## 彻底解决springMVC中文乱码

一、页面编码

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

<meta http-equiv="Content-Type" content="text/html;charset=UTF-8"/>

二、URL中的乱码

　　改tomcat中server.xml中Connector的port=“8080”，加上一个 URIEncoding=”utf-8”

三、配置过滤器，指定所有请求的编码

　　（1）配置spring的编码过滤器，为了防止spring中post方式提交的时候中文乱码，方法：修改web.xml文件，添加spring的编码过滤器

复制代码

<!-- 配置编码方式过滤器,注意一点:要配置在所有过滤器的前面 -->

<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>

复制代码

　　（2）配置编码过滤器，方法：先创建filter类，再修改web.xml文件，注意的是放在spring的编码过滤器之后

　　filter类：

复制代码

package com.qiyuan.filter;

import java.io.IOException;

import javax.servlet.Filter;

import javax.servlet.FilterChain;

import javax.servlet.FilterConfig;

import javax.servlet.ServletException;

import javax.servlet.ServletRequest;

import javax.servlet.ServletResponse;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public class EncoidingFilter implements Filter {

private String encoding="";

@Override

public void destroy() {

// TODO Auto-generated method stub

}

//过滤方法 是否往下执行

@Override

public void doFilter(ServletRequest arg0, ServletResponse arg1, FilterChain chain)

throws IOException, ServletException {

HttpServletRequest request=(HttpServletRequest)arg0;

HttpServletResponse response=(HttpServletResponse)arg1;

request.setCharacterEncoding(encoding);

response.setCharacterEncoding(encoding);

//过滤通行证

chain.doFilter(request, response);

}

//根据web.xml文件的配置进行初始化

@Override

public void init(FilterConfig arg0) throws ServletException {

this.encoding = arg0.getInitParameter("Encoding");

}

}

复制代码

　　修改web.xml，添加如下配置：

复制代码

<!-- 配置编码过滤 -->

<filter>

<filter-name>EncoidingFilter</filter-name>

<filter-class>com.qiyuan.filter.EncoidingFilter</filter-class>

<init-param>

<param-name>Encoding</param-name>

<param-value>utf-8</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>EncoidingFilter</filter-name>

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

</filter-mapping>

**特别注意:在tomcat中修改server.xml的配置加编码格式,只是对get请求有效,在web.xml中添加编码过滤器只是对post请求有效.**

## Java的多种map介绍

HashMap

最常用的Map,它根据键的HashCode 值存储数据,根据键可以直接获取它的值，具有很快的访问速度。HashMap最多只允许一条记录的键为Null(多条会覆盖);允许多条记录的值为 Null。非同步的。

TreeMap

能够把它保存的记录根据键(key)排序,默认是按升序排序，也可以指定排序的比较器，当用Iterator 遍历TreeMap时，得到的记录是排过序的。TreeMap不允许key的值为null。非同步的。

Hashtable

与 HashMap类似,不同的是:key和value的值均不允许为null;它支持线程的同步，即任一时刻只有一个线程能写Hashtable,因此也导致了Hashtale在写入时会比较慢。

LinkedHashMap

保存了记录的插入顺序，在用Iterator遍历LinkedHashMap时，先得到的记录肯定是先插入的.在遍历的时候会比HashMap慢。key和value均允许为空，非同步的。

## 实现runnable接口开启多线程

这里需要将runnable接口的实现类的实例当做参数传给Thread类,有thread类的实例使用start方法开启线程.

## Java1.8的stream



## Lambda表达式



## Jdk1.8的抽象方法

在jdk1.8及以后,抽象类中也可以有实体方法,当普通类A继承了抽象类B后,假如实体类对象A调用的是抽象类B中有方法体的方法,且在实体类A中没有覆盖该方法(注意这里的覆盖是在方法上打了@Override标签)那么会执行抽象类中的方法,要想执行实现类中的方法就加上@Override

## Tomcat的启动和停止

使用./shutdown.sh或者是sh shutdown.sh关闭Tomcat

使用./startup.sh或者是sh startup.sh启动Tomcat

## Linux的less命令

* ctrl + F - 向前移动一屏
* ctrl + B - 向后移动一屏

d 向后翻半页

y 向前滚动一行

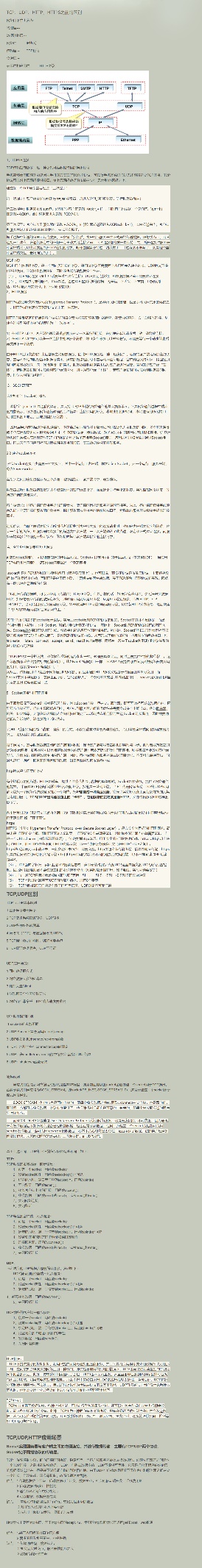
[pagedown]： 向下翻动一页  
[pageup]： 向上翻动一页

* G - 移动到最后一行
* g - 移动到第一行

## 接口中的常量

接口中可以声明参数,但是参数一定是 final + static类型的,加static的意思是接口不能被实例化,如果一个接口没有被继承,那么接口中的参数是不能被访问到的,所以需要静态字段修饰,加final修饰是因为接口被多个类继承的时候,如果某个类实体修改了参数,别的类也会发生变化,这样不符合接口的一致性原则.

## Tcp与udp和http关系



## Linux中jar解压缩

Jar -zxvf jar名称

## Linux的tree命令

显示当前目录的文件的目录结构

## 迭代器的remove方法

Iterator的remove方法在遍历集合时候调用iter.remove(),删除返回的当前的元素,且不会破坏后续循环的执行.

## CURL命令

Linux模拟浏览器访问

curl -XGET <http://192.168.154.200:9001/>

## telnet

测试端口是否通

telnet ip 端口

## 虚拟机的配置

关闭字节码校验: -noverify

## Tomcat实现原理：

## Jsp表达式

<% java代码（变量，方法，语句等）%>

<%! 定义的变量和方法等 %>

<%= 往外部输出%>

## 反射

反射获取class对象的三种方式

类名.class 实例化对象.getClass() Class.forName(“类路径”)

获取构造器

Constructor constructor = clazz.getConstructor(int.class) //这里获取构造器，其中的参数根据构造器的参数来确定，需要注意的是不支持拆装箱

Object object = constructor.newInstance(123);

通过clazz获取方法

Method method = clazz.getMethod(“init”);

Method.invoke(object);

## 类之间的关系

类之间的关系分成三种 依赖、聚合、继承

例如账户、订单、商品项 这三个类来举例

订单依赖于账户，需要有账户类的实例来记录订单信息，订单中的方法会用到账户类信息，而商品项相对于订单来说是聚合，一个订单可能会有多个商品。依赖是 uses-a的关系，而聚合是has-a的关系，继承是is –a 的关系。

## 系统变量之System.getenv()和System.getProperty()





## git的强制推送

git push -f -u origin master