**第一部分：Java EE基本设置**

**web项目结构WebContent根目录：**

WEB-INF: //放配置文件，外部没法直接访问，只能通过action、servlet等服务器端转向才行，所以image、css、js等资源文件不能放在这

classes //编译好的.class文件

lib //框架等所需要的类包jar

content //webstorm建立项目，加上bower，lib等在外面引用

web.xml //配置管理web：jsp、servlet、listener、filter、所有框架中核心的Servlet或Filter都需要在这里配置。

images：//放置图片资源

css:

js:

index.jsp：伪首页，利用forward直接指向WEB-INF下的首页资源

**与WebContent同级目录：**

src : //放置编写的.java文件、strut2.xml文件，运行的时候会自动加载到WEB-INF/class下

**注意点：**

1. web项目中绝对路径是以tomcat即8080：为基准，不包含项目名称。相对路径则是以当前路径为基准。
2. jdk可以向下兼容，不能向上兼容，即编译好的1.8在1.7的环境下无法运行，就算没有用到1.8的特性也不行，所以部署项目要么用1.7编译好了部署上去，要么把源码放在服务器上再编译。
3. 在JavaWeb Project中，使用相对路径是读不到文件的，因为JavaWeb Project与Java Project不同，项目最终要部署在web服务器中，文件只能使用绝对路径来读取。
4. source/source folder区别：需要编译的文件都需要放在source folder下面，source下的文件不会被编译，且folder下的源文件编译好自动output到classes下面。

**第二部分：jsp servlet**

**Jsp四种语法：java语言在html中表示**

<%-- --%> //jsp注释

<%! public int count;

public String info(){

return "hello";

}

%> //jsp声明

<%=count%> //jsp输出

<% %> //jsp脚本

**Jsp3个编译指令：**

格式 <%@ page/include 属性名=“属性值”…%>

Page: jsp页面的描述信息，通知servlet引擎处理信息

其他两个不重要。

**Jsp7个动作指令：完全可以用jsp脚本替换**

格式 <jsp：forward page=“”>

**Jsp9个内置对象：**

Jsp中的java脚本和html都会被解析成\_jspService方法里的可执行java代码，但是怎么和核心servlet通信，需要靠Map结构（下面4种）作为中介，web1.0 。

application:全局共享（直到整个web关闭） setAttrubute getAttribute来实现，但servlet中需要靠函数获取此对象

getInitParameter可以用来获取web.xml中的配置参数--这些参数在<context-param>中

request： 封装着一次用户请求和请求参数，HttpServletRequest接口实例（交互信息使用）。使用setAttribute，getAttribute设置数据，getRequestDispatcher(path).forward()来发送请求(和jsp动作指令forward功能相同)

Session：可以保存一次会话（连接服务器到断开）中数据（客户端状态的数据，如购物记录），set/get来使用

Page：只能被当前jsp中java脚本使用

**Servlet：**

没有内置对象，获取这些对象需要显示函数获取；静态html必须使用页面输出流输出；service()函数完全等于解析jsp的\_jspService()函数，重写可以相应所有客户端请求。

通过注解/通过web.xml两种方式，对哪种url响应

**Filter：**

通过注解或web.xml方式配置，对哪种url拦截处理。主要功能就是实现多个servlet中功能复用，代码就是servlet中公共部分，语法基本和servlet一样。

**Listener:**

同上两种配置方式，对整个web发生的事件响应，实现对应的接口则响应对应的事件，需要重写接口中的函数。

ApplicationContext:代表当前web对象

**第三部分：Git Github使用**

**本地仓库：**

2 **$ git init** 即可初始化workbench目录下文件成git代码库

3 **$ git add .** 把workbench目录下文件提交到暂存区(每次修改之后都需要先add修改的文件，然后再commit)

4 **$ git commit –m “message”** 提交暂存区到本地仓库

9 **$ git status** 显示当前本地仓库的状态，修改了什么，删除了什么，有没有提交等。

10 **$ git diff readme.txt** 对比工作区和版本库该文件区别

11 **$ git log**显示从最近到最远的提交日志 git commit –m “内容”

12 **$ git reflog** 记录每一次提交的commit-id号

13 **$ git reset --hard commit-id** 回滚到commit-id所指版本

14 **$ git checkout -- readme.txt** 放弃工作区的修改(删除，增加等操作都算)，回到最近一次commit或add时状态。

**远程仓库：**

4 **$ ssh-keygen -t rsa -C xiyihai@163.com** 登陆GitHub点“Add SSH Key”，在Key文本框里粘贴id\_rsa.pub文件的内容

5 **$ git remote add origin git@github.com:xiyihai/PicLabel.git** 关联远程仓库(一次即可，名字需要和远程相同)，若提示出错信息：fatal: remote origin already exists.则输入 **git remote rm origin**，删除远程仓库master

6 **$ git push origin master** 把本地代码push到远程master分支(操作目录下已建立git仓库)

7 **$ git pull origin master** 把远程最新代码更新到本地(多人协作时，应该先pull)

8 **$ git clone git@github.com:xiyihai /PicLabel.git** 把项目clone到本地，此时该项目已经是git仓库，且关联好了远程仓库。但若是该github上没有你的ssh，则应该用5步骤重新关联一个远程仓库。