1. Java 中的注解

@Target说明了Annotation所修饰的对象范围：Annotation可被用于 packages、types（类、接口、枚举、Annotation类型）、类型成员（方法、构造方法、成员变量、枚举值）、方法参数和本地变量（如循环变量、catch参数）。在Annotation类型的声明中使用了target可更加明晰其修饰的目标

1. // 时间表示格式可以改变，yyyyMMdd需要写例如20160523这种形式的时间

SimpleDateFormat sdf = new SimpleDateFormat("yyyy/MM/dd");

String str = "2016/05/23";

// 将字符串的日期转为Date类型，ParsePosition(0)表示从第一个字符开始解析

Date date = sdf.parse(str, new ParsePosition(0));

Calendar calendar = Calendar.getInstance();

calendar.setTime(date);

// add方法中的第二个参数n中，正数表示该日期后n天，负数表示该日期的前n天

calendar.add(Calendar.DATE, -10);

Date date1 = calendar.getTime();

String out = sdf.format(date1);

System.out.println(out);

1. 数据创建

Int[] aa={1,2,3,4,5,6,7};

1. idea Tomcat 部署 war和war exploded的区别

war模式：将WEB工程以包的形式上传到服务器

war exploded模式：将WEB工程以当前文件夹的位置广西上传到服务器

（1）war模式这种可以称之为是发布模式，看名字也知道，这是先打成war包，再发布；  
（2）war exploded模式是直接把文件夹、jsp页面 、classes等等移到Tomcat 部署文件夹里面，进行加载部署。因此这种方式支持热部署，一般在开发的时候也是用这种方式。  
（3）在平时开发的时候，使用热部署的话，应该对Tomcat进行相应的设置，这样的话修改的jsp界面什么的东西才可以及时的显示出来。

5. Log4j的概念

Log4j由三个重要的组成构成：日志记录器(Loggers)，输出端(Appenders)和日志格式化器(Layout)。

1.Logger：控制要启用或禁用哪些日志记录语句，并对日志信息进行级别限制

2.Appenders : 指定了日志将打印到控制台还是文件中

3.Layout : 控制日志信息的显示格式

#Log4J采用类似C语言中的printf函数的打印格式格式化日志信息，打印参数如下：

# %m 输出代码中指定的消息

# %p 输出优先级，即DEBUG,INFO,WARN,ERROR,FATAL

# %r 输出自应用启动到输出该log信息耗费的毫秒数

# %c 输出所属的类目,通常就是所在类的全名

# %t 输出产生该日志事件的线程名

# %n 输出一个回车换行符，Windows平台为“\r\n”，Unix平台为“\n”

# %d 输出日志时间点的日期或时间，默认格式为ISO8601，也可以在其后指定格式

# 如：%d{yyyy年MM月dd日 HH:mm:ss,SSS}，输出类似：2012年01月05日 22:10:28,921

# %l 输出日志事件的发生位置，包括类目名、发生的线程，以及在代码中的行数

# 如：Testlog.main(TestLog.java:10)

# %F 输出日志消息产生时所在的文件名称

# %L 输出代码中的行号

# %x 输出和当前线程相关联的NDC(嵌套诊断环境),像java servlets多客户多线程的应用中

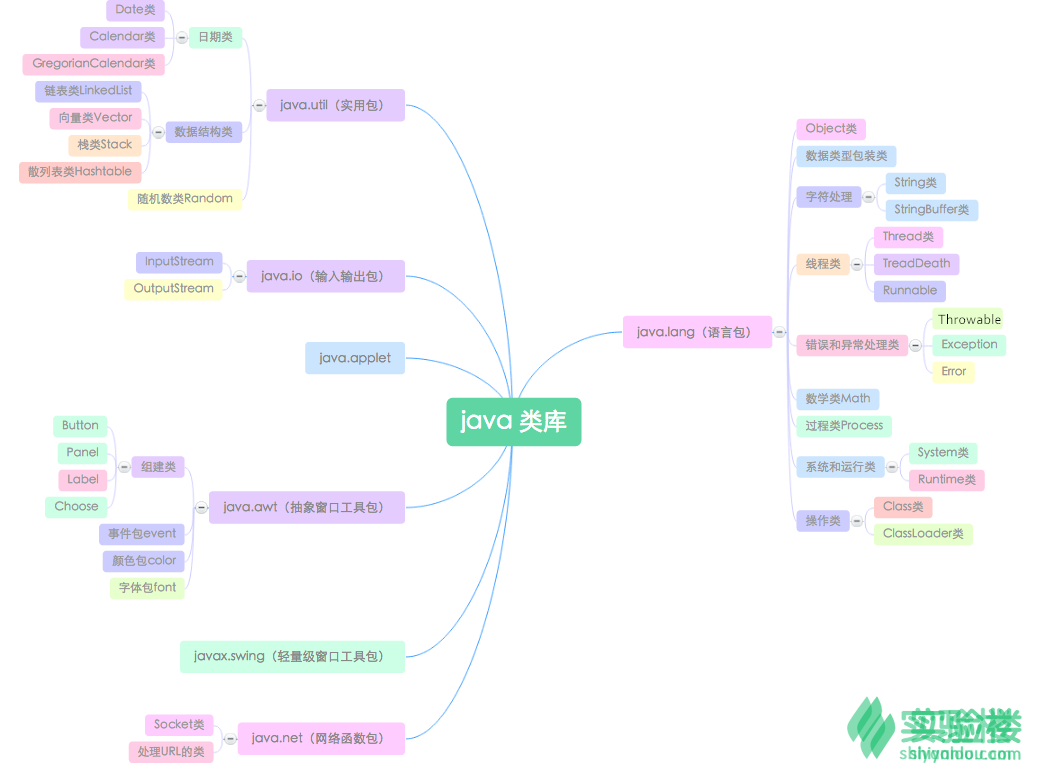
# %% 输出一个"%"字符

1. jvm ：java虚拟机

jre： 运行环境

jdk： 开发工具包

三者的关系

1. java API
2. 
3. 数据类型 byte 8位整数，short 16位，int 32位，long 64位，float 32位浮点数double64位浮点数，boolean，char 单一的16位Unicode字符
4. **synchronized 修饰符 同一时间只能被一个线程访问，**transient 修饰符 序列化的对象包含被transient修饰的实例变量时，java虚拟机跳过该特定的变量。Volatile 修饰符的成员变量在每次被线程访问时，都强制从共享内存中重新读取该成员变量的值。而且，当成员变量发生变化时，会强制县城将变化值回写到共享内存。

## JSP 处理

以下步骤表明了 Web 服务器是如何使用JSP来创建网页的：

* 就像其他普通的网页一样，您的浏览器发送一个 HTTP 请求给服务器。
* Web 服务器识别出这是一个对 JSP 网页的请求，并且将该请求传递给 JSP 引擎。通过使用 URL或者 .jsp 文件来完成。
* JSP 引擎从磁盘中载入 JSP 文件，然后将它们转化为 Servlet。这种转化只是简单地将所有模板文本改用 println() 语句，并且将所有的 JSP 元素转化成 Java 代码。
* JSP 引擎将 Servlet 编译成可执行类，并且将原始请求传递给 Servlet 引擎。
* Web 服务器的某组件将会调用 Servlet 引擎，然后载入并执行 Servlet 类。在执行过程中，Servlet 产生 HTML 格式的输出并将其内嵌于 HTTP response 中上交给 Web 服务器。
* Web 服务器以静态 HTML 网页的形式将 HTTP response 返回到您的浏览器中。
* 最终，Web 浏览器处理 HTTP response 中动态产生的HTML网页，就好像在处理静态网页一样。

# JSP 语法

<%代码片段%>

<jsp:scriptlet>代码片段</jsp:scriptlet>

中文乱码：

<%@ page language="java" contentType="text/html; charset=UTF-8"

pageEncoding="UTF-8"%>

## JSP声明

<%! Declaration;[declaration;]+….%>

<jsp:declaration>代码片段</jsp:declaration>

## JSP表达式

<%=表达式%>

<jsp:expression>表达式</jsp:expression>

## JSP指令

<%@ page %>定义页面的依赖属性，比如脚本语言、error页面、缓存需求等等

<%@ include file=”文件相对url地址”%>包含其他文件

<%@ taglib url=”url” prefix=”prefixOfTag” %> 引入标签库的定义，可以是自定义标签

## JSP行为

JSP行为标签使用XML语法结构来控制servlet引擎。它能够动态插入一个文件，重用JavaBean组件，引导用户去另一个页面，为Java插件产生相关的HTML等等。

行为标签只有一种语法格式，它严格遵守XML标准：

<jsp:action\_name attribute="value" />

行为标签基本上是一些预先就定义好的函数，下表罗列出了一些可用的JSP行为标签：：

|  |  |
| --- | --- |
| **语法** | **描述** |
| jsp:include | 用于在当前页面中包含静态或动态资源 |
| jsp:useBean | 寻找和初始化一个JavaBean组件 |
| jsp:setProperty | 设置 JavaBean组件的值 |
| jsp:getProperty | 将 JavaBean组件的值插入到 output中 |
| jsp:forward | 从一个JSP文件向另一个文件传递一个包含用户请求的request对象 |
| jsp:plugin | 用于在生成的HTML页面中包含Applet和JavaBean对象 |
| jsp:element | 动态创建一个XML元素 |
| jsp:attribute | 定义动态创建的XML元素的属性 |
| jsp:body | 定义动态创建的XML元素的主体 |
| jsp:text | 用于封装模板数据 |

## JSP隐含对象

JSP支持九个自动定义的变量，江湖人称隐含对象。这九个隐含对象的简介见下表：

|  |  |
| --- | --- |
| **对象** | **描述** |
| request | **HttpServletRequest**类的实例 |
| response | **HttpServletResponse**类的实例 |
| out | **PrintWriter**类的实例，用于把结果输出至网页上 |
| session | **HttpSession**类的实例 |
| application | **ServletContext**类的实例，与应用上下文有关 |
| config | **ServletConfig**类的实例 |
| pageContext | **PageContext**类的实例，提供对JSP页面所有对象以及命名空间的访问 |
| page | 类似于Java类中的this关键字 |
| Exception | **Exception**类的对象，代表发生错误的JSP页面中对应的异常对象 |

Include 与jsp:include区别 第一个直接在将文件原封不动的插入到页面中，然后jsp编译器在对这个合成文件进行编译最后编译后的也是一个文件（静态包含）

后一个在执行动作时候会把请求转发到包含页面，执行完输入后在返回页面据徐执行（动态包含）

## JSP 读取表单数据

* **getParameter():** 使用 request.getParameter() 方法来获取表单参数的值。
* **getParameterValues():** 获得如checkbox类（名字相同，但值有多个）的数据。 接收数组变量 ，如checkbox类型
* **getParameterNames():**该方法可以取得所有变量的名称，该方法返回一个 Enumeration。
* **getInputStream():**调用此方法来读取来自客户端的二进制数据流。

JSP过滤器

创建一个过滤器类,实现Filter接口

web.xml中添加配置

<filter>

<filter-name>LogFilter</filter-name>

<filter-class>com.runoob.test.LogFilter</filter-class>

<init-param>

<param-name>Site</param-name>

<param-value>菜鸟教程</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>LogFilter</filter-name>

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

</filter-mapping>

# JSP 日期处理

获取Date对象后，您就能够使用下表列出的所有方法：

|  |  |
| --- | --- |
| **序号** | **方法 & 描述** |
| 1 | **boolean after(Date date)**  如果比给定的日期晚，则返回true，否则返回false |
| 2 | **boolean before(Date date)**  如果比给定的日期早，则返回true，否则返回false |
| 3 | **Object clone( )**  获取当前对象的一个副本 |
| 4 | **int compareTo(Date date)**  如果与给定日期相等，则返回0，如果比给定日期早，则返回一个负数，如果比给定日期晚，则返回一个正数 |
| 5 | **int compareTo(Object obj)**  与 compareTo(Date) 方法相同，如果 obj 不是Date类或其子类的对象，抛出ClassCastException异常 |
| 6 | **boolean equals(Object date)**  如果与给定日期相同，则返回true，否则返回false |
| 7 | **long getTime( )**  返回从1970年1月1日凌晨至此对象所表示时间的毫秒数 |
| 8 | **int hashCode( )**  返回此对象的哈希码 |
| 9 | **void setTime(long time)**  使用给定参数设置时间和日期，参数time表示从1970年1月1日凌晨至time所经过的毫秒数 |
| 10 | **String toString( )**  将此对象转换为字符串并返回这个字符串 |

## 日期比较

就像我在开头所提到的，您可以在JSP脚本中使用任何Java方法。如果您想要比较两个日期，

可以参照下面的方法来做：

* 使用getTime()方法得到毫秒数，然后比较毫秒数就行了。
* 使用before()，after()，equals()方法。比如，new Date(99,2,12).before(new Date(99,2,18))返回true。
* 使用compareTo()方法，这个方法在Comparable接口中定义，在Date中实现。

# JSP 标准标签库（JSTL）

JSP标准标签库（JSTL）是一个JSP标签集合，它封装了JSP应用的通用核心功能。

JSTL支持通用的、结构化的任务，比如迭代，条件判断，XML文档操作，国际化标签，SQL标签。 除了这些，它还提供了一个框架来使用集成JSTL的自定义标签。

根据JSTL标签所提供的功能，可以将其分为5个类别。

核心标签

格式化标签

SQL 标签

XML 标签

JSTL 函数

|  |  |
| --- | --- |
| **标签** | **描述** |
| [<c:out>](http://www.runoob.com/jsp/jstl-core-out-tag.html) | 用于在JSP中显示数据，就像<%= ... > |
| [<c:set>](http://www.runoob.com/jsp/jstl-core-set-tag.html) | 用于保存数据 |
| [<c:remove>](http://www.runoob.com/jsp/jstl-core-remove-tag.html) | 用于删除数据 |
| [<c:catch>](http://www.runoob.com/jsp/jstl-core-catch-tag.html) | 用来处理产生错误的异常状况，并且将错误信息储存起来 |
| [<c:if>](http://www.runoob.com/jsp/jstl-core-if-tag.html) | 与我们在一般程序中用的if一样 |
| [<c:choose>](http://www.runoob.com/jsp/jstl-core-choose-tag.html) | 本身只当做<c:when>和<c:otherwise>的父标签 |
| [<c:when>](http://www.runoob.com/jsp/jstl-core-choose-tag.html) | <c:choose>的子标签，用来判断条件是否成立 |
| [<c:otherwise>](http://www.runoob.com/jsp/jstl-core-choose-tag.html) | <c:choose>的子标签，接在<c:when>标签后，当<c:when>标签判断为false时被执行 |
| [<c:import>](http://www.runoob.com/jsp/jstl-core-import-tag.html) | 检索一个绝对或相对 URL，然后将其内容暴露给页面 |
| [<c:forEach>](http://www.runoob.com/jsp/jstl-core-foreach-tag.html) | 基础迭代标签，接受多种集合类型 |
| [<c:forTokens>](http://www.runoob.com/jsp/jstl-core-foreach-tag.html) | 根据指定的分隔符来分隔内容并迭代输出 |
| [<c:param>](http://www.runoob.com/jsp/jstl-core-param-tag.html) | 用来给包含或重定向的页面传递参数 |
| [<c:redirect>](http://www.runoob.com/jsp/jstl-core-redirect-tag.html) | 重定向至一个新的URL. |
| [<c:url>](http://www.runoob.com/jsp/jstl-core-url-tag.html) | 使用可选的查询参数来创造一个URL |
| 标签 | 描述 |
| [<fmt:formatNumber>](http://www.runoob.com/jsp/jstl-format-formatnumber-tag.html) | 使用指定的格式或精度格式化数字 |
| [<fmt:parseNumber>](http://www.runoob.com/jsp/jstl-format-parsenumber-tag.html) | 解析一个代表着数字，货币或百分比的字符串 |
| [<fmt:formatDate>](http://www.runoob.com/jsp/jstl-format-formatdate-tag.html) | 使用指定的风格或模式格式化日期和时间 |
| [<fmt:parseDate>](http://www.runoob.com/jsp/jstl-format-parsedate-tag.html) | 解析一个代表着日期或时间的字符串 |
| [<fmt:bundle>](http://www.runoob.com/jsp/jstl-format-bundle-tag.html) | 绑定资源 |
| [<fmt:setLocale>](http://www.runoob.com/jsp/jstl-format-setlocale-tag.html) | 指定地区 |
| [<fmt:setBundle>](http://www.runoob.com/jsp/jstl-format-setbundle-tag.html) | 绑定资源 |
| [<fmt:timeZone>](http://www.runoob.com/jsp/jstl-format-timezone-tag.html) | 指定时区 |
| [<fmt:setTimeZone>](http://www.runoob.com/jsp/jstl-format-settimezone-tag.html) | 指定时区 |
| [<fmt:message>](http://www.runoob.com/jsp/jstl-format-message-tag.html) | 显示资源配置文件信息 |
| [<fmt:requestEncoding>](http://www.runoob.com/jsp/jstl-format-requestencoding-tag.html) | 设置request的字符编码 |

## JSTL函数

JSTL包含一系列标准函数，大部分是通用的字符串处理函数。引用JSTL函数库的语法如下：

<%@ taglib prefix="fn"

uri="http://java.sun.com/jsp/jstl/functions" %>

|  |  |
| --- | --- |
| **函数** | **描述** |
| [fn:contains()](http://www.runoob.com/jsp/jstl-function-contains.html) | 测试输入的字符串是否包含指定的子串 |
| [fn:containsIgnoreCase()](http://www.runoob.com/jsp/jstl-function-containsignoreCase.html) | 测试输入的字符串是否包含指定的子串，大小写不敏感 |
| [fn:endsWith()](http://www.runoob.com/jsp/jstl-function-endswith.html) | 测试输入的字符串是否以指定的后缀结尾 |
| [fn:escapeXml()](http://www.runoob.com/jsp/jstl-function-escapexml.html) | 跳过可以作为XML标记的字符 |
| [fn:indexOf()](http://www.runoob.com/jsp/jstl-function-indexof.html) | 返回指定字符串在输入字符串中出现的位置 |
| [fn:join()](http://www.runoob.com/jsp/jstl-function-join.html) | 将数组中的元素合成一个字符串然后输出 |
| [fn:length()](http://www.runoob.com/jsp/jstl-function-length.html) | 返回字符串长度 |
| [fn:replace()](http://www.runoob.com/jsp/jstl-function-replace.html) | 将输入字符串中指定的位置替换为指定的字符串然后返回 |
| [fn:split()](http://www.runoob.com/jsp/jstl-function-split.html) | 将字符串用指定的分隔符分隔然后组成一个子字符串数组并返回 |
| [fn:startsWith()](http://www.runoob.com/jsp/jstl-function-startswith.html) | 测试输入字符串是否以指定的前缀开始 |
| [fn:substring()](http://www.runoob.com/jsp/jstl-function-substring.html) | 返回字符串的子集 |
| [fn:substringAfter()](http://www.runoob.com/jsp/jstl-function-substringafter.html) | 返回字符串在指定子串之后的子集 |
| [fn:substringBefore()](http://www.runoob.com/jsp/jstl-function-substringbefore.html) | 返回字符串在指定子串之前的子集 |
| [fn:toLowerCase()](http://www.runoob.com/jsp/jstl-function-tolowercase.html) | 将字符串中的字符转为小写 |
| [fn:toUpperCase()](http://www.runoob.com/jsp/jstl-function-touppercase.html) | 将字符串中的字符转为大写 |
| [fn:trim()](http://www.runoob.com/jsp/jstl-function-trim.html) | 移除首位的空白符 |

# JSP 表达式语言

<jsp:setProperty name=”box” property=”perimeter” value=”100”/>

${expr}

 StringBuffer 之间的最大不同在于 StringBuilder 的方法不是线程安全的

Java休眠（sleep）Thread.sleep(1000\*3)

计算时间间隔

public static void main(String args[]) {

try {

long start = System.currentTimeMillis( );

System.out.println(new Date( ) + "\n");

Thread.sleep(5\*60\*10);

System.out.println(new Date( ) + "\n");

long end = System.currentTimeMillis( );

long diff = end - start;

System.out.println("Difference is : " + diff);

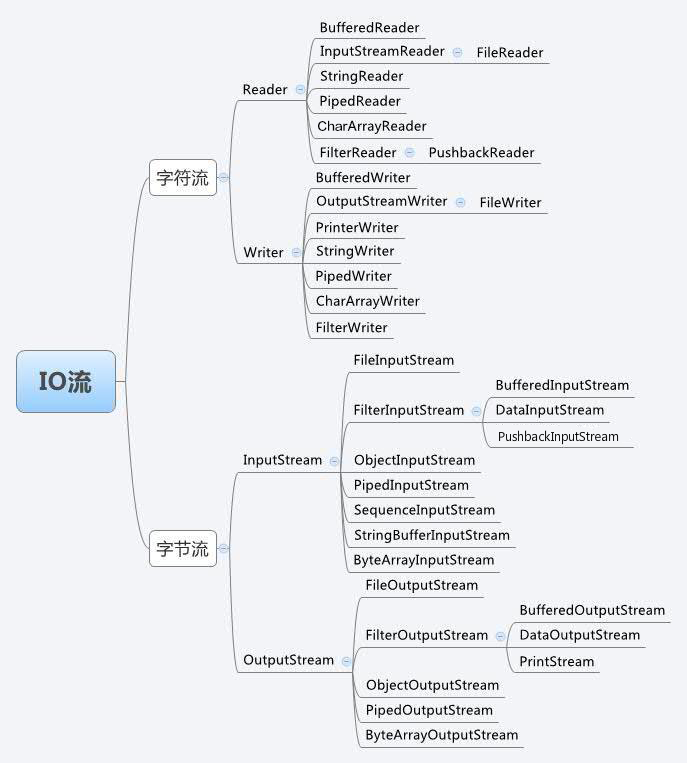
}

catch (Exception e) {

System.out.println("Got an exception!");

}

}



没有任何方法的接口是标记接口

用来标识对象，使对象拥有某些特权

集合框架包括两种容器，集合和图

Collection有三种子类型List,Set,Quque 实现有ArrayList ,Linked, List, HashSet,LinkedHashSet，HashMap,LinkedHashMap

compareTo() 方法的实现思路：两个字符数组依次从前开始比较，如果对象位置出现字符不同则返回两个字符的编码之差，后面的字符不再比较；如果两个字符数组的长度不一样，并且较短的数组和较长数组所有对应位置的字符都相同，则返回两个数组的长度之差。

Split() 中如果用过个字符切割 “a,b-c”.split(“[,-]”)

 Collections 类的 Collections.max() 和 Collections.min() 方法来查找数组中的最大和最小值

数组与集合之间的转换

Integer[] numbers={1,2,3,4,5};

List<Integet> list=Arrays.asList(numbers);

集合转数组第一种

List<Integer> list=new ArrayList<>();

Integer[] dd=**new** Integer[cc.size()];  
cc.toArray(dd);

集合转数组第二种

Object[] dd=cc.toArray();  
Integer[] ee=(Integer[]) dd;