Git（16：20）

取消在当前目录下进行本地初始化（不小心输入了git init）

执行rm -rf .git

在自己的项目目录中建仓库：git init

#创建本地test分支

$ git branch test

##查看本地分支有哪些

$ git branch

doc

\* master

##将本地test分支推送到远程服务器

$ git push origin test

##切换到test分支

$ git checkout test

Switched to branch 'test'

##提交信息

giscafer@LAOHOUBIN-PC /G/002\_project/Comments (test)

$ git commit -m '提交数据结构表设计文档到test分支上'

[test 867e877] 提交数据结构表设计文档到test分支上

查看运程分支：

git branch –r

查看全部分支：

git branch –a

本地分支推送到远程分支

git push -u origin master

SSH

（1） 首先在客户端生成一对密钥（ssh-keygen）；

　　（2） 并将客户端的公钥ssh-copy-id 拷贝到服务端；

　　（3） 当客户端再次发送一个连接请求，包括ip、用户名；

　　（4） 服务端得到客户端的请求后，会到authorized\_keys中查找，如果有响应的IP和用户，就会随机生成一个字符串，例如：qwer；

　　（5） 服务端将使用客户端拷贝过来的公钥进行加密，然后发送给客户端；

　　（6） 得到服务端发来的消息后，客户端会使用私钥进行解密，然后将解密后的字符串发送给服务端；

（7） 服务端接受到客户端发来的字符串后，跟之前的字符串进行对比，如果一致，就允许免密码登录。

FTP

public interface FtpServerContext extends FtpletContext {

ConnectionConfig getConnectionConfig();

MessageResource getMessageResource();

FtpletContainer getFtpletContainer();

Listener getListener(String name);

Map<String, Listener> getListeners();

CommandFactory getCommandFactory();

void dispose();

ThreadPoolExecutor getThreadPoolExecutor();

}

# 死锁

**产生死锁的四个必要条件：**

（1） 互斥条件：一个资源每次只能被一个进程使用。  
（2） 请求与保持条件：一个进程因请求资源而阻塞时，对已获得的资源保持不放。  
（3） 不剥夺条件:进程已获得的资源，在末使用完之前，不能强行剥夺。  
（4） 循环等待条件:若干进程之间形成一种头尾相接的循环等待资源关系。

# HTLML

<body>标签里面的java代码几乎与js里面的java代码同时执行，实际上很多时候两时间都一样。

<%xxx%>这是jsp中嵌入的java程序，是在服务器端运行的。  
<script>xxx</script>这是页面脚本，是在浏览器客户端运行的。

3.<body>里面的onload方法在页面结束加载之后触发。

1. **<html>**
2. **<head>**
3. **<script>**
4. alert("script-head");
5. **</script>**
6. **</head>**
7. **<body** onload="alert('html-tag');"**>**
8. **<script>**
9. alert("script-body");
10. **</script>**
11. **</body>**
12. **<script>**
13. alert("script-body-outer");
14. **</script>**
15. **</html>**
16. **<script>**
17. alert("script-html-outer");
18. **</script>**

script-head、script-body、script-body-outer、script-html-outer、html-tag为输出结果。

# Spring

@RestController

@RequestMapping("/classPath")

**public** **class** **Application** {

@RequestMapping("/methodPath")

**public** String **method**() {

**return** "mapping url is /classPath/methodPath";

}

}

@RequestMapping("/users/{username}")

**public** String **userProfile**(@PathVariable("username") String username) {

**return** String.format("user %s", username);

}

@RequestMapping("/posts/{id}")

**public** String **post**(@PathVariable("id") **int** id) {

**return** String.format("post %d", id);

}

在之前所有的@RequestMapping注解的方法中，返回值字符串都被直接传送到浏览器端并显示给用户。但是为了能够呈现更加丰富、美观的页面，我们需要将HTML代码返回给浏览器，浏览器再进行页面的渲染、显示。

一种很直观的方法是在处理请求的方法中，直接返回HTML代码，但是这样做的问题在于——一个复杂的页面HTML代码往往也非常复杂，并且嵌入在Java代码中十分不利于维护。更好的做法是将页面的HTML代码写在模板文件中，渲染后再返回给用户。为了能够进行模板渲染，需要将@RestController改成@Controller：

Servlet（Server Applet）是Java Servlet的简称，称为小服务程序或服务连接器，用Java编写的服务器端程序，主要功能在于交互式地浏览和修改数据，生成动态Web内容。

该文本行的流可以这样处理：  
InputStream input = … ; // get the InputStream from the client socket

|  |  |  |
| --- | --- | --- |
| 1 | BufferedReader reader = new BufferedReader(new InputStreamReader(input)); | |
| 2 |  |

|  |  |
| --- | --- |
| 3 | String nameLine   = reader.readLine(); |
| 4 | String ageLine    = reader.readLine(); | |

|  |  |
| --- | --- |
| 5 | String emailLine  = reader.readLine(); |
| 6 | String phoneLine  = reader.readLine(); |