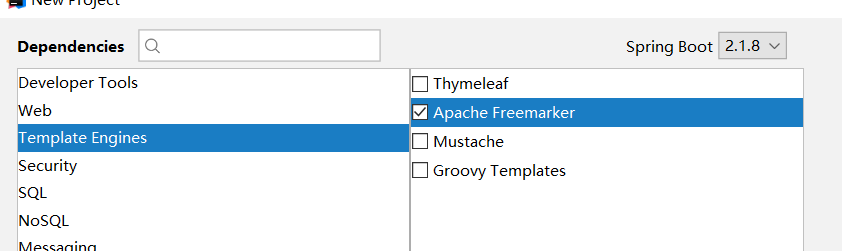
# 一、FreeMarker

是一个模板引擎技术，用于生成静态页面。Freemarker需要生成资源的话需要有这两样东西：模板+动态的数据。

## 1.初体验

### 1）引入依赖



### 2）代码

|  |
| --- |
| *//引入模板引擎的配置对象* @Autowired **private** Configuration **configuration**;   @Test **public void** contextLoads() **throws** IOException, TemplateException {      *//通过配置对象的方法得到一个模板对象* Template template = **configuration**.getTemplate(**"index.ftl"**);   *//提供数据* Map<String,Object> data = **new** HashMap<>();   data.put(**"username"**,**"laowang"**);   *//设置输出流：静态资源存放的位置。 模板+数据==静态资源* Writer out = **new** FileWriter(**"D:\\教学相关\\bk\\支付宝支付\\my-free-marker-demo\\src\\main\\resources\\static\\index.html"**);  *//模板对象处理模板和数据生成静态资源* template.process(data,out);    } |

## 2.FreeMarker使用细节

1）关于对象的用法

2）关于对象中日期的用法

|  |
| --- |
| **${stu**.**username}  ${stu**.**age}  ${stu**.**birthday**?**date}  ${stu**.**birthday**?**time}  ${stu**.**birthday**?**datetime}** |

效果：

|  |
| --- |
| 小强  20  2019-9-24  10:22:21  2019-9-24 10:22:21 |

3）在模板中遍历集合

|  |
| --- |
| **<#list stus as stu>** 序号：**${stu\_index**+1**}** 姓名：**${stu**.**username}** 年龄：**${stu**.**age}** 出生日期：**${stu**.**birthday**?**date} </#list>** |

4）在模板中进行if-else判断

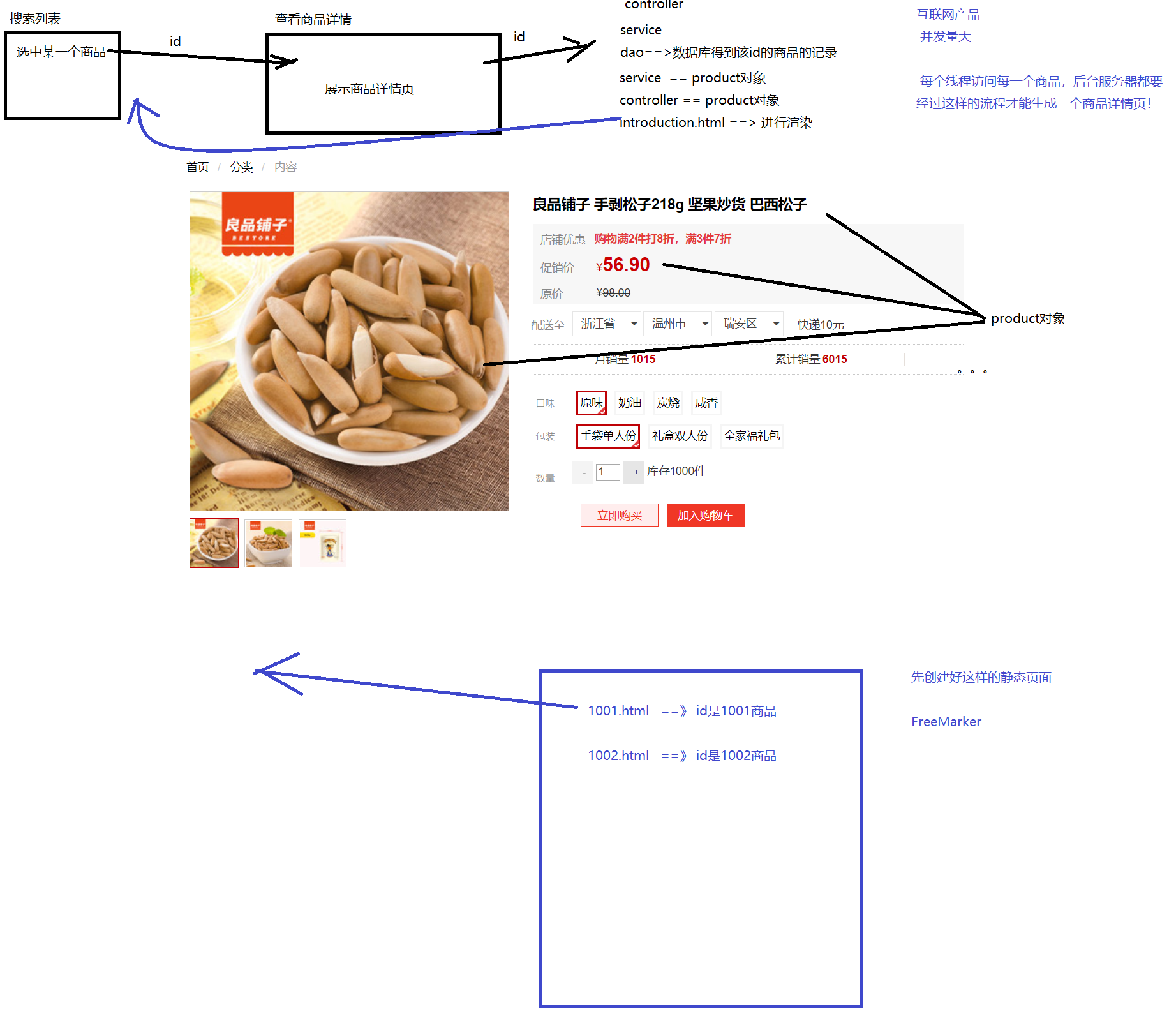
|  |
| --- |
| **<#if** (**stu2**.**age**<21)**>** 年龄太小，不能领证  **<#elseif** (**stu2**.**age**<28)**>** 适合结婚  **<#else>** 剩男剩女 **</#if>** |

5）关于null值的使用

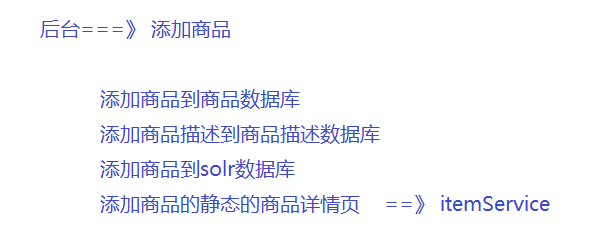
如果数据中的值是null值，那么在使用时必须在键的后面加上"!"或“!'默认值'”

|  |
| --- |
| **${msg**!**'默认值'}  ${msg**!**}** |

# 二、为什么要使用freemarker



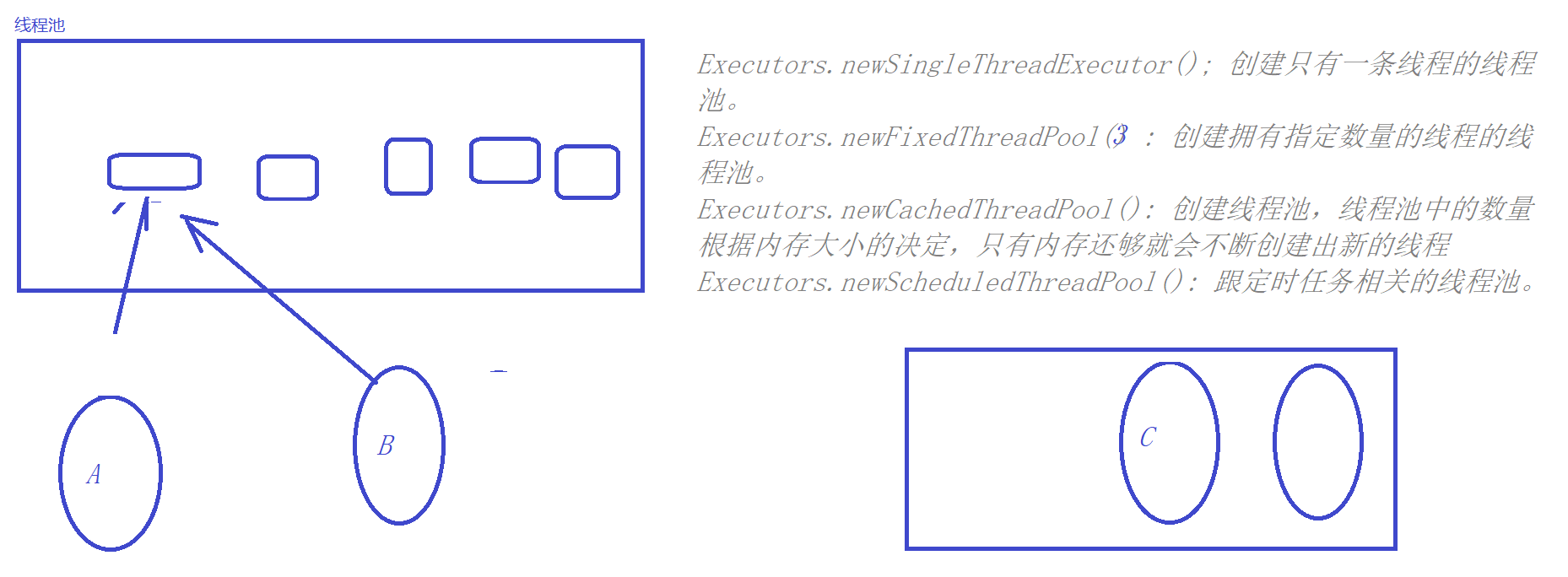
## 1.添加完商品后就要生成一个该商品详情的静态页面



代码：

|  |
| --- |
| @Autowired **private** Configuration **configuration**;  @Value(**"${itemService.path}"**) **private** String **itemServicePath**;    **public** ResultBean initDetail(TProduct product){   **try** {  Template template = **configuration**.getTemplate(**"introduction.ftl"**);    Map<String,Object> data = **new** HashMap<>();  data.put(**"product"**,product);   Writer out = **new** FileWriter(**itemServicePath**+product.getId()+**".html"**);   template.process(data,out);    } **catch** (IOException | TemplateException e) {  e.printStackTrace();  **return** ResultBean.*error*(**"生成静态资源失败..."**);  }    **return** ResultBean.*success*(**"生成静态资源成功..."**); } |

2.批量生成多个静态资源



关于多线程中，线程池的创建的注意事项

|  |
| --- |
| */\*\*  \* 使用多线程来批处理  \*  \* 如何使用多线程==》线程池来创建  \*  \* 线程池如何实现？  \*  \* jdk提供了四种线程池的创建，用哪个？哪个都不用！因为每个都会造成内存的溢出！  \*  \* 线程池中的关键参数  \* 初始化线程数大小  \* 最大线程数大小  \* 发呆时间： 进过多少时间后，该线程依然处于空闲状态，那么就会被回收。 20  \* 时间单位：  \* 队列： 存放等待的用户  \*  \*  \*  \** ***@param products*** *\** ***@return*** *\*/* @Override **public** ResultBean batchGenerateDetail(List<TProduct> products) {   *//当前机器的cpu核心数* **int** corePooleSize = Runtime.*getRuntime*().availableProcessors();   *//自定义线程池，这个线程池根据cpu的核心数来创建线程数量，最大化使用cpu 提高效率，而且队列也不会过长，防止内存溢出* ExecutorService pool = **new** ThreadPoolExecutor(corePooleSize,corePooleSize\*2,  20L, TimeUnit.***MILLISECONDS***,**new** LinkedBlockingDeque<Runnable>(1000));       */\*  Executors.newSingleThreadExecutor(); 创建只有一条线程的线程池。 造成内存溢出： 队列过长  Executors.newFixedThreadPool() : 创建拥有指定数量的线程的线程池。 5 造成内存溢出： 队列过长  Executors.newCachedThreadPool(): 创建线程池，线程池中的数量根据内存大小的决定，只有内存还够就会不断创建出新的线程  造成内存溢出： 线程数过多。  Executors.newScheduledThreadPool(): 跟定时任务相关的线程池。  \*/* **return null**; } |

3.线程池的使用

|  |
| --- |
| @Override  **public** ResultBean batchGenerateDetail(List<TProduct> products) {   *//当前机器的cpu核心数* **int** corePooleSize = Runtime.*getRuntime*().availableProcessors();   *//自定义线程池，这个线程池根据cpu的核心数来创建线程数量，最大化使用cpu 提高效率，而且队列也不会过长，防止内存溢出* ExecutorService pool = **new** ThreadPoolExecutor(corePooleSize,corePooleSize\*2,  20L, TimeUnit.***MILLISECONDS***,**new** LinkedBlockingDeque<Runnable>(1000));   */\*  Executors.newSingleThreadExecutor(); 创建只有一条线程的线程池。 造成内存溢出： 队列过长  Executors.newFixedThreadPool() : 创建拥有指定数量的线程的线程池。 5 造成内存溢出： 队列过长  Executors.newCachedThreadPool(): 创建线程池，线程池中的数量根据内存大小的决定，只有内存还够就会不断创建出新的线程  造成内存溢出： 线程数过多。  Executors.newScheduledThreadPool(): 跟定时任务相关的线程池。  \*/* List<Future<Boolean>> futures = **new** ArrayList<>();  **for** (TProduct product : products) {  *//使用一个callable对象，提交给线程池，线程池会使用一条线程来执行里面的call方法* Future<Boolean> future = pool.submit(**new** GenerateDetailCallable(product));  *//future里存放了call方法执行完后的返回结果* futures.add(future);  } *// System.out.println(futures);* **return** ResultBean.*success*(**"批量生成资源成功..."**);  }    *//自定义Callable的实现类，重写call方法，call方法和run方法的区别就是call可以有返回值* **class** GenerateDetailCallable **implements** Callable<Boolean>{    **private** TProduct **product**;  **public** GenerateDetailCallable(TProduct product){  **this**.**product**=product;  }    @Override  **public** Boolean call() {  **try** {  *//使用freemarker生成静态页面* Template template = **configuration**.getTemplate(**"introduction.ftl"**);   Map<String,Object> data = **new** HashMap<>();  data.put(**"product"**,**this**.**product**);   Writer out = **new** FileWriter(**itemServicePath**+**product**.getId()+**".html"**);   template.process(data,out);  } **catch** (Exception e) {  e.printStackTrace();  **return false**;  }    **return true**;  }  } |

# 三、把静态资源部署在nginx上

web服务器：

tomcat： servlet容器 作为静态资源服务器

nginx： 静态资源服务器 单纯作为静态资源服务器的，性能比tomcat好！

也不止用于静态资源服务器，还能作反向代理，负载均衡

## 1.如何使用nginx作静态资源服务器