))!=-1){

sb.append(ch,0,temp);

}

//拿到json形式的数据

String jsonData=sb.toString();

//返回应答

HttpResponse resq=getResponse();

//设置应答头内容

resp.setHeader();

..

//写入数据

resp.getWriter().write("。。。。");

**}**

Pbulic void do(String url，String params){

Url url=new **//使用java原生的UrlConnection**

Url(url);

UrlConnection con=url.openConnection(url);

con.setDoInput(true);//获得输入流

con.setDoOutput(true);//输出流

con.setRequestProperty("content-type","application/json;chartset=utf-8");//设置请求头

con.setTimeOut()//设置超时时间

con.setUserCatch(false);//是否使用缓存

BufferWrite bw=new BufferedWrite(new OutputSteamWriter(con.getOutPutStream()));

bw.write(param);//写入参数

bw.close();

BufferReader br=new BufferedReader(new InputSteamReader(con.getInputStream()));//开始发动请求，并得到相应

br.readLine();//得到返回信息；

# }

# 小点

### //这个流，继承bufferedReader 可以用来读取一行数据

LineNumberReader line=new LineNumberReader(new FileReader("e://ashao.txt"));

int count=1;

String str=line.readLine();

while(str!=null){

System.out.println("第"+count+"行，"+"内容："+str);

str=line.readLine();

count++;

}

### //文件zip

ZipInputStream 和ZipOutPutStream;(org.apach e.tools.zip)

### //格式化数据

1.//%s:字符串；%d：整数；%b:boolean

String str= String.format("nihao,%s,%d,%b", "121",1,true);

System.out.println(str);

2. MessageFormat

String str2=MessageFormat.format("nihao,{0},{1},''{2}''", "121",1,true);

System.out.println(str2);输出：nihao,121,1,'true'

### //精确数据BigDecimal

1. 构造方法

不建议使用double类型的构造方法；应使用String的方法；

对于double应该使用Double.toString(double b ); 进行转化成String 类型

1. valueOf（double b）,valueOf(Long l)；

也可以使用valueOf( double b)进行double的转化;

1. 加，减，乘，除

add();加法

subtract（） 减

multiply（）乘

b.divide(a,new MathContext(3, RoundingMode.HALF\_DOWN)); 除

对于除法，应该有精度要求和舍入的方式（上面是四舍五入法）

BigDecimal主要用于精确计算，涉及到金钱方面的业务；

每一次运算都会产生一个新的BigDecimal对象

# 9.Spring定时任务

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

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:jee="http://www.springframework.org/schema/jee"

xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

<!-- 定时器 -->

<bean class="org.springframework.scheduling.quartz.SchedulerFactoryBean">

<property name="triggers">

<list>

<!-- 这里面可以写好多的定时任务 -->

<ref bean="payCheckJobTrigger" />

</list>

</property>

</bean>

<!-- 执行的周期 -->

<bean id="payCheckJobTrigger" class="org.springframework.scheduling.quartz.CronTriggerBean">

<property name="jobDetail" ref="payCheckJobDetail" />

<!--每天凌晨3点执行一次 -->

<property name="cronExpression" value="0 0 3 \* \* ?" />

</bean>

<!-- 执行的方法 -->

<bean id="payCheckJobDetail"

class="org.springframework.scheduling.quartz.MethodInvokingJobDetailFactoryBean">

<property name="targetObject" ref="payCheckJob" />

<property name="targetMethod" value="bestBillCheck" />

</bean>

<bean id="payCheckJob" class="com.ss.checkbill.action.PayCheckJob">

<property name="payCheckService" ref="payCheckService"></property>

</bean>

</beans>

# js知识

## //ajax传递数组

var arr=[];

arr[0]=1212;

arr[1]=12;

data:{

"data",JSON.stringify(arr);

}

后台使用String,接受，

使用jsonObject 和jsonArray解析

$.inArray(value,arr);//元素在arr数组中的位置，没有返回-1；

Arr.splice(index);//删除下标为index的元素；

## //ajax提交一个from表单

data:$('#formId').serialize() ;序列化表单数据，但是并不会对二进制文件进行操作

(“#formId”).serializeArray();

 JSON.stringify()方法是将一个JavaScript值(对象或者数组)转换为一个 JSON字符串//为jQuery注册一个全局函数

$.extend(

{

add:function(a,b){

return a+b;

}

}

);

$.add(2,3);即可调用 返回 5；

Ajax type=”DELETE”或”PUT”时，不能使用data属性传递参数；

ajax请求不能弹出下载文件的框；

//ajax异步可以上传文件， enctype=”multipart/form-data”

1. **var** formdata = **new** FormData(document.getElementById("myForm"));
2. $.ajax({
3. url: "dealing.php",
4. type: "POST",
5. data: formdata,
6. processData: **false**,  // 告诉jQuery不要去处理发送的数据
7. contentType: **false**   // 告诉jQuery不要去设置Content-Type请求头
8. });

## //jquery方法

Clone(); //复制元素

例：$(“#span”).append($(“p”).clone()); //span元素后面追加p标签

Closest() //查找当前元素的第一个祖先元素

例: $(“#li”).closest(“ui”);

## //进入页面加载

1.

(function($){

})(jQuery)

2.

$(document).ready(function(){

})

3.window.onload=function(){

}

4.

$(function(){

})

## // ifream

Parent.xml中：<ifream name=”” id=”childrea\_1” src=”children.xml”></ifream>

在parent.xml中调用children.xml中的function A();

$(“#childrea\_1”)[0].contentWindow.A();

在children.xml中调用父类的方法 B();

window.Parent.B();

$(“#childrea\_1”)[0].contentWindow；可以返回父页面的window对象；

Javascript 中所有的全局方法，变量，对象都是window对象中的方法或者属性；

window对象表示浏览器窗口；

常用属性

Window.top;获得最顶级的window对象；

Window.parent;获得父类的window对象；

Window.history.go(-1)等价于Window.history.back();等价于点击浏览器返回按钮;

Window.location.href=”url”；返回或者设置url;

常用方法

Window.open(url,”\_self/\_blank”,””);

## //闭包

简单来说，闭包是指可以访问另一个函数作用域变量的函数，一般是定义在外层函数中的内层函数。

Function getName(){

Var name=”张三”;

Var showName=function(){

Console.info(“我是”+name);

}

Return showName;

}

Var myName=getName()();// 我是张三；

//变量提升

对于 变量和函数（不包括var a=function(){console.info(“a”)}）js都会进行变量提升机制；

函数的优先级别大于变量；

例如：

Var c=”22”;

Function c(){

C=”0”;

Console.info(c);

}

C();

执行会报错，c is not a function;

以上代码相当于

Var c=function(){

C=”0”;

Console.info(c);

}

C=”22”;

所以会报错；

## //call()方法和apply()

****都是在特定的作用域中调用函数，等于设置函数体内this对象的值，以扩充函数赖以运行的作用域。****

一般来说，this总是指向调用某个方法的对象，但是使用call()和apply()方法时，就会改变this的指向，call(obj,arg1..);this会指向传入的obj对象；

window.color="red";

var obj={

color:"blue"

}

function changeColor(){

var color="black";

console.info(this.color);

}

changeColor.call(this);//red

changeColor.call(obj);//blue

changeColor.call();//red

changeColor();//red

//继承

无论什么时候，只要创建了一个新函数，就会根据一组特定的规则为该函数创建一个prototype属性，这个属性指向函数的原型对象，在默认情况下，所有原型对象都会自动获得一个constructor(构造函数)属性，这个属性包含一个指向prototype属性所在函数的指针。Person.prototype.constructor指向Person. 而通过这个属性我们可以继续为原型对象添加属性和方法；

第一种继承：

原型链



问题

1. 父类中的属性和函数可以被共享，但是不能被修改；
2. 子类不能向父类传递参数；

Call方法实现继承

function parent(name,age){

This.name=name;

This.age=age;

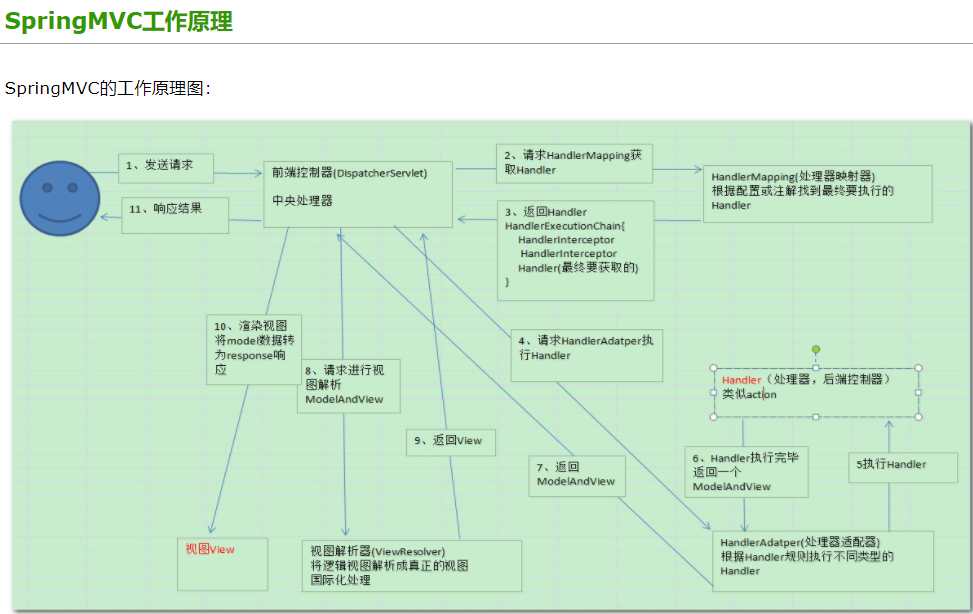
}

Function son(name,age){

//继承

Parent.call(this,name,age);  
}

# springMVC小结



## 配置

### Web.xml中

<!--springMVC的servlet映射 -->

<servlet>

<!--servlet姓名-->

<servlet-name>springMVC</servlet-name>

<!--前端控制器-->

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<init-param>

<!--初始化参数，加载springMVC.xml文件-->

<param-name>contextConfigLocation</param-name>

<param-value>classpath:springMVC.xml</param-value>

</init-param>

</servlet>

<servlet-mapping>

<!--映射-->

<servlet-name>springMVC</servlet-name>

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

</servlet-mapping>

<!-- 字符过滤器 解决乱码问题-->

<filter>

<filter-name>encding</filter-name>

<filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>

<init-param>

<param-name>encoding</param-name>

<param-value>UTF-8</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>encding</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

### 1.2 springMVC中

<!-- 注解配置扫描-->

<context:component-scan base-package="com.ss"></context:component-scan>

<!-- 配置器 -->

<bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter">

<!--ajax请求，传递json数据，需求导入两个jsonJack的包-->

<property name="messageConverters">

<list>

<bean class="org.springframework.http.converter.json.MappingJacksonHttpMessageConverter"></bean>

</list>

</property>

</bean>

<!-- 映射器 -->

<bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping">

</bean>

<!-- 视图解析器 -->

<bean

class="org.springframework.web.servlet.view.InternalResourceViewResolver">

<property name="prefix" value="/WEB-INF/jsp/"></property>

<property name="suffix" value=".jsp"></property>

</bean>

## 常用注解

注：需要在springMVC.xml中配置注解扫描

<context:component-scan base-package="com.ss"></context:component-scan>

### 2.1类上注解

@Controller controller上的注释，这是一个控制器，默认为单例的,在类上注释

@Service service层上的注释

@Repository dao层上的注释

@Component泛指组件，当组件不好归类的时候，我们可以使用这个注解进行标注。

### 2.2方法上的注解

@Autowire 默认通过byType自动注入，省略了get,set方法，（该接口只有一个实现类时）

若有多个，则需要配合@Qualifier("testAop")使用，”testAop”为在类上id

@Resource 相当于autowire默认通过byName进行注入；

@RequestMapping (value=”/hello”,param={“name”},method=RequestMethod.GET)

value:对应前台的请求路径;如<a href=”hello”>helloAction</a>

param:参数中必须带有name属性

mothod：请求方式必须是get请求；

## 3 拦截器

* 1. 实现HandlerInterceptor接口或者继承HandlerInterceptorAdapter

Perhandler(); 处理请求之前执行的方法；

Posthandler(); 开始返回页面的时候开始执行，可以对modelAndView执行

AfterCompletion();处理完请求之后开始执行；

* 1. 不拦截静态资源方法

一、spirngMvc.xml中加入 <mvc:default-servlet-handler/>启用默认加载servlet

Web.xml中配置需要由默认servlet加载的路径

<servlet-mapping>

<servlet-name>default</servlet-name>

<servlet-mapping>\*.js</servlet-mapping>

<servlet-mapping>\*.css</servlet-mapping>

二、 对应DispatcherServlet 的映射为 \*.do；这个servlet只处理已.do结尾的请求地址；

3.2 不拦截某写请求地址

可在拦截器中注入不拦截的请求字符串集合；

# 12 hibernate

[IMG_256](https://www.cnblogs.com/whgk/p/javascript:void(0);)

//Student实体类public class Student {

private Integer sid;

private String sname;

//用set集合来保存选的多个课程

private Set<Course> courseSet = new HashSet<Course>();

set、get.....

｝

//Student.hbm.xml

<class name="domain.Student" table="student">

<id name="sid" column="sid">

<!-- 主键生成策略 -->

<generator class="increment"></generator>

</id>

<!-- 一些常规属性 -->

<property name="sname"></property>  
  
<!-- 关键的地方就在这里了。一定要搞清楚两个column分别指的是什么意思 脑袋中要有哪个数据库关系图-->  
  
<!--要查询到所有的course，就需要通过连接表，所以申明连接表的名称-->

<set name="courseSet" table="student\_course"> <!-- 本实体类在连接表中的外键名称，过程我们上面分析的很清楚了，为什么需要这个呢？让hibernate知道连接表中有一个外键名为s\_id的指向本实体类 -->

<key column="s\_id"></key>

<!-- 多对多映射关系，映射类和其映射类在连接表中的外键名称 这个的意思跟上面的一样，也是声明让hibernate知道，这样一来，hibernate就知道如何查询了-->

<many-to-many class="domain.Course" column="c\_id"></many-to-many>

</set>

</class>

[IMG_257](https://www.cnblogs.com/whgk/p/javascript:void(0);)

　　　　　　Course和Course.hbm.xml

[IMG_258](https://www.cnblogs.com/whgk/p/javascript:void(0);)

//Course实体类public class Course {

private int cid;

private String cname;

private Set<Student> studentSet = new HashSet<Student>();

...

｝

//Course.hbm.xml 有了上面的分析，这个就简单了，内容和意义跟上面的一模一样。

<class name="domain.Course" table="course">

<id name="cid" column="cid"> <!-- 主键生成策略 -->

<generator class="increment"></generator>

</id> <!-- 一些常规属性 -->

<property name="cname"></property>

<set name="studentSet" table="student\_course"> <!-- 本类在连接表中外键的名称， -->

<key column="c\_id"></key>  
　　　　　　<!--多对多映射关系，映射类和其映射类在连接表中的外键名称-->

<many-to-many class="domain.Student" column="s\_id"></many-to-many>

</set>

</class>

[IMG_259](https://www.cnblogs.com/whgk/p/javascript:void(0);)

# 12 调用存储过程

## 12.1创建一个简单的存储过程

（输入员工编号，输出姓名，月薪，职位）

create or replace procedure queryempinfo(eno in number,

pename out varchar2,

psal out number,

pjob out varchar2)

as

begin

--得到该员工的姓名 月薪和职位

select ename, sal, job into pename, psal, pjob from emp where empno = eno;

end;

//参数中不能带范围

Create or replace procedure pro\_test(num\_code in number,num\_name out varchar)

As

--声明变量，可以带范围

num\_code\_1 number(10);

Num\_name\_1 varchar(20);

Begin

num\_code\_1 :=numcode;

Num\_name\_1 :=num\_name;

Work\_str :=’1’;--字符串类型带冒号；

Work\_num :=1;--整形不用带冒号

--申明一个游标

Declare

course cour\_num is select 字段1,字段2,字段3 from user\_info;

Order\_i cour\_num%rowtype;

Begin

For order\_i in cour\_num loop

Insert into user\_info valeus(order\_i.字段1，order\_i.字段2，order\_i.字段3);

End loop;

End ;

--if语句

If(...) then

...

elseif(...) then

....

else ...

end if;

--异常处理

Exception

End;

## 12.2 java代码中调用

String url = "jdbc:oracle:thin:@127.0.0.1:1521:orcl";

String driverName = "oracle.jdbc.driver.OracleDriver";

String username = "scott";

String password = "\*\*\*\*\*\*";

String sql = "call queryempinfo(?,?,?,?)";

// 调用存储过程

public void callProcedure() {

try {

Class.forName(driverName);

conn = DriverManager.getConnection(url, username, password);

stat = conn.prepareCall(sql);

// 一个输入参数和三个输出参数

stat.setInt(1, 7566);

stat.registerOutParameter(2, OracleTypes.VARCHAR);

stat.registerOutParameter(3, OracleTypes.NUMBER);

stat.registerOutParameter(4, OracleTypes.VARCHAR);

stat.execute();

String name = stat.getString(2);

int sal = stat.getInt(3);

String job = stat.getString(4);

System.out.println("name: " + name + ", sal: " + sal + ", job: " + job);

} catch (Exception e) {

e.printStackTrace();

} finally {

close(conn, stat, rs);

}

}

## 12.3触发器

当对emp表有插入，删除，更新操作时，向emp2表插入旧值；

Create or replace trigger tri\_add\_emp

Before/after insert/delete/update on emp[表名]

For each row (对于每一行)

As

Declare

--本地的变量

Begin

Insert into emp2 values(:old.id,:old.name:old.age,:old.sex);

End;

# 13解析dom节点

public static String getXml(){

String xml="";

xml+="<?xml version=\"1.0\" encoding=\"GB2312\" standalone=\"no\"?> ";

xml+="<books> ";

xml+=" <book email=\"zhoujunhui\">";

xml+=" <name>rjzjh <age>23</age> </name> ";

xml+=" <price>jjjjjj</price>";

xml+=" </book> ";

xml+=" </books> ";

return xml;

}

public static String getTarget(String xml,String targerNode) throws Exception{

//字节流

ByteArrayInputStream is=new ByteArrayInputStream(xml.getBytes("utf-8"));

//dom节点解析器工厂

DocumentBuilderFactory factory=DocumentBuilderFactory.newInstance();

//得到一个dom节点解析器

DocumentBuilder build=factory.newDocumentBuilder();

//解析xml数据

Document document= build.parse(is);

NodeList list= document.getElementsByTagName(targerNode);

Node n1=list.item(0);

System.out.println(n1.getNodeName()); // name

Node n2=n1.getFirstChild();

System.out.println(n2.getNodeName()); //#text

String s= n2.getNodeValue();

System.out.println(s); //rjzjh

return null ;

}

# 14 使用xfire创建一个webservice，并在代码中调用

## 引入相关的jar包

## HelloWorld 实现HelloWorld接口

## application.xml配置

<!-- 引入xfire配置-->

<import resource=”classpath:org/codehaus/xfire/spring/xfire.xml”>

<!-- webservice配置 --**>**

<bean id="baseWebService" class="org.codehaus.xfire.spring.remoting.XFireExporter" lazy-init="false" abstract="true">

<property name="serviceFactory" ref="xfire.serviceFactory" />

<property name="xfire" ref="xfire" />

</bean>

<!---业务bean-->

<bean id="helloWorld" class="com.ss.hello.service.impl.HelloWorldImpl"></bean>

<bean id="helloWordlSerivce" parent="baseWebService">

<property name="serviceBean" ref="helloWorld"></property>

（参考业务bean）

<property name="serviceClass" value="com.ss.hello.service.HelloWorld"></property>

（接口地址）

</bean>

## web.xml配置

1. <context-param>
2. <param-name>contextConfigLocation</param-name>
3. <param-value>classpath:applicationContext.xml</param-value>
4. </context-param>
5. <listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

1. </listener>
3. <servlet>
4. <!-- 配合Spring容器中XFire一起工作的Servlet -->
5. <servlet-name>xfire</servlet-name>
6. <servlet-class>org.codehaus.xfire.spring.XFireSpringServlet</servlet-class>
7. </servlet>
8. <servlet-mapping>
9. <servlet-name>xfire</servlet-name>
10. <!-- 在这个URI下开放Web Service服务 -->
11. <url-pattern>/service/\*</url-pattern>
12. </servlet-mapping>
13. 启用即可发布

访问地址：<http://localhost:8080/webInterface/service/HelloWorld?wsdl>即可测试

### Java代码中调用

注意：调用的项目中必须存在和webservice中一样的类，包名也必须相同；

main方法中调用

Service service=new ObjectServiceFactory().create(HelloWorld.class);

XFireProxyFactory factory = new XFireProxyFactory();// 创建工厂实例

String url="http://127.0.0.1:8080/webInterface/service/HelloWorld";

try {

HelloWorld hello=(HelloWorld) factory.create(service,url);

System.out.println(hello.sayHello("zhangsan"));

} catch (MalformedURLException e) {

e.printStackTrace();

}

1. 通过 wsimport 命令生成客户端代码；wsimport -keep -d D:\temp\d -s D:\temp\s -p com.map -verbose http://ws.webxml.com.cn/WebServices/MobileCodeWS.asmx?wsdl

-keep：是否生成java源文件

-d：指定.class文件的输出目录

-s：指定.java文件的输出目录

-p：定义生成类的包名，不定义的话有默认包名

-verbose：在控制台显示输出信息

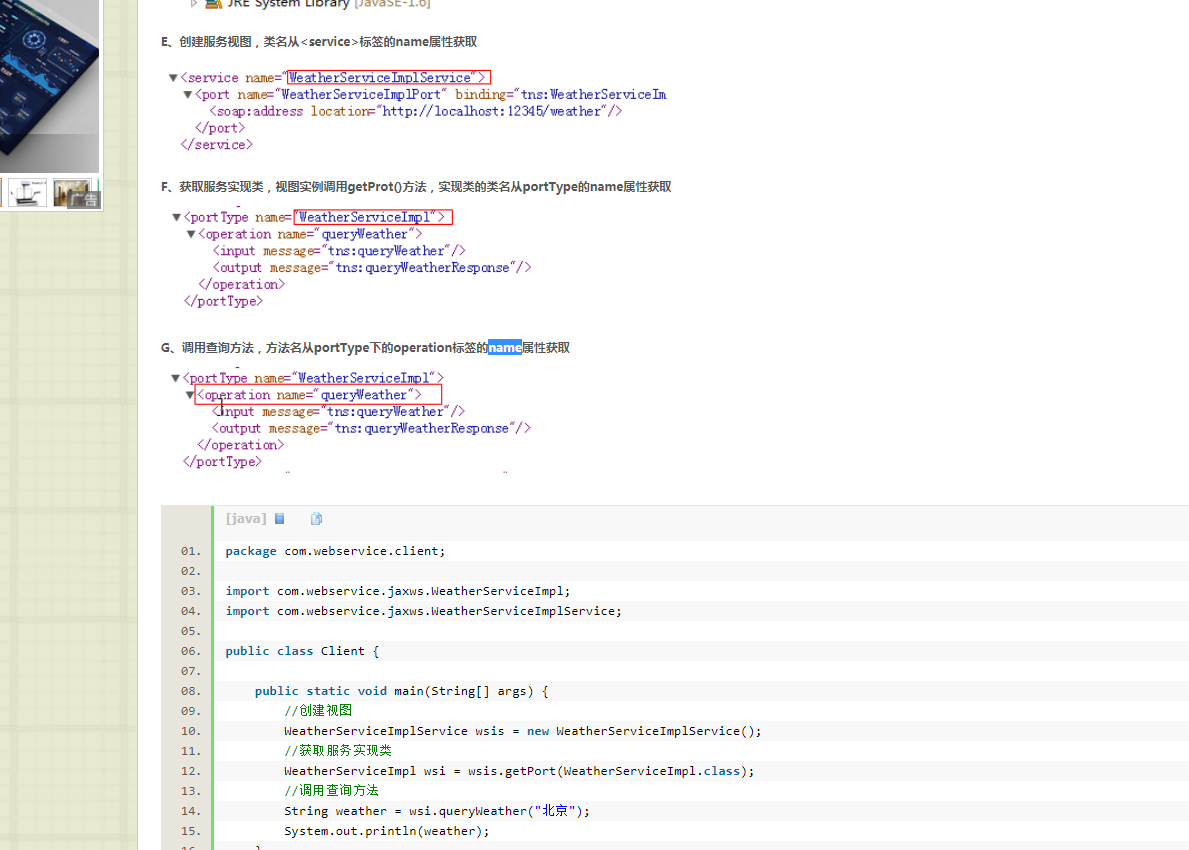
-b：指定jaxws/jaxb绑定文件或额外的schemas

-extension：使用扩展来支持SOAP1.2

浏览器中打开webservice接口地址主要查看3个元素

1. service ：服务视图，从service标签的name属性获取
2. 获取服务的实现类：从porttype标签的name属性中获取

3；调用对应的方法：从porrtype标签下面的operation标签中获取



//oracle中字符长度   
varchar2最大是4000字节，那么就看你的oracle字符集：（select userenv('language') from dual;）  
如果字符集是16位编码的，ZHS16GBK，那么每个字符16位，2字节，所以可以容纳2000字符。  
如果是32位编码的字符集，那么只能存储 1000个字符。

varchar(50) 表示该字段类型为varchar2类型，长度为50，可以存单字节字符50个，如字母、数字等，可以存储双字节字符25个，如汉字。不同的是，若输入的不足，则以空格填充。  
varchar2(50) 表示该字段类型为varchar2类型，长度为50，可以存单字节字符50个，如字母、数字等，可以存储双字节字符25个，如汉字。  
nvarchar2(50)表示该字段类型为nvarchar2,长度为50,不论英文，数字，中文都能存50个。

//web项目中获得各种资源的路径

1. 通用，可在任何一个class中使用，获得当前这个项目calss文件存放绝对路径

this.getClass().getClassLoader().getResource(“”).getPath();

同

Thead.getCurrentThead.getContextClassLoader().getResource().getPath(“timer.xml”); （推荐）

例如：web项目E:\eclipseM9/workspace/tree/WEB-INF/classes/

普通java项目：/E:/workspace/lpl/bin/ /E:/workspace/lpl/bin/timer.xml

//题外，直接返回这个文件的输入流的形式

InputStream in= Thread.currentThread().getContextClassLoader().getResourceAsStream("file.properties")

1. ServletActionConext.getActionContext.getRealPath(“/upload”);

获得此文件在web项目中的绝对路径，从此项目在tomcat部署的路径开始算吧，一般以“/”开头，upload一般放在webRoot下；例如：E:\tools\apache-tomcat-8.0.41-windows-x86\apache-tomcat-8.0.41\wtpwebapps\stupmcn\upload

使用classLoader 是从项目的class路径开始查找文件

File file =new File(String 文件路径); file.getAbsolutePath();//从项目启动bin目录下开始查找文件；

# 15：http与https

1. http是http协议运行在TCP协议的80端口上，所有的传递信息都是明文，从request中可以拿到；
2. https是http协议运行在ssl之上，ssl运行在TCP协议的443端口上，传递信息经过ssl证书加密；（https=http+ssl）

http 1.1之前都是短连接，从1.1开始长连接；

短链接：请求一次之后立即断开；

长连接：一次请求之后不会立即断开，直到该页面中的所有请求都完成；

# 16 Redis（19、20、21、22）

1. NOSQL: not only sql,解决高并发情况下的读写问题；海量数据的高效率读写和访问；高扩展性和高可用性
2. Java中使用jedis连接Redis数据库；导入jedis-2.9.0.jar
3. JedisPoolConfig config =new JedisPoolConfig();
4. //最大连接数
5. config.setMaxTotal(30);
6. //最大空闲连接数
7. config.setMaxIdle(10);
8. //最长等待时间
9. config.setMaxWaitMillis(5000);

//连接池

1. JedisPool jedisPool=new JedisPool(config, host, port);
2. //主对象
3. Jedis jedis=jedisPool.getResourse();

数据类型

1. String 类型

//存值

Jedis.set(“key”,”value”); 成功返回 “OK”

//取值

Jedis.get(“key”);

//调用一次其值加一，从0开始;

Jedis.incr(“key”); jedis.decr(“key”);

//删除

Jedis.del(“key”);

//判断是否存在

Jedis.exist(“key”);

//设置过期时间

Jedis.expire(“key”,int second);

//取消过期时间

Jedis.persist(“key”);

//设置key-value的同时，设置过期时间

//计数-增加 从1 开始，每次加一，返回当前的值

Long num=Jedis.incr(“number”);

//计数--减少 每次减一，返回当前的值

Jedis.decr(“number”);

//追加，append();

Jedis.append(“key”,”value”);如key不存在则正常添加，存在则在之前的value值后追加

1. List 类型（列表）

//增加(将值插入到列表的首位)

Jedis.lpush(“key”,”value1”,”value2”....);

//增加（末尾添加）

Jedis.rpush(“key”,values);

//删除（删除count个值为value1的数据，count=0则删除全部）

Jedis.lrem(“key”,count,”value1”);

//修改 修改key中下标为count的数据为value(下标从0开始)

Jedis.lset(“key”,count,”value”);

//查看(start-end的值，如果end=-1则查看全部)

Jedis.lrange(“key”,int start,int end);

//根据下标查询

Jedis.lindex(“key”,int index);

//获取列表长度

Jedis.llen(“key”);

//获取第一个

Jedis.lpop(“key”);

//获取最后一个

Jedis.rpop(“key”);

三 hash类型（哈希表）

//添加

Jedis.hset(“key”,”field”,”value”);

//删除

Jedis.hdel(“key”,”field”);

//查看某一个key的值

Jedis.hget(“key”,”field”);

//得到所有

Jedis.hgetAll(“key”,”field”);

//得到所有的keys

Jedis.hkeys(“key”);

//得到所有的value

Jedis.hvals(“key”);

//长度

Jedis.hlen(“key”);

//自增(调用一次增长num)

Jedis.hincrby(“key”,”field”,int num);

四 set类型（集合，没有重复元素）

//添加

Sadd(“key”,”value”);

//删除

Srem(“value”);

//查看所有成员

Smembers(“key”);

//长度

Scard(“key”);

//检查元素是否存在集合中

Sismember(“key”);

//交集运算

Sinter(“key1”,”key2”,”key3”);

//并集运算

Sunion(“key1”,”key2”..);

//差集运算,返回存在第一个集合，不存在其他集合中的元素

Sdiff(“key1”,”key2”..);

1. Sort set 集合（可以进行排序）

//增加

Zadd(“key”,int score,”value”);

或 zadd(“key”,map<String,double> map)

//删除元素

Zrem(“key”,”value”);

//从小到大输出

Zrange(“key”,0,-1);

//反向输出

Zrevrange(“key”,0,-1)

//查询莫一个范围之前人员个数

Zcount(“key”,double b1,double b2);

//查询某一个成员的分数

Zscore(“key”,”value”);

//查询莫一个人的排名

Zrank(“key”，”value”);

# 17 oracle

## 17.1分区表

表空间：一个或多个数据文件的集合，通常文件是指表；

分区表：通过某一种维度将一张完整的表在物理上分为很多小表，储存在不同的表空间中，但在逻辑上仍然是一张表；

优点：

1.提高查询速度；查询时只需查找当前分区中的数据，不需要全表扫描；

Select \* from user\_info where add\_time>to\_date(‘2018-04-20’,’yyyy-mm-dd’);

如果以add\_time进行分区，此sql会自动定义到符合条件的分区中；

2.一个分区中损坏，不会影响其他分区中的数据；

创建分区：

2.1按时间自动分区

Create table user\_info(

id number(10) primary key,

User\_name varchar2(20),

Add\_time date

)

Partition by range(add\_time)

Internal ((numtoyMinterval (1,'MONTH'))) //按时间自动建立分区，每一个月建立一个

(

Partition p1 values less then(to\_date(‘2018-04-01’,’yyyy-mm-dd’)) tablespace newGanSu

)

2.2 范围分区

Create tabel user\_info(

Id number(10) primary key,

User\_name varchar2(20),

Add\_time date

)

Partition by range(id)

(

Partition p1 values less then (10000) tablespace test\_space\_1,

Partition p2 values less then(20000) tablespace test\_space\_2,

Partition p3 values less then(maxvalue) tablespace test\_space\_3

)

//id的值小于10000存在p1分区中，大于10000小于20000在p2中，大于20000在p3中

查询某一个分区：select max(request\_date) from WX\_VISIT\_INFO partition(p1)

## 17.2 oracel常用函数：

1.To\_char();

2.To\_date();

3.Nvl();

4.wm\_concat(列名)

5.Decode(value,if 1,then 1,if 2,then 2,else ,then 3);

含义为

If value ==if 1

输出1；

If value == if 2

输出2

否则

输出3；

1. round(1/3,2);四舍五入保留2位小数；
2. Trunc();截取字符串；



# 18 sql优化

1. In、 not in 和exists、not exists

执行顺序

In：先进行in语句中的子查询，后检查外部查询中的结果在子查询中是否存在；

Exists:先进行外部的查询语句，后进行子查询语句，有查询结果返回true,并保留结果否则返回false,删除该条语句；

效率问题：

子查询结果小又有索引，外部查询结果多的情况新则应该使用in；反之应该使用exist;

not in 一定会进行全表扫描，不会用到索引；

Not Exists的子查询中依然会用到索引，所以 not exists 一定会比not in 快；

1. Is null 和 is not null(判断字符是否为空)

这种写法会放弃索引，索引字段上不会有空值；也就是说某列上有空值，即使建立索引也不会提高性能；

建议： a >”” 或 a>0

1. Like 操作符

Like “王%” 使用索引，

Like “”

like”%369%” 不会使用索引，

总结：“%”在前面是不会使用索引的；

Oracle 中使用 reverse（“123456”） ---》654321

一、Reverse( code ) like reverse(“code%“) --> code like “%code”

并建立函数索引

二、Oracle中 instr(t.code ,”code ”)>0 code 在t.code中出现的位置；

4. <> 和！=

都不会使用索引进行查询，会进行全表扫描；

建议使用 > and <组合；

1. 减少访问数据库的次数
2. 避免使用 “\*”
3. Where 子句中的连接顺序；

Oracle 采用字下而上的顺序进行解析，where 子句根据这个 原理， 那些连接多个表之间的条件应该在where子句最前面，能够过滤掉大数据量的条件应该在最末尾；

# 19 设计模式

1. 单例

应用场景：

1. 只有唯一的一个实例
2. 为其他对象的调用提供一个方法
3. 单例类里面必须自己创建一个自己的实例
4. 工厂

场景：

不知道具体应该用哪一个类的实例时，可以使用工厂；

1. 建造者

（1）相同的方法，不同的执行顺序，产生不同的事件结果时，可以采用建造者模式。

（2）多个部件或零件,都可以装配到一个对象中，但是产生的运行结果又不相同时，则可以使用该模式。

（3）产品类非常复杂，或者产品类中的调用顺序不同产生了不同的效能，这个时候使用建造者模式是非常合适。

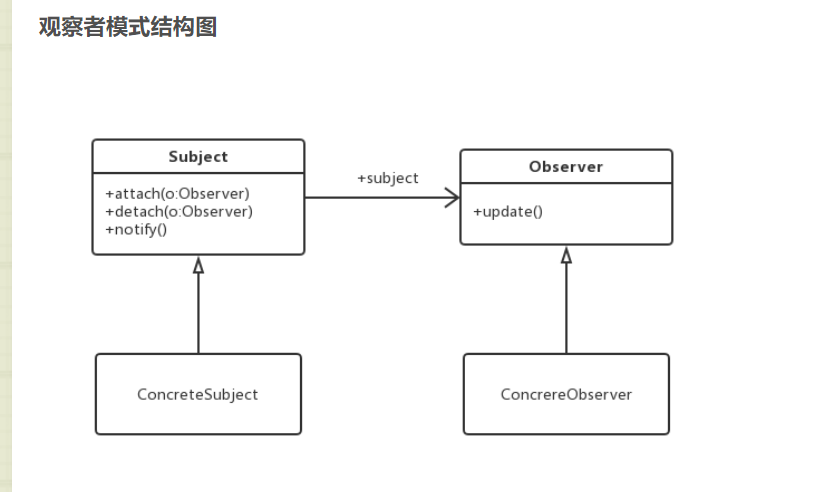
（4）在对象创建过程中会使用到系统中的一些其它对象，这些对象在产品对象的创建过程中不易得到时，也可以采用建造者模式封装该对象的创建过程。该种场景，只能是一个补偿方法，因为一个对象不容易获得，而在设计阶段竟然没有发觉，而要通过创建者模式柔化创建过程，本身已经违反设计最初目标

4. 装饰者模式

对已有的业务逻辑进一步的封装，使其增加额外的功能，如java中的IO流就使用了者模式，用户在使用的时候，可以任意组装，达到自己想要的效果。

1. 观察者模式

对象间一对多的依赖关系，当一个对象的状态发生改变时，所有依赖于它的对象都得到通知并被自动更新。



# 20 easyUI



## 常用功能---表格(Datagrid)

1 初始化表格数据

Var dataGrid=null;

$(function(){

dataGrid=$(“#表格id”).datagrid({

url:”请求数据的地址，数据返回形式为json数组,json的key与name对应”，

fitColumns:true;//列宽自适应;最后一列的宽度为width=’1’

Rownumbe:true;

sortName:”排序字段”，

SortOrder:’asc/desc’,

Pagination:true;true,显示工具栏

Rownumbers:true,显示行号

Onselect:function(index,row){

//选中的时候触发 index:索引；row:行对象；row.name可以获得name的值

}

onDbClickCell:funtion(index,row){

}

})

})

2.刷新页面数据

Var param={};

Param.name=name;Param.id=id;查询参数；

$(“#表格id”).datagrid(‘reload’,param);带参数提交；

$(“#表格id”).datagrid(‘load’);刷新页面；

3列属性：

在<th

Align=ligth,

Formatter=nameFormatter

>中使用

Function nameFormatter(value,row,ndex){

Value:字段值；

Index:索引；

Row：行属性；

}

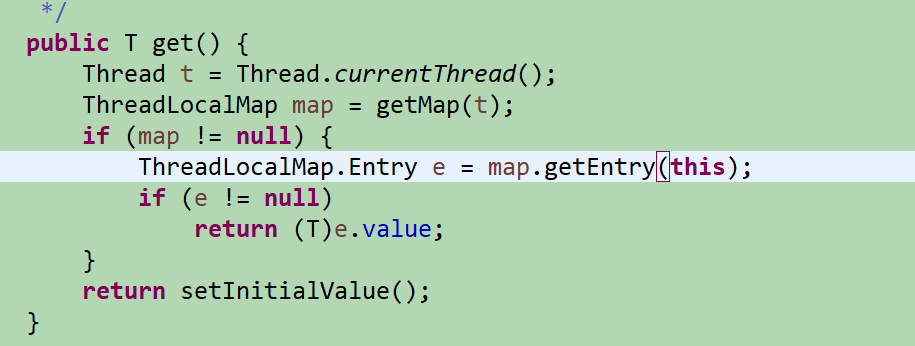
// 可以参考easyUI文档；

# ThreadLocal

为每一个使用该变量的线程提供了一个单独的副本，各线程之间使用互相不受影响；

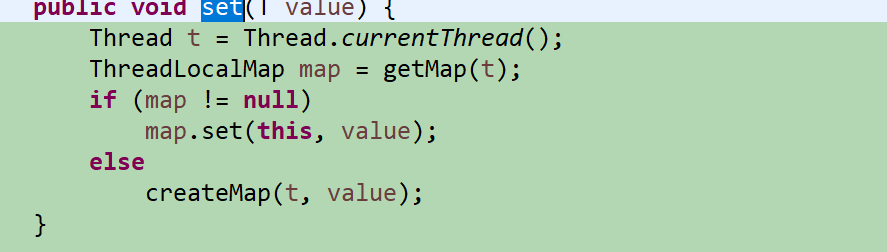
主要方法：

get();



ThreadLocalMap：作为ThreadLocal中的一个静态内部类,其本质为一个map；

Set();



# 攻击

## Xss攻击(跨域脚本攻击)：

攻击者往web网页中插入恶意的html 或则 script代码，当用户浏览的时候，嵌入的代码就会被执行，从而破坏用户的；

1. 反射型；xss代码出现在url中，作为输入参数提交到服务器，经过解析后，xss代码随相应内容一起传回浏览器，被浏览器进行解析；

<img src=”null” onerror=”alter(123)”/> //xss会自动执行；

1. 存储型；xss代码进入服务端，存储在文件系统或数据库中，下次请求时，随相应内容一起被浏览器解析；

防御：

1. html entity编码 ； 对于特殊字符 <,>,””,’’,&进行编码；
2. 过滤

移除包含script,javascript,style,ifream和onclick,onerror的参数；

## sql注入：

用户输入恶意的sql语句，提交到服务端，并进行sql查询；

1. 采用预编译的方式；使用ParperStatment对象；

采用预编译绑定变量的 SQL 语句而不是直接拼接 SQL 语句，sql引擎会预先对待执行的sql语句进行分析，产生语法树；后面你无论输入的是什么参数，都不会影响语法树的整体结构；而且采用预编译的方式，sql语句的执行效率更高；

Csrf攻击

# 23 session 和cookies

http协议是一种无状态的协议，第二次访问页面的时候，服务器没有办法知道当前用户是否为上一次访问页面的用户；由此产生cookie技术；

用户第一次访问服务器的时候，服务器会生成一个sessionid，返回客户点，客户端会将其保存在cookie中（请求头中可以看到**Cookie:** JSESSIONID=33D3C551640AB9CB2BB0D611E303E77B，此cookie没有设定保存的时间，因此浏览器关闭后cookie也会失效），以后用户访问页面的时候，浏览器会将此jsessionId上传到服务器，服务器根据此jsessionId 可以获得，session中的信息；用来判断当前用户是否生效；

**Session**

服务器端维持用户信息的一种机制；当创建session是，首先会检查此session有没有一个与之相关联的sessionid;如果有，就通过此sessionid查找到session；若没有就创建session并创建一个与之相关联的sessio

**Cookies**

Cookies是服务器客户端保留服务器信息的一种机制；

### **Jsessionid**

这是tomcat容器的一种叫法，其实就是sessionid,其他容器也许就不叫jsessionId了；

# Mongdb

段

\_id--->自动Nosql;c++编写；json格式；基于分布式文件存储的非关系型数据库；

Database-->数据库

Collection -->表

Document-->row{“name’:”张三”,”age”:”23”}

Field---->>字生成的一个唯一标识

默认数据库为 db, 默认存储在 data 目录下;

操作文件：GridFs :

上传的大文件会被分割成多个chunk(小的文件);一般为256kb左右；每个chunk会作为一个文档（document）被添加chunks的集合中；

//保存

Db.collectionName.save( document ); 或者 db.collectionName.insert( document );

集合名若不存在，则会创建；

//删除

Db.collectionName.remove( query条件 );

//修改

Db.collectionName.update(query,set,upset,multi)

Query:查询条件；

Set:需要被改变的值；{$set:{“name”:”zhangsan”}}

Upset:true/false; 如果不存在是否添加；默认false；

Multi: true/false; 是否更新全部的记录；默认false,只更新查到的第一条记录；

//查看

Db.collectionName.find(query条件); 或者 db.collectionName.findOne( query条件 ); 只返回一条;

多个条件 and查询

find({“name”:”zhangsan”,”age”:”30”})---->where name=”zhangsan” and age=30

或者 or查询

find({$or:[{“name”:”zhangsan”},{”age”:”30”}]) ----> where name=”zhangsan” or age=30

大于($gt)，大于等于($gte)，小于($lt)，小于等于($lte)，等于($eq)， 不等于($ne)

$gt -------- greater than >

$gte --------- gt equal >=

$lt -------- less than <

$lte --------- lt equal <=

$ne ----------- not equal !=

$eq -------- equal =

db.documentName.find({“age”,{$gt:60}});

Java中调用mongDb;

1. 引入相关的jar包
2. 与spring进行整合，注入MongoTemplate(文档对象)，GridFsTemplate(文件对象)
3. 通过MongoTemplate 完成document的增删改查，使用GridFsTemplate完成对文件操作

MongoTemplate对象；

//保存

BaseObject obj=new BasicObject();

Obj.put(“name”,”张三”);

ojb.put(“age”,23);

....

mongoTemplate.save(obj,”表名称”);//若没有当前表则会重新创建；

{

“\_id”:Object(“161321dfa61fd3af1e5”),

“name”:”张三”,

“age”:23

}

//删除

Query query=new Query( );

query.addCriteria(Criteria.where(“\_id”).is(“value”)); --->where id=value

mongoTemplate.remove();

//更新

Query query =new Query();查询条件；

Update update=new Update();

Update.addSet(String key,String value);//需要更新的字段

String collectionName=””;数据库名称;

MongodbTemplate.updateFirst(Query query,Update,String collectionName);//更新第一个

# Shiro;

# 单点登陆（SSO）

# Sring 源码

# SpringMVC源码

1. springMVC最主要的类为DisPatcherServlet,本质为一个servlet;
2. 项目启动时初始化各种类并进行依赖注入，spring容器本质为一个map；
3. servlet的init方法中进行类的初始化；
   1. 获取项目中所有class文件的路径封装到list中；
   2. 初始化对象

通过反射获取所有的类实例；Class.forName();以注解的value为key，对象作为value；

将所有的对象添加到map集合中；

* 1. 依赖注入

从map中获取所有对象中的属性通过field.set(Object obj1,Object obj2);将对象注入到属性中去； 例：Student类中有Class对象；obj1为Student对象，obj2为Class对象；

3.4 方

# 28.Springboot

1. 主要注解：

(1)@SpringBootApplication启动类上的注解，其中包含3个注解

@Configration 配置类注解，类上使用该注解可以在其中配置bean对象

@CompoentScan 扫描包，默认扫描当前包下的注解

@EnableAutoConfigration ..

1. @Configration + @PropertySource(value="classpath:init.properties") 读取配置文件

在另一个属性类中，在属性上使用@Value(“${name}”) 可以获取init中的name值

1. @Compoment+ @ConfigrationProperties();//读取application.properties中的文件内容
2. @Import; 注入一个类，由spring进行管理；
3. BeanPostProcessor接口，初始化bean之前可以进行操作，方法必须由返回值；
4. @ComponentScan 包扫描路径，默认为当前包下路径；
5. 事件监听
6. Springboot扩展分析

（1）BeanPostProcessor接口

Bean在初始化的时候执行；

（2）ApplicationContextInitializer

Spring容器在refresh之前进行执行；之后再context.addInitializer(..);

1. CommandLineRunner 接口

实现这个接口，并纳入spring管理之中，表示在spring容器初始化完成之后开始执行；当有多个实现类时，通过@order();控制执行顺序； 类似开机启动

1. 补充

(1).classpath下新建banner.txt文件；可以在控制台输出，文件内容；

(2).设置默认配置,application.properties文件中没有的时候会，读取默认配置

App.setDefaultProperties(Map map)

1. Web

注解类似于springMVC;若不支持jsp页面必须加入

Application.properteis文件中

1. .配置tomcate port : server.prot=8089;
2. 配置应用上下文；server.contextPath=/spring-boot;
3. 配置返回页面的路径

前缀：spring.mvc.view.prefix=/WEB-INF/jsp/

后缀：spring.mvc.view.suffix=.jsp

1. 使用freemarker模板方法

一：pom文件中导入

<dependency>

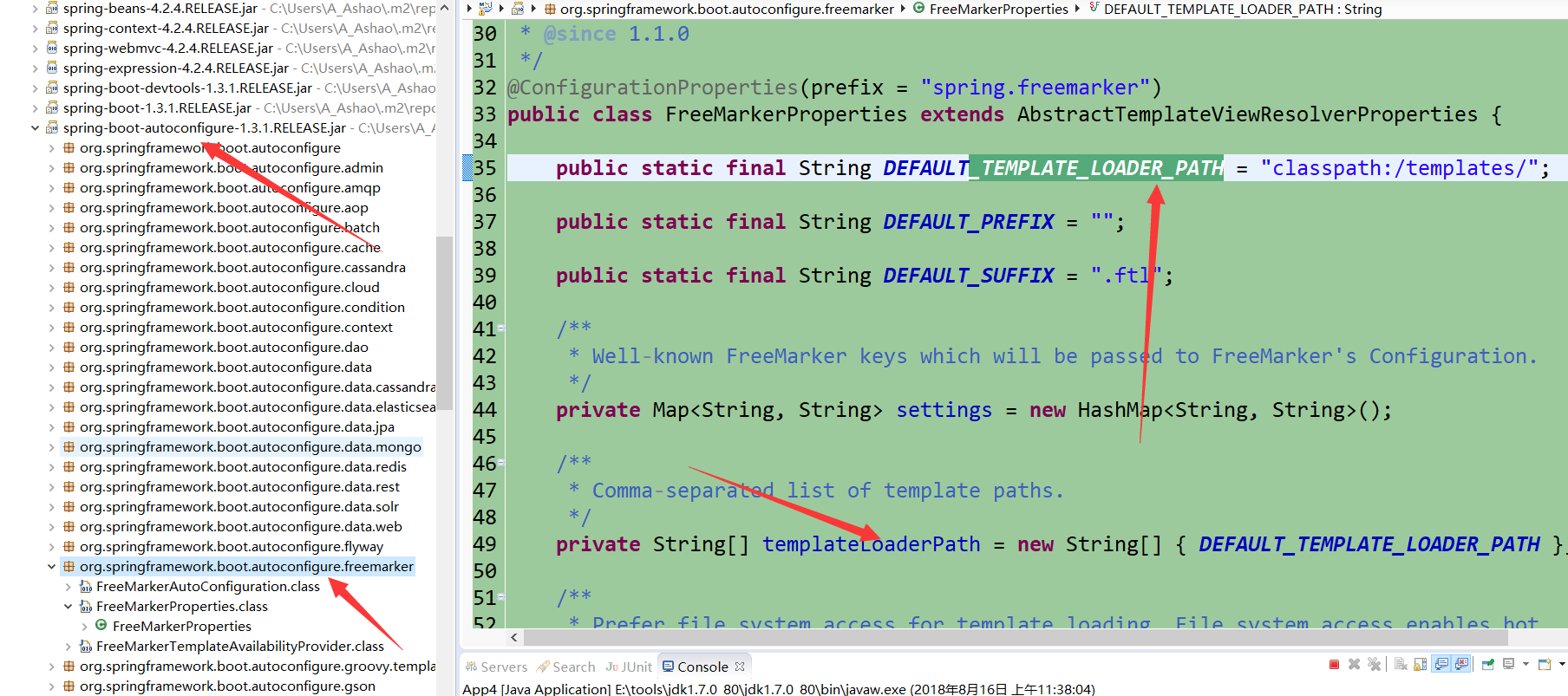
<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-freemarker</artifactId>

</dependency>

二 ：freemarker的默认路径在classpath下的templates文件下，若需要修改必须在application.properties文件中加入spring.freemarker.template-loader-path=....

源代码



1. springboot默认使用tomcat作为容器，若要更换成其他的如jetty,步骤为

一： pom文件中排除tomcat的依赖；

二： pom文件中增加jetty的依赖文件

1. Springboot中使用servlet 和 使用静态资源

一：编写servlet(@webServlet(“”) 继承 httpservlet), filter(@WebFilter实现filter接口)

二：在springboot启动类中使用@ServletComponentScan

静态资源

Classpath路径下新建public/static/resources 目录，即可直接在浏览器中访问；

1. SpringBoot中使用拦截器

一：编写拦截器类 实现 HandlerInterceptor接口；

二：编写配置类 @Configuration

Public Myconfig extends WebMvcConfigurerAdapter{

}

继承WebMvcConfigurerAdapter ,实现addInterceptor接口；

1. Springboot中统一异常处理

编写异常配置类，实现ErrorPageRegisrar接口，实现其中的方法，并纳入spring容器管理之中;

注：取消springboot中默认的异常返回：在启动类中使用exclue=ErrorMvcAutoConfiguration.class;

1. Springboot容器tomcat 配置 (autoConfiguration.web 下 jar包中)

一：配置日志输出位置 server.tomcat.accessLog.directory=h:/temp/logs...

1. Aop

一：引入aop依赖<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-aop</artifactId>

</dependency>

二：使用@aspect定义切边类（）,并纳入spring管理

@PontCut(execution(\* com.ss.controller..\*.\*(..)))

@after @before @round @AfterThrowing 方法中的参数 JoinPoint

三: aop 默认是启动的，可以在application.properties中配置spring.aop.auto=true;

Apring.aop.proxy-target-class=false

false(默认配置，使用jdk的动态代理，实现接口的方式)

True(使用cgLib的方式实现代理，继承方式)

1. Springboot jdbc
2. springboot默认可以支持四种数据源 dbcp，dbcp2，druid;

Pom.xml文件中分别引入spring-jdbc, 链接数据库驱动，想使用的数据源的依赖，之后springboot会自动将DataSources, JdbcTemplate 注入到spring管理的容器中；

1. Application.properties文件中的配置

Spring.datasource.driverClassName=com.mysql.jdbc.Driver

Spring.datasource.url=jdbc:mysql://localhost:3306/test

Spring.datasource.password=root

Spring.datasource.username=root

Spring.datasource.type==org.apache.commons.dbcp2.BasicDataSource（用于指定数据源)

1. 之后可以通过JdbcTemplate进行数据库操作



1. 日志处理

springboot默认使用logback进行日志记录，可以直接将配置好的logback.xml文件放在classpath目录下，则可以直接生效；

Application.properties文件中配置

//配置默认级别， debug,info ,warn,error,off(关闭)

Logging.level.\*=debug (\*可以指 包名表示在当前的包下生效，也可以用ROOT代替，表示所有都生效)

//配置日志文件路径

logging.path=e:/temp/logs

//配置控制台打印格式

logging.pattern.console=...

//配置日志文件中的格式

logging.file.console=...