常见问题

## 使用JSONObject需要的JAR包

1. commons-logging-1.0.4.jar
2. commons-lang-2.3.jar
3. commons-collections-3.2.jar
4. commons-beanutils-1.7.0.jar
5. json-lib-2.2.1-jdk15.jar
6. ezmorph-1.0.4.jar

## 2、将ajax接收的数据显示到JSP页面需要的处理

使用eval()这个方法，可以将回调函数中的数据遍历成数组，然后便可得到数据即可显示到页面

## 3、出现Could not open JDBC Connection for transaction; nested exception is java.sql.SQLException: Connections could not be acquired from the underlying database!

**以上问题可能有一下原因造成：**

**1，驱动配置有误：driver=com.mysql.jdbc.Driver  
2，数据库连接地址有误：url=jdbc:mysql://localhost:3306/test?3useUnicode=true&characterEncoding=utf8  
3，密码或帐号有误：username=root  
      password=root**

**4，数据库未启动或无权访问**

**5，项目未引入对应的驱动jar包mysql-connector-java-5.1.6-bin.jar**

**6,**mysql root没有远程访问的权限，需要增加权限，增加权限的步骤如下：  
进入mysql数据库：  
grant all privileges on \*.\* to 'root'@'%' identified by 'root' with grant option;  
flush privileges;

之前出现问题的是第六个

## 4、接收ajax的请求

可以使用request来接收参数

## 5、乱码问题

<filter>

<filter-name>CharacterEncodingFilter</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>CharacterEncodingFilter</filter-name>

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

</filter-mapping>

Post请求出现乱码可以使用上面的这个方法

自定义中文过滤器，实现Filter接口即可

reuqest.setCharacterEncoding("UTF-8");

response.setContentType("text/html;charset=UTF-8");

对于get请求中文参数出现乱码解决方法有两个：

修改tomcat配置文件添加编码与工程编码一致，如下：

<Connector URIEncoding="utf-8" connectionTimeout="20000" port="8080" protocol="HTTP/1.1" redirectPort="8443"/>

另外一种方法对参数进行重新编码：

String userName new

String(request.getParamter("userName").getBytes("ISO8859-1"),"utf-8")

ISO8859-1是tomcat默认编码，需要将tomcat编码后的内容按utf-8编码

## 6、Eclipse使用jstl

1、需要jstl.jar jstl-standard.jar 两个jar包，或者使用taglibs-standard-impl.jar

2、需要在META-INF 中加入三个 .tld 文件



## InputStream类相关

InputStream.read()方法是一个一个字节的读取，它返回的数字是0-255的字节所对应的数字，流读取完之后返回的是 “--1” ，这样就可以判断流是否读完，int read(byte []) 是将流读取的字节全部保存在字节数组中，然后可以用String(byte[]) 将字节转换成String字符串。

BufferedInputStream 则是将读取的数据保存在缓冲区中，BufferedInputStream类中有定义一个缓冲数组 ，先将读取的数据放在缓冲区中，当缓冲区满的时候再将它读取出来。

当我们打开或者是关闭流的时候都会出现异常，这个时候会抛出异常，程序结束，后面的代码便不会执行，也就不会执行close（）方法，这个时候需要使用try,catch,finally

使用缓冲(提高效率),并循环读取(读完所有内容).

总结：读完文件的所有内容。很显然可以使用普通的read方法，一次读一个字节直到读到文件末尾。为了提高效率可以使用read(byte[] byt);方法就是所谓的使用缓冲提高效率。我们可以读取大文本数据测试(大于1K的文本文件.)

|  |
| --- |
| /\*\*  \* 使用字节数组当缓冲  \* \*/  **private** **static** **void** showContent5(String path) **throws** IOException {  FileInputStream fis = **new** FileInputStream(path);  **byte**[] byt = **new** **byte**[1024];  **int** len = fis.read(byt);  System.*out*.println(len);  String buffer = **new** String(byt, 0, len);  System.*out*.print(buffer);  } |

注意：如何将字节数组转成字符串？ 可以通过创建字符串对象即可。

发现:一旦数据超过1024个字节，数组就存储不下。

如何将文件的剩余内容读完？

我们可以通过通过循环保证文件读取完。

|  |
| --- |
| /\*\*  \* 使用字节数组当缓冲  \* \*/  **private** **static** **void** showContent7(String path) **throws** IOException {  FileInputStream fis = **new** FileInputStream(path);  **byte**[] byt = **new** **byte**[1024];  **int** len = 0;  **while** ((len = fis.read(byt)) != -1) {  System.*out*.println(**new** String(byt, 0, len));  }  } |

## 关于char

每个字符背后都对应一个ASCII码，比如 ’ A ’ 对应的ASCII码就是65，因此将char值赋给int型就是赋值一个ASCII码值

## HashMap实现原理分析

数组：数组的存储空间是连续的，占用内存大，故空间复杂度也很大，但是数组的二分查找的复杂度小，是O（1），数组的特点是是：寻址容易(即查询容易)，插入删除困难

链表：链表的存储空间是散列的，占用内存空间比较宽松，故空间复杂度小，但时间复杂度大，为O（N），链表的特点是：寻址困难，但是插入和删除简单。

HashMap的存取实现：用的是线性数组

大致过程：

存储：

Int hash = key . hashCode();

Int index = hash % Entry[ ] . length;

Entry [ index ] = value;

取值：

Int hash = key . hashCode();

Int index = hash % Entry [ ].length;

Return Entry[ index ];

疑问：如果hash % Entry[ ].length 得到的index的值相同，会不会产生覆盖？

解答：不会，HashMap 中有一个静态类Node[ key ,value, next ] (其实是实现了Map中的Entry静态类), map 中的值就是就是保存到Node 中的，HashMap 使用的是链式数据结构，Node中的next就是指向下一个Node;

再散列rehash过程

当哈希表的容量超过默认的容量的是时候，必须调整table的大小。当哈希表的容量达到最大可能值的时候，那么该方法就将容量调整到Integer.MAX\_VALUE,返回，重新创建一张表，并将原数据映射到新表中

**void** resize(**int** newCapacity) {

        Entry[] oldTable = table;

**int** oldCapacity = oldTable.length;

**if** (oldCapacity == *MAXIMUM\_CAPACITY*) {

            threshold = Integer.*MAX\_VALUE*;

**return**;

        }

        Entry[] newTable = **new** Entry[newCapacity];

        transfer(newTable);

        table = newTable;

        threshold = (**int**)(newCapacity \* loadFactor);

    }

    /\*\*

     \* Transfers all entries from current table to newTable.

     \*/

**void** transfer(Entry[] newTable) {

        Entry[] src = table;

**int** newCapacity = newTable.length;

**for** (**int** j = 0; j < src.length; j++) {

            Entry<K,V> e = src[j];

**if** (e != **null**) {

                src[j] = **null**;

**do** {

                    Entry<K,V> next = e.next;

                    //重新计算index

**int** i = *indexFor*(e.hash, newCapacity);

                    e.next = newTable[i];

                    newTable[i] = e;

                    e = next;

                } **while** (e != **null**);

            }

        }

    }

解决hash冲突的方法：

1. 开放定址法（线性探测再散列，二次探测再散列，伪随机探测再散列）
2. 再哈希法
3. **链地址法**
4. 建立一个公共溢出区

## HashMap和HashTable之间的区别？

HashTable是一个线程安全的API，它的方法是使用了synchronized 进行了修饰。尽管不推荐使用HashTable开发一个高性能的应用，但是它确实能够保证应用中的线程的安全。相反，HashMap不能够保证线程的安全，因此要开发一个高性能的应用可以使用ConcurrentHashMap来进行开发。HashMap在单线程中比HashTable有更高的性能，这得益于HashMap使用了多种方式避免哈希碰撞，其中辅助hash函数就是著名的例子。

HashTable和HashMap都是基础的关联数组，哈希指的是一种通过Key来获取数据的过程(映射)

## ArrayList和LinkedList之间的区别

在数据结构方面：

ArrayList使用的是动态数组，LinkedList使用的是双向链表，链表方便插入和删除，因此LinkedList的插入和删除更为快捷；数组方便查询与遍历，因此ArrayList更倾向于查询操作。所以使用List，首选ArrayList，如果插入和删除操作频繁，可考虑LinkedList

## SequenceInputStream序列流

序列流，可以实现多个流进行合并

构造函数

构造函数

SequenceInputStream(InputStream s1, InputStream s2)

SequenceInputStream(InputStream s1, InputStream s2)

## 对象的序列化

当创建对象的时候，程序正在运行，但是当程序结束的时候，对象也就随之销毁了。若想让程序不运行仍能够存在并保存其状态信息，可以将其序列化保存在文件设备中

对象的序列化：将内存中的对象直接写入到文件设备中

对象的反序列化：将文件设备中持久化设备转换为内存对象

|  |
| --- |
| ObjectOutput  writeObject(Object obj) 序列化对象  将对象写入到底层存储或流  ObjectInput  readObject() 反序列化对象  读取并返回对象 |

上述两个都是接口，所以要想使用可以使用它们的具体实现类

ObjectInputStream ObjectOutputStream

在使用ObjectOutputStream写入对象的时候，该对象需要实现Serializable接口，否者会抛异常NotSerializableException

简单的来说，实现序列化的对象可以通过ObjectOutputStream将其写入到文件中去，然后通过ObjectInputStream读出来

## 15、Cookie和Session

Cookie是客户端技术，程序把每个用户的数据以Cookie的形式写到用户各自的浏览器中，当用户再次用浏览器访问WEB资源的时候就会带着各自的数据，这样WEB处理的就是用户各自的数据了

Session是服务器端技术，利用这个技术，服务器可以为每个用户的浏览器创建一个独享的Session对象，用户请求的WEB资源也可以将其放在各自的session中

Cookie的分类：会话级别的cookie，浏览器关闭就失效；持久化级别的cookie，写到本地文件中去（需要设置有效的保存时间）

所以通过cookie可以判断用户是否是第一次访问，显示上次浏览过的商品信息。通过request.setCookie（），一个Cookie只能识别一种信息，name,value,浏览器一般可以存放300个cookie，每个web站点可以存放20个cookie，每个cookie大小限制为4K

Session的实现原理：基于cookie，利用cookie回写一个JSESSIONID(JSESSIONID就是为每个session起的唯一标识)

禁用Cookie后servlet共享数据导致的问题

解决方案:可以使用URL重写（就是把session id附加在URL路径的后面，附加的方式也有两种，一种是作为URL路径的附加信息，另一种是作为查询字符串附加在URL后面。网络在整个交互过程中始终保持状态，就必须在每个客户端可能请求的路径后面都包含这个session id。）

response. **encodeRedirectURL**(java.lang.String url)

用于对sendRedirect方法后的url地址进行重写。

response. encodeURL(java.lang.String url)

用于对表单action和超链接的url地址进行重写

## Http协议

客户端连接上服务器后，要想获得某个WEB资源就必须遵守一定的通讯规范，HTTP协议就是用于定义客户端和WEB服务器之间通讯的规范。

HTTP1.0和HTTP1.1之间的区别：

HTTP1.0：链接之后马上断开

HTTP1.1：链接之后不会马上断开，还可以继续操作，一段时间不操作后，会自动断开

客户端向服务器发送HTTP请求

* **举例：**

GET /books/java.html HTTP/1.1----------------------------------请求行

Accept: \*/\* ------------------ MIME类型

Accept-Language: en-us ------------------ 语言

Connection: Keep-Alive ------------------

Host: localhost ------------------------------- 多个消息头

Referer: http://localhost/links.asp------------------

User-Agent: Mozilla/4.0 ------------------

Accept-Encoding: gzip, deflate------------------

请求行用于描述客户端的请

求方式、请求的资源名称，

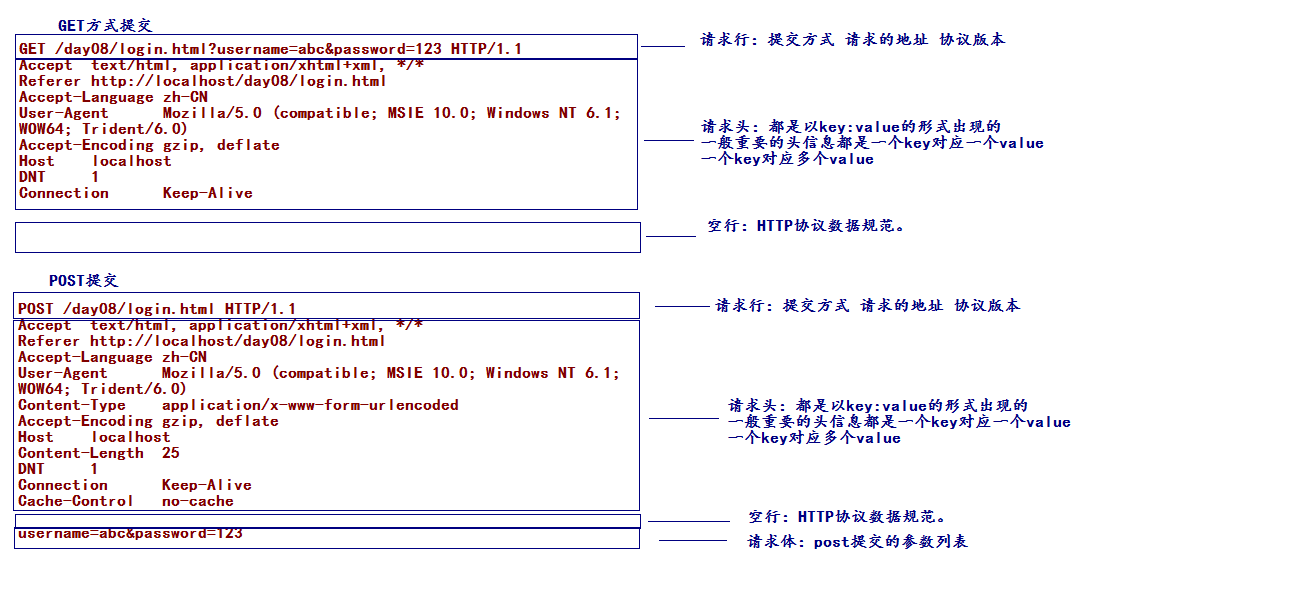
以及使用的HTTP协议版本号

消息头用于描述客户端请求

哪台主机，以及

客户端的一些环境信息等

* **用于HTTP请求中的常用头**
* **Accept: text/html,image/\* 客户端识别文件类型**
* **Accept-Charset: ISO-8859-1 客户端字符集**
* **Accept-Encoding: gzip 客户端支持编码类型 gzip 压缩格式**
* **Accept-Language:zh-cn 客户端语言**
* **Host: www.itcast.com:80 访问服务器地址**
* **If-Modified-Since: Tue, 11 Jul 2000 18:23:51 GMT （重要） 和Last-Modified 一起使用 控制缓存**
* **Referer: http://www.itcast.com/index.jsp （重要）代表当前网页的来源 防止盗链**
* **User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0) 浏览器类型（文件下载时，不同浏览器进行不同处理）**
* **Connection: close/Keep-Alive 请求后连接是关闭还是保持（**使客户端到服务器端的连接持续有效，当出现对服务器的后继请求时，Keep-Alive功能避免了建立或者重新建立连接。**）**
* **Date: Tue, 11 Jul 2000 18:23:51 GMT 请求时间**



GET和POST区别？

1) 区分哪些是GET请求？ 哪些POST请求？

GET：在浏览器直接输入URL、<a href=""> 、<form method="get" >

POST： <form method="post" >

2) GET请求数据位于请求行中 ，POST请求数据位于请求体中

GET /day4/form.html?username=zhangsan HTTP/1.1

POST /day4/form.html HTTP/1.1

...

username=lisi

3) GET请求数据在URL上显示，所有有长度限制，通常是1kb

Http中200、302、304、404和500等响应状态码所表示的意义？

200:访问成功（表示一切正常，返回的是正常请求结果）

302：临时重定向（指出被请求的文档已被临时移动到别处，此文档的新的URL在Location响应头中给出）

304：未修改（表示客户机缓存的版本是最新的，客户机应该继续使用它。）

404：访问的文件不存在（服务器上不存在客户机所请求的资源）

500：内部服务器错误（服务器端的CGI、ASP、JSP等程序发生错误）

## md5的加盐处理

如果两个人的密码相同，那么经过md5加密后的结果就是相同的，如果一个人可以查看数据库发现别人的md5加密结果和自己的是相同的，那么他就可以使用别人的账号来登录。为此

// 对密码进行加盐后加密，加密后再通过Hibernate往数据库里存

String = changedPswd=DigestUtils.md5Hex(name+pswd);

上面代码中盐就是用户名，可以的话还可以用用户注册时的邮件，注册时间等非空信息（如果是空信息这个加盐处理会失效）。

JDK里面有一个java.security.MessageDigest类，这个类就是用来加密的。

## 并行和并发的区别？

并行就是两个或两个以上的任务同时进行，需要多个多核CPU；

并发就是两个任务都进行请求，但是处理器只能接受一个任务请求，所以就安排两个任务轮流进行，由于时间间隔较短，所以感觉像是在同时进行。

好比说我同时和两个人聊天，左手操作电脑和甲聊天，右手操作电脑和乙聊天，这叫做并行；

比如说有一台电脑，我现在立刻和甲聊天，发完消息后立即和乙聊天发送消息。

## 创建两种多线程的区别？

查看源码的区别：

继承Thread：由于子类继承了Thread重写了run（）方法，当调用start方法的时候会直接调用子类的run（）方法（具体怎么调用run的我们看不到，因为是由虚拟机来进行调用的）

实现Runnable:构造函数中传入了Runnable的引用，并将其赋给了它的成员变量， start方法调用run（）方法进行判断Runnable的引用是否为空，如果不为空则直接调用的是继承Runnable的子类的run方法

继承Thread：

A: 可以直接调用Thread中的方法，代码简单

B: 如果已经有一个父类就不能够继承Thread，因为Java是单继承的

实现Runnable：

A: 可以有父类，只需要实现接口就行了

B: 不能够直接使用Thread中的方法，首先需要获得线程的对象才能够Thread中的方法，代

码复杂

匿名内部类：  
new Thread(){

public void run(){

//重写run（）方法

}

}.start();

new Thread（new Runnable（）{

public void run(){

重写run（）方法

}

}).start();

匿名内部类使用局部变量需要在局部变量前加上final

注意：在同步Synchronized中的锁可以是任意对象，但是不能够使用匿名对象，就是new Demo（）

非静态的同步方法锁对象是this，静态方法的同步锁对象是该类的字节码对象，即Demo.class 要记住类的字节码对象是唯一的所以绝对不会出错

如果用引用数据类型成员变量当做锁对象，必须是静态的（总结一下就是锁对象必须是唯一的）

为了保证线程安全，尽量不要嵌套syncronized同步代码块，防止发生死锁

在同步代码块中用哪个对象锁，就用哪个对象的wait()

为什么wait和notify方法会在Object类中？

因为锁对象是任意的，Object类是所有类的基类，所以需要定义在Object类中

请说出sleep和wait之间的区别？

A:sleep传入一个参数，时间到了自动醒来

Wait可以传一个参数，也可以不传入参数，若传入则表示在传入的时间后开始等待，不传入则直接等待

B:sleep在同步函数和同步代码块中，不释放锁（抱着锁睡）

Wait在同步函数和同步代码块中，释放锁

If语句是在哪里等待就会在哪里起来（不用再去判断条件了）

While语句是会重新进去循环进行判断条件

## Buffer的基本使用

使用Buffer的四个基本步骤：

1. 写入数据到Buffer;
2. 调用flip方法；
3. 从Buffer中读取数据
4. 调用chear方法或者compact方法。

limit , position , capacity

capacity：作为一个内存块，Buffer有一个固定的大小值，也叫“capacity”.你只能往里写capacity个byte、long，char等类型。一旦Buffer满了，需要将其清空（通过读数据或者清除数据）才能继续写数据往里写数据。

position：当你写数据到Buffer中时，position表示当前的位置。初始的position值为0.当一个byte、long等数据写到Buffer后， position会向前移动到下一个可插入数据的Buffer单元。position最大可为capacity – 1.

当读取数据时，也是从某个特定位置读。当将Buffer从写模式切换到读模式，position会被重置为0. 当从Buffer的position处读取数据时，position向前移动到下一个可读的位置。

Limit：在写模式下，Buffer的limit表示你最多能往Buffer里写多少数据。 写模式下，limit等于Buffer的capacity。

当切换Buffer到读模式时， limit表示你最多能读到多少数据。因此，当切换Buffer到读模式时，limit会被设置成写模式下的position值。换句话说，你能读到之前写入的所有数据（limit被设置成已写数据的数量，这个值在写模式下就是position）

Clear方法

http://www.blogjava.net/Images/OutliningIndicators/None.gifpublic final Buffer clear()  
http://www.blogjava.net/Images/OutliningIndicators/ExpandedBlockStart.gif{  
http://www.blogjava.net/Images/OutliningIndicators/InBlock.gif    position = 0; //重置当前读写位置  
http://www.blogjava.net/Images/OutliningIndicators/InBlock.gif    limit = capacity;   
http://www.blogjava.net/Images/OutliningIndicators/InBlock.gif    mark = -1;  //取消标记  
http://www.blogjava.net/Images/OutliningIndicators/InBlock.gif    return this;  
http://www.blogjava.net/Images/OutliningIndicators/ExpandedBlockEnd.gif}  
http://www.blogjava.net/Images/OutliningIndicators/None.gifflip方法

http://www.blogjava.net/Images/OutliningIndicators/ExpandedBlockStart.gifpublic final Buffer flip() {  
http://www.blogjava.net/Images/OutliningIndicators/InBlock.gif limit = position;  
http://www.blogjava.net/Images/OutliningIndicators/InBlock.gif position = 0;  
http://www.blogjava.net/Images/OutliningIndicators/InBlock.gif mark = -1;  
http://www.blogjava.net/Images/OutliningIndicators/InBlock.gif return this;  
http://www.blogjava.net/Images/OutliningIndicators/ExpandedBlockEnd.gif} rewind方法

1http://www.blogjava.net/Images/OutliningIndicators/ExpandedBlockStart.gifpublic final Buffer rewind() {  
2http://www.blogjava.net/Images/OutliningIndicators/InBlock.gif position = 0;  
3http://www.blogjava.net/Images/OutliningIndicators/InBlock.gif mark = -1;  
4http://www.blogjava.net/Images/OutliningIndicators/InBlock.gif return this;  
5http://www.blogjava.net/Images/OutliningIndicators/ExpandedBlockEnd.gif}

## transient 修饰的变量

transient字是表明域不能序列化

## 使用clone方法的作用？

使用这个方法会对原有实例所有的字段进行复制，也就相当于copy，各自的改动互不影响

注：要想使用这个方法首先需要实现Cloneable的接口，重写clone方法。

## 浅复制和深复制（原型模式）

浅复制：就是把对象中的所有变量都进行复制，但是原对象中的引用对象不复制，依旧是指向，即原对象和复制后的对象都指向那个引用对象的地址，没有进行完全的复制（复制；即相当于Windows下的copy，各自更改互不影响），实现Cloneable接口重写clone方法即可。（其实可以任意定义方法名例如cloneA，重点是super.clone（），调用的是Object中的clone方法，又是native本地方法）

深复制；就是在浅复制的基础上将原对象中的引用对象也进行了复制。

常用的方法：原对象实现Serializable接口，然后将对象写入流中，再从流中读出来

1. /\* 深复制 \*/
2. **public** Object deepClone() **throws** IOException, ClassNotFoundException {
4. /\* 写入当前对象的二进制流 \*/
5. ByteArrayOutputStream bos = **new** ByteArrayOutputStream();
6. ObjectOutputStream oos = **new** ObjectOutputStream(bos);
7. oos.writeObject(**this**);
9. /\* 读出二进制流产生的新对象 \*/
10. ByteArrayInputStream bis = **new** ByteArrayInputStream(bos.toByteArray());
11. ObjectInputStream ois = **new** ObjectInputStream(bis);
12. **return** ois.readObject();
13. }

## JDK7中的NIO

Paths:

Paths.get(first, more) 第二个参数是可选参数，或者第一个参数是一个父路径名，然后第二个参数是文件名，返回的是path

Path:

Path接口中resolve方法的作用相当于把当前路径当成父目录，而把参数中的路径当成子目录或是其中的文件，进行解析之后得到一个新路径；

resolveSibling方法的作用与resolve方法类似，只不过把当前路径的父目录当成解析时的父目录；

relativize方法的作用与resolve方法正好相反，用来计算当前路径相对于参数中给出的路径的相对路径；

subpath方法用来获取当前路径的子路径，参数中的序号表示的是路径中名称元素的序号；

startsWith和endsWith方法用来判断当前路径是否以参数中的路径开始或结尾。

在一般的路径中，“.”和“..”分别用来表示当前目录和上一级目录。通过normalize方法可以去掉路径中的“.”和“..”。所有这些方法的返回值都是Path接口的实现对象，因此这些方法可以很容易地级联起来。

FileSystems:

FileSystems.getDefault().getPath("G:\\upload\\","黄大哥.mp3");

FileSystems.getDefault() 得到的是默认的操作系统的fileSystem

## Servlet文件上传下载

首先得使用两个jar包

核心API介绍

1、DiskFileItemFactory

作用:可以设置缓存大小以及临时文件保存位置.

默认缓存大小是 10240(10k).

临时文件默认存储在系统的临时文件目录下.（可以在环境变量中查看）

1、new DiskFileItemFactory();

缓存大小与临时文件存储位置使用默认的.

2、DiskFileItemFactory(int sizeThreshold, File repository)

sizeThreshold :缓存大小

repository:临时文件存储位置

注意，对于无参数构造，也可以设置缓存大小以及临时文件存储位置.

setSizeThreshold(int sizeThreshold)

setRepository(File repository)

2、ServletFileUpload

1、ServletFileUpload upload=new ServletFileUpload(factory);

创建一个上传工具，指定使用缓存区与临时文件存储位置.

2、List<FileItem> items=upload.parseRequest(request);

它是用于解析request对象，得到所有上传项.每一个FileItem就相当于一个上传项.

3、boolean flag=upload.isMultipartContent(request);

用于判断是否是上传.

可以简单理解，就是判断encType="multipart/form-data";

4、设置上传文件大小

void setFileSizeMax(long fileSizeMax) 设置单个文件上传大小

void setSizeMax(long sizeMax) 设置总文件上传大小

5、解决上传文件中文名称乱码

setHeaderEncoding("utf-8");

注意:如果使用reqeust.setCharacterEncoding("utf-8")也可以，但不建议使用。

3、FileItem

1、isFormField

用于判断是否是上传组件.

如果是<input type="file">返回的就是false,否则返回true.

2、getFieldName();

返回值String,得到组件名称 <input name="">

3、getName();

返回值是String,得到的是上传文件的名称.

注意:浏览器不同，它们得到的效果不一样。

1.包含全路径名称 例如: C:\Users\Administrator\Desktop\a.txt

2.只包含上传文件名称 例如:a.txt

4、getString();

这个方法可以获取非上传组件的内容,相当于 getParameter方法作用。

问题:如果信息是中文，会出现乱码，解决方案 getString("utf-8");

如果是上传组件，上传的文件是文本文件，可以获取到文件文件的内容。

但是如果不是文件文件，例如：是一张图片，这样获取合适吗?

5、获取上传文件的内容,保存到服务器端.

item.getInputStream();它是用于读取上传文件内容的输入流.

使用文件复制操作就可以完成文件上传。

6、删除临时文件

item.delete();

String savePath = "G:\\upload\\upload\\upload\\";

File file = **new** File(savePath);

**if**(!file.exists()){

System.***out***.println("文件不存在，需要重新创建。。。");

file.mkdirs();

}

DiskFileItemFactory diskFactory = **new** DiskFileItemFactory();

ServletFileUpload upload = **new** ServletFileUpload(diskFactory);

upload.setHeaderEncoding("UTF-8");//解决上传文件名中文乱码问题

**if**(!upload.*isMultipartContent*(request)){

System.***out***.println("不是表单提交的数据");

**return**;

}

**try** {

List<FileItem> list = upload.parseRequest(request);

**for**(FileItem item : list){

**if**(item.isFormField()){

System.***out***.println(item.getFieldName());

}**else**{

String name = item.getName();

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

InputStream in = item.getInputStream();

FileOutputStream out = **new** FileOutputStream(savePath + name);

**int** len = 0;

**byte** byt[] = **new** **byte**[1024];

**while**((len = in.read(byt)) != -1){

out.write(byt, 0, len);

}

in.close();

out.close();

item.delete();

}

}

} **catch** (FileUploadException e) {

// **TODO** Auto-generated catch block

System.***out***.println("文件上传失败。。。");

下载方式有两种：

* 1. 使用<a>标签给出服务器中的资源地址，如果浏览器能够解析那么就自动加载到页面，不会进行下载，使用的是服务器默认的DefaultServlet
  2. 如果是本地服务器中的资源那么可以在servlet中给出地址，jsp页面只需要给出文件名即可，然后获取response.OutputStream()写回到浏览器中

// 1.得到要下载 的文件名称

String filename = request.getParameter("filename");

filename = new String(filename.getBytes("iso8859-1"), "utf-8");// 解决中文乱码

// 2.在d:/upload目录下查找这个文件是否存在（要下载的文件）

File file = new File("d:/upload/" + filename);

if (file.exists()) {

// /文件存在，完成下载

// 下载注意事项1--设置下载文件的mimeType

String mimeType = this.getServletContext().getMimeType(filename);

response.setContentType(mimeType);

String agent = request.getHeader("user-agent");

if (agent.contains("MSIE")) {

// IE浏览器

filename = URLEncoder.encode(filename, "utf-8");

} else if (agent.contains("Firefox")) {

// 火狐浏览器

BASE64Encoder base64Encoder = new BASE64Encoder();

filename = "=?utf-8?B?"

+ base64Encoder.encode(filename.getBytes("utf-8"))

+ "?=";

} else {

// 其它浏览器

filename = URLEncoder.encode(filename, "utf-8");

}

// 下载注意事项2--永远是下载

response.setHeader("content-disposition", "attachment;filename="

+ filename);

FileInputStream fis = new FileInputStream(file); // 读取要下载文件的内容

OutputStream os = response.getOutputStream(); // 将要下载的文件内容通过输出流写回到浏览器端.

int len = -1;

byte[] b = new byte[1024 \* 100];

while ((len = fis.read(b)) != -1) {

os.write(b, 0, len);

os.flush();

}

os.close();

fis.close();

} else {

throw new RuntimeException("下载资源不存在.");

}

如果要下载的文件在其他的服务器上那么可以使用URL，URLConnection类,可以和其他服务器进行连接

**try**{

String filePath = request.getParameter("path");

String fileName= filePath.substring(filePath.lastIndexOf("/")+1);

String mimeType = **this**.getServletContext().getMimeType(fileName);

response.setContentType(mimeType);

// 下载注意事项2--永远是下载

response.setHeader("content-disposition", "attachment;filename="

+ fileName);

URL url = **new** URL(filePath);

URLConnection con = url.openConnection();

InputStream in = con.getInputStream();

OutputStream out = response.getOutputStream();

**byte** byt[] = **new** **byte**[1024 \* 4];

**int** len = 0;

**while**((len = in.read(byt)) != -1){

out.write(byt, 0, len);

out.flush();

}

in.close();

out.close();

}**catch**(Exception e){

System.***out***.println("失败了。。。");

}

注意：

response.setHeader("content-disposition","attachment;filename="+ fileName)使用这个永远都是下载

response.setContextType()其实就是告诉浏览器以什么方式（文件还是图片）响应回去

总结：文件上传下载其实主要还是流的传递，但是首先你得获得这个流才行

## JSTL和EL之间的区别

1. JSTL常用于判断，而EL则是用来显示数据的
2. JSTL需要在添加一个标签头<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%> ，但是EL不要引入任何东西，JSTL和EL之间的结合可以使网页不再有<%=表达式%>,EL表达式使用的是${ }

## OGNL是Struts2标签库

首先也需要引入一个标签头，然后常用的标签头有#，%, $

1．#符号

#符号的用途一般有三种。

* 访问非根对象属性，例如#session.msg表达式，由于Struts 2中值栈被视为根对象，所以访问其他非根对象时，需要加#前缀。实际上，#相当于ActionContext. getContext()；#session.msg表达式相当于ActionContext.getContext().getSession(). getAttribute("msg") 。
* 用于过滤和投影（projecting）集合，如persons.{?#this.age>25}，persons.{?#this.name=='pla1'}.{age}[0]。
* 用来构造Map，例如示例中的#{'foo1':'bar1', 'foo2':'bar2'}。

2．%符号

%符号的用途是在标志的属性为字符串类型时，计算OGNL表达式的值，这个类似js中的eval，很暴力。

3．$符号

$符号主要有两个方面的用途。

— 在国际化资源文件中，引用OGNL表达式，例如国际化资源文件中的代码：reg.agerange=国际化资源信息：年龄必须在${min}同${max}之间。

— 在Struts 2框架的配置文件中引用OGNL表达式，

## 27、Hibernate的延迟加载出现的问题

懒加载出现的异常：

org.hibernate.LazyInitializationException: could not initialize proxy - no Session

解决办法就是配置一个过滤器

<!--配置过滤器，解决Hibernate的延迟加载问题 -->

<filter>

<filter-name>openSession</filter-name>

<filter-class>org.springframework.orm.hibernate3.support.OpenSessionInViewFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>openSession</filter-name>

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

</filter-mapping>

## 28、session.removeAttribute()和session.invalidate()之间的区别？

Session.removeAttribute()是将session中的某一个属性删除

Session.invalidate()是将session中的所有属性都删除

举例：

一个浏览器浏览一个网站会保存很多的属性在session中

Session.removeAttribute()是将某个属性删除，但是session.invalidate()则是将所有的属性删除（仅限于当前浏览器）

## 29、jsp页面中使用jstl中出现的异常

According to TLD or attribute directive in tag file, attribute test does not accept any expressions

解决方案：将

<%@ taglib uri="<http://java.sun.com/jstl/core>" prefix="c"%>

改为

<%@ taglib uri=<http://java.sun.com/jstl/core_rt> prefix="c"%>

## 30、从session中取出变量会持久化

User user = (User) request.getSession.getAttribute（“exitUser”）；

user会持久化，向user中存数据会再session中更新，取出的值会是最新的

## 31、解决a different object with the same identifier value was already associated with the session错误

原因：在hibernate中同一个session里面有了两个相同标识，但是是不同实体  
  
解决方法一：session.clean()

　　PS：如果在clean操作后面又进行了saveOrUpdate(object)等改变数据状态的操作，有可能会报出"Found two representations of same collection"异常。

　　解决方法二：session.refresh(object)

　　PS：当object不是数据库中已有数据的对象的时候，不能使用session.refresh(object)因为该方法是从hibernate的session中去重新取object，如果session中没有这个对象，则会报错所以当你使用saveOrUpdate(object)之前还需要判断一下。

　　解决方法三：session.merge(object)

　　PS：Hibernate里面自带的方法，推荐使用。

2、Found two representations of same collection

　　错误原因：见1。

　　解决方法：session.merge(object)

以上两中异常经常出现在一对多映射和多对多映射中

注：上文中的session指的是Dao层中的，即Session session = HibernateSessionFactory.currentSession();，在整合的框架中可以直接使用this.getHibernateTemplate().merge(T entity )。

Merage()这个方法中的对象成员必须要补全，若没有补全则会在数据库中将没有赋值的成员变量赋值为空

## 32、Java代码执行顺序

如果有静态化代码先执行静态化代码，然后执行非静态化代码

如果有两个静态化代码块则由上到下执行