JAVASE-DAY09/DAY10 晚课

1. 多线程

线程:单一的顺序执行流程就是一个线程，顺序执行:代码一句一句的先后执行。

多线程:多个线程并发执行。线程之间的代码是快速被 CPU 切换执行的，造成一种感官上"同时"执行的效果。

A B C D

A--CPU 0.1

B--

1.1 线程的创建方式

1. **继承 Thread，重写 run 方法，在 run 方法中定义线程要执行的任务**

- 优点:
  - 结构简单，便于匿名内部类创建
- 缺点:
  - 继承冲突:由于 java 单继承，导致如果继承了线程就无法再继承其他类去复用方法
  - 耦合问题:线程与任务耦合在一起，不利于线程的重用
  - 高内聚 低耦合
  - A B

2. **实现 Runnable 接口单独定义线程任务**

- 优点:
  - 由于是实现接口，没有继承冲突问题
  - 线程与任务没有耦合关系，便于线程的重用

- 缺点:
  - 创建复杂一些(其实也不能算缺点)

  线程的生命周期（线程的几种状态）（记）

  a. 创建状态 new Thread();

  b. 就绪状态 start(); // 等待 CPU 的调度

  c. 运行状态 run();

  d. 阻塞状态 sleep();

  e. 死亡状态

2. 线程并发安全

2.1 什么是并发安全

当多个线程并发操作同一临界资源,由于线程切换时机不确定,导致操作临界资源的顺序出现混乱严重时可能导致系统瘫痪。临界资源:操作该资源的全过程同时只能被单个线程完成.

```
Java
package cn.tedu.thread;


/**
 * 多线程并发安全问题
 * 当多个线程并发操作同一临界资源，由于线程切换的时机不确定，导致操作顺序出现
 * 混乱，严重时可能导致系统瘫痪。
 * 临界资源:同时只能被单一线程访问操作过程的资源。
 */
public class SyncDemo {
    public static void main(String[] args) {
        Table table = new Table();
        Thread t1 = new Thread(){
            public void run(){
                while(true){
```

```
                    int bean = table.getBean();
                    Thread.yield();
                    System.out.println(getName()+":"+bean);
                }
            }
        };
        Thread t2 = new Thread(){
            public void run(){
                while(true){
                    int bean = table.getBean();
                    /*
                        static void yield()
                        线程提供的这个静态方法作用是让执行该方法的线程
                        主动放弃本次时间片。
                        这里使用它的目的是模拟执行到这里 CPU 没有时间了
, 发生
                        线程切换，来看并发安全问题的产生。
                     */
                    Thread.yield();
                    System.out.println(getName()+":"+bean);
                }
            }
        };
        t1.start();
        t2.start();
    }
}

/**
 * 被访问的临界资源
 */
class Table{
    private int beans = 20;//桌子上有 20 个豆子

    public int getBean(){
        if(beans==0){
            throw new RuntimeException("没有豆子了!");
        }
        Thread.yield();
        return beans--;
    }
```

```
}
```

2.2 解决并发安全问题

解决并发安全问题的本质就是将多个线程并发(同时)操作改为同步(排队)操作来解决。

同步与异步的概念:同步和异步都是说的多线程的执行方式。

- 异步执行：多条线程抢占资源，没有先后顺序
- 同步执行：多条线程执行存在先后顺序

2.3 Synchronized 关键字

synchronized 有两种使用方式

- 在方法上修饰,此时该方法变为一个同步方法
- 同步块,可以更准确的锁定需要排队的代码片段

同步方法

当一个方法使用 synchronized 修饰后,这个方法称为"同步方法",即:多个线程不能同时 在方法内部执行.只能有先后顺序的一个一个进行. 将并发操作同一临界资源的过程改为同步执行就可以有效的解决并发安全问题.

```
Java
package cn.tedu.thread;


/**
 * 多线程并发安全问题
 * 当多个线程并发操作同一临界资源，由于线程切换的时机不确定，导致操作顺序出现
 * 混乱，严重时可能导致系统瘫痪。
 * 临界资源:同时只能被单一线程访问操作过程的资源。
 */
public class SyncDemo {
```

```
    public static void main(String[] args) {
        Table table = new Table();
        Thread t1 = new Thread(){
            public void run(){
                while(true){
                    int bean = table.getBean();
                    Thread.yield();
                    System.out.println(getName()+":"+bean);
                }
            }
        };
        Thread t2 = new Thread(){
            public void run(){
                while(true){
                    int bean = table.getBean();
                    /*
                        static void yield()
                        线程提供的这个静态方法作用是让执行该方法的线程
                        主动放弃本次时间片。
                        这里使用它的目的是模拟执行到这里 CPU 没有时间了
, 发生
                        线程切换，来看并发安全问题的产生。
                     */
                    Thread.yield();
                    System.out.println(getName()+":"+bean);
                }
            }
        };
        t1.start();
        t2.start();
    }
}

class Table{
    private int beans = 20;//桌子上有 20 个豆子

    /**
     * 当一个方法使用 synchronized 修饰后，这个方法称为同步方法，多个线
程不能
     * 同时执行该方法。
```

```
     * 将多个线程并发操作临界资源的过程改为同步操作就可以有效的解决多线
程并发
     * 安全问题。
     * 相当于让多个线程从原来的抢着操作改为排队操作。
     */
    public synchronized int getBean(){
        if(beans==0){
            throw new RuntimeException("没有豆子了!");
        }
        Thread.yield();
        return beans--;
    }
}
```

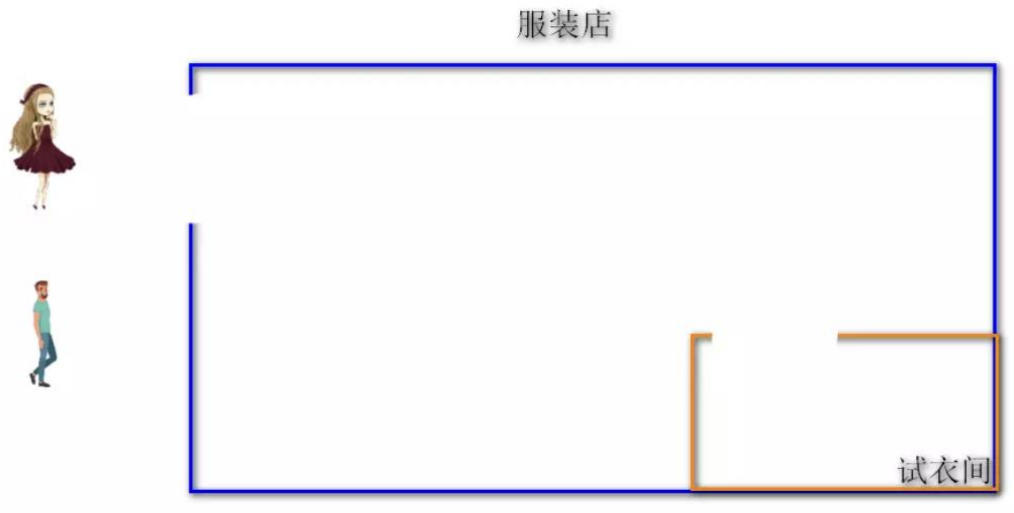
同步块

有效的缩小同步范围可以在保证并发安全的前提下尽可能的提高并发效率.同步块可以更准确的控制需要多个线程排队执行的代码片段.

语法:

```
Java
synchronized(同步监视器对象){
需要多线程同步执行的代码片段
}
```

同步监视器对象即上锁的对象,要想保证同步块中的代码被多个线程同步运行,则要求多个线程看到的同步监视器对象是同一个.




```Java
package cn.tedu.thread;


/**
 * 有效的缩小同步范围可以在保证并发安全的前提下尽可能提高并发效率。
 *
 * 同步块
 * 语法:
 * synchronized(同步监视器对象){
 *     需要多个线程同步执行的代码片段
 * }
 * 同步块可以更准确的锁定需要多个线程同步执行的代码片段来有效缩小排队范围。
 */
public class SyncDemo02 {
    public static void main(String[] args) {
        Shop shop = new Shop();
        Thread t1 = new Thread(){
            public void run(){
                shop.buy();
            }
        };
        Thread t2 = new Thread(){
            public void run(){
```

```
                shop.buy();
            }
        };
        t1.start();
        t2.start();
    }
}

class Shop{
    public void buy(){
        /*
            在方法上使用 synchronized，那么同步监视器对象就是 this。
         */
//    public synchronized void buy(){
        Thread t = Thread.currentThread();//获取运行该方法的线程
        try {
            System.out.println(t.getName()+":正在挑衣服...");
            Thread.sleep(5000);
            /*
                使用同步块需要指定同步监视器对象，即:上锁的对象
                这个对象可以是 java 中任何引用类型的实例，只要保证多个需要排队
                执行该同步块中代码的线程看到的该对象是"同一个"即可
             */
            synchronized (this) {
//            synchronized (new Object()) {//没有效果!
                System.out.println(t.getName() + ":正在试衣服...");
                Thread.sleep(5000);
            }

            System.out.println(t.getName()+":结账离开");
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

    }
}
```

3. 常见面试题

3.1 线程的创建方式

1. **继承 Thread，重写 run 方法，在 run 方法中定义线程要执行的任务**

- 优点:
  - 结构简单，便于匿名内部类创建
- 缺点:
  - 继承冲突:由于 java 单继承，导致如果继承了线程就无法再继承其他类去复用方法
  - 耦合问题:线程与任务耦合在一起，不利于线程的重用

2. **实现 Runnable 接口单独定义线程任务**

- 优点:
  - 由于是实现接口，没有继承冲突问题
  - 线程与任务没有耦合关系，便于线程的重用
- 缺点:
  - 创建复杂一些(其实也不能算缺点)

3.2 什么是锁（重点）

在 Java 中，**锁（Lock）是一种用于控制并发访问的机制**，用于确保在**同一时间只有一个线程可以访问共享资源**。锁可用于保护临界区（一段代码，可能同时被多个线程访问的部分），以确保线程的安全性和数据的一致性。

Lock 接口

- Synchronized 关键字

synchronized 关键字是 Java 中最基本和最常用的锁机制。它可以应用于方法或代码块。当一个线程进入被 synchronized 关键字保护的方法或代码块时，它会获取锁，并且其他线程必须

等待锁释放后才能进入。- - synchronized 关键字会自动管理锁的获取和释放，使得编程更加简单，但灵活性较差。

synchronized 的两种用法

1. 直接在方法上声明，此时该方法称为同步方法，同步方法同时只能被一个线程执行（面试题）

   a. Synchronized 声明在成员方法上，锁的对象是 this

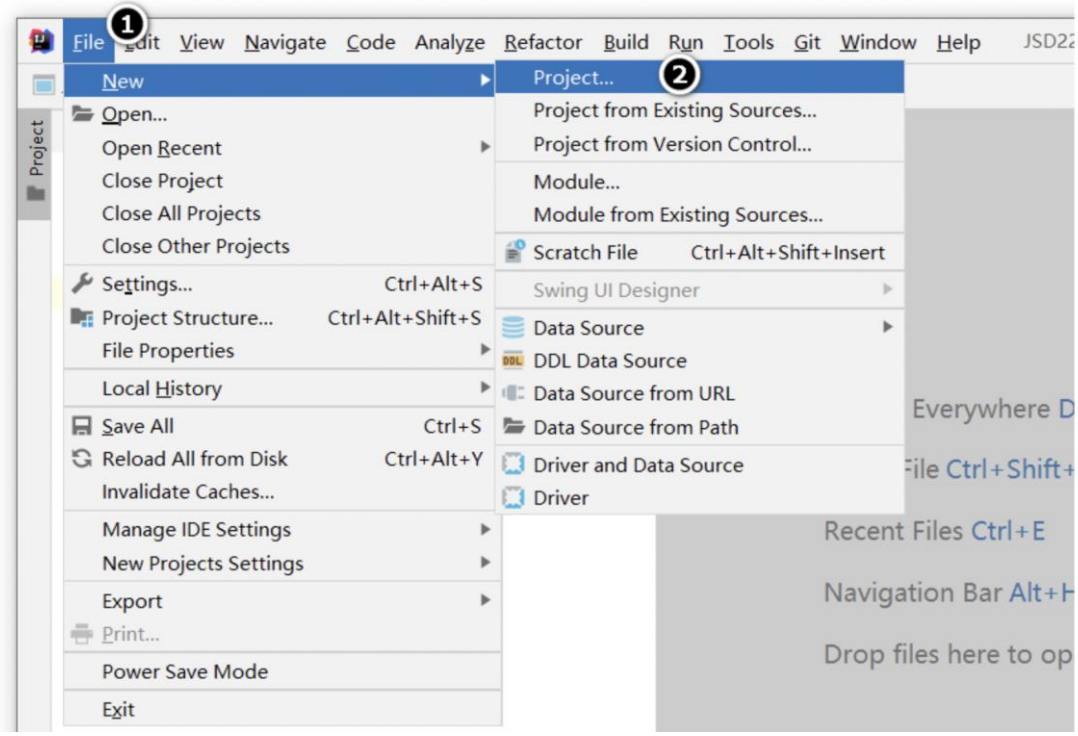
   b. Synchronized 声明在静态方法上，锁的对象是类对象

2.同步块，推荐使用。同步块可以更准确的控制需要同步执行的代码片段。

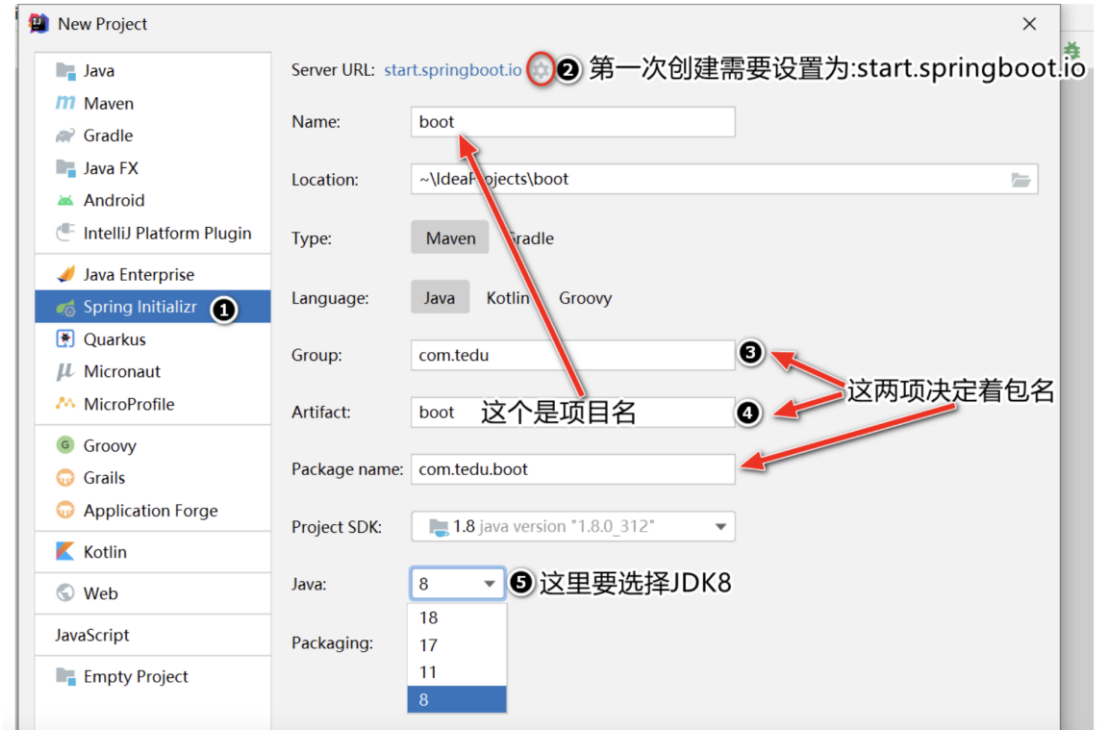
4. Spring Boot 项目

4.1 创建 Spring Boot 项目

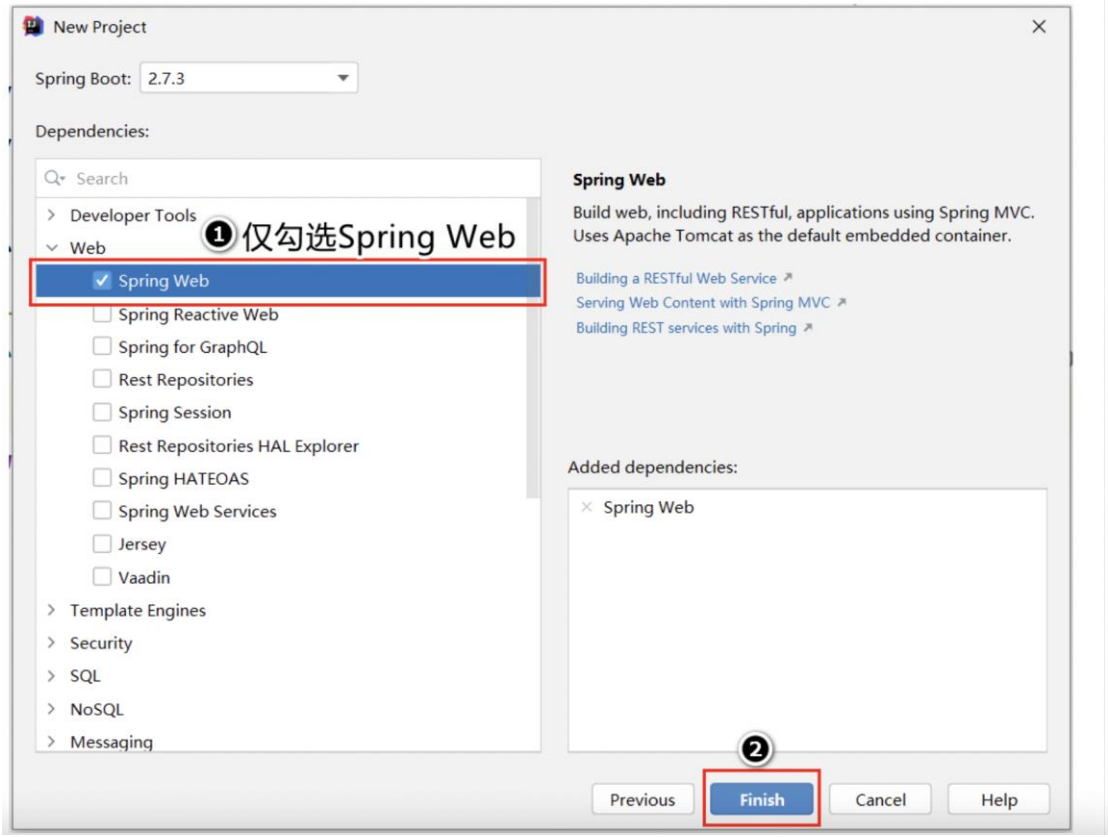
1. 第一步:新建一个项目



2. 第二步:选择 Spring Boot 项目，按照图上的步骤选择并输入对应内容，之后点击 next 进行下一步。注:第二小步可改用阿里云:https://start.aliyun.com

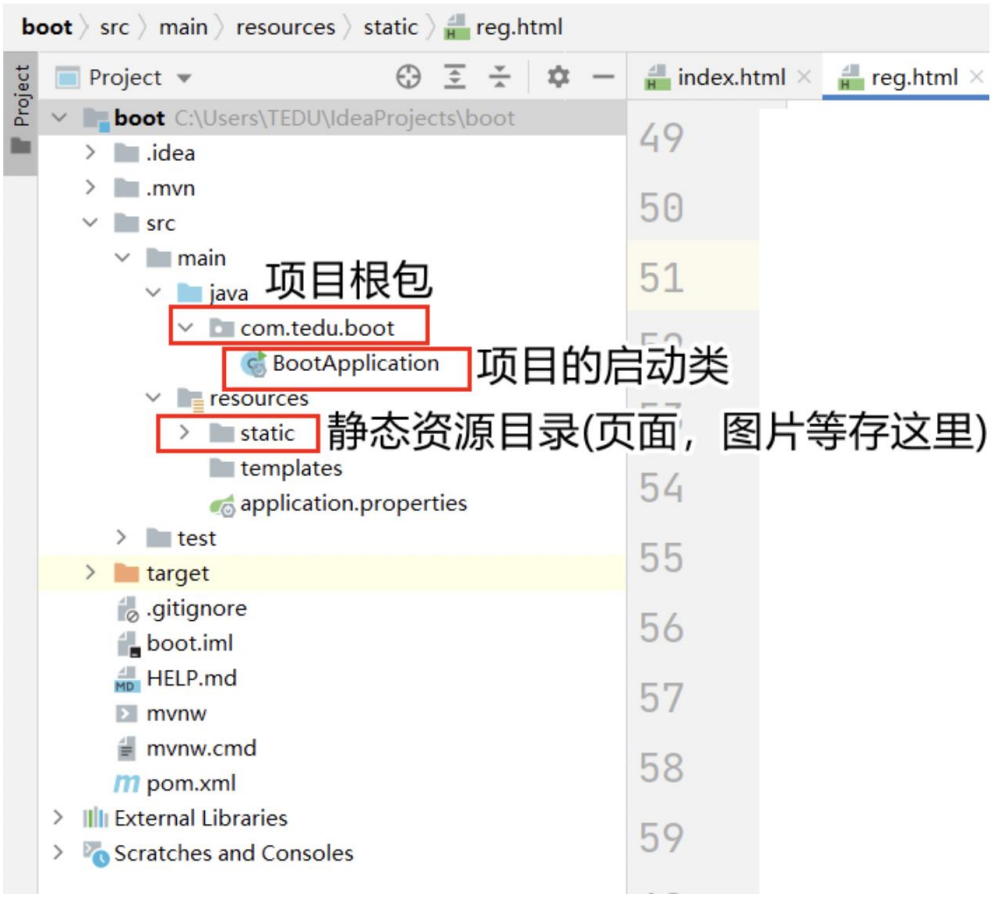


3. 第三步:勾选 Spring Web，并点击 finish 完成项目的创建

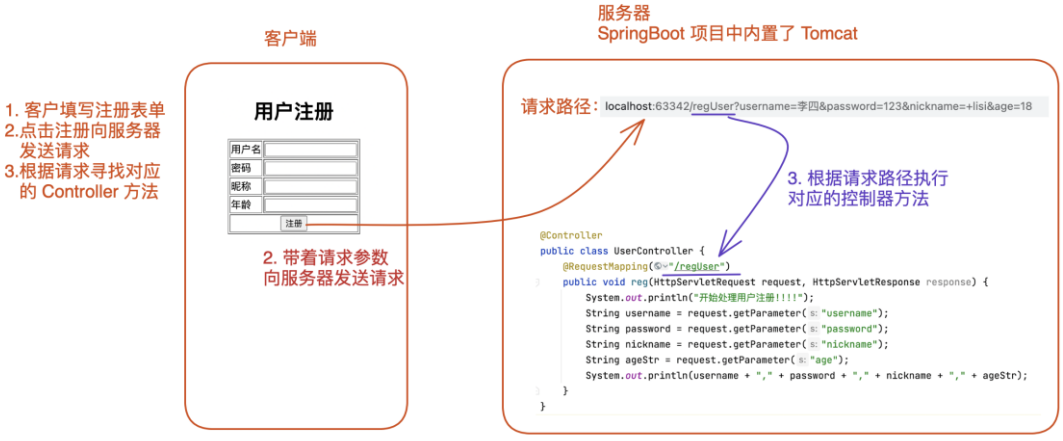


4.2 Spring Boot 项目结构

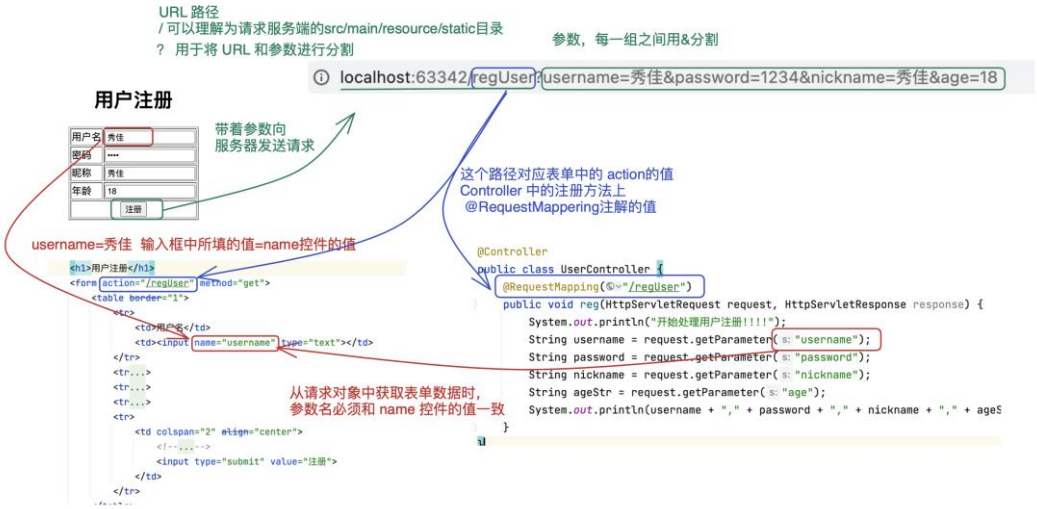
- **项目根包:**SpringBoot 项目要求我们将来定义的类和包必须放在这个包下
- **项目启动类:**SpringBoot 项目自动生成，里面包含 main 方法，用来启动
- **静态资源目录:**用来保存当前 web 应用(我们的网站)中所有的静态资源(页面，图片和其他素材)

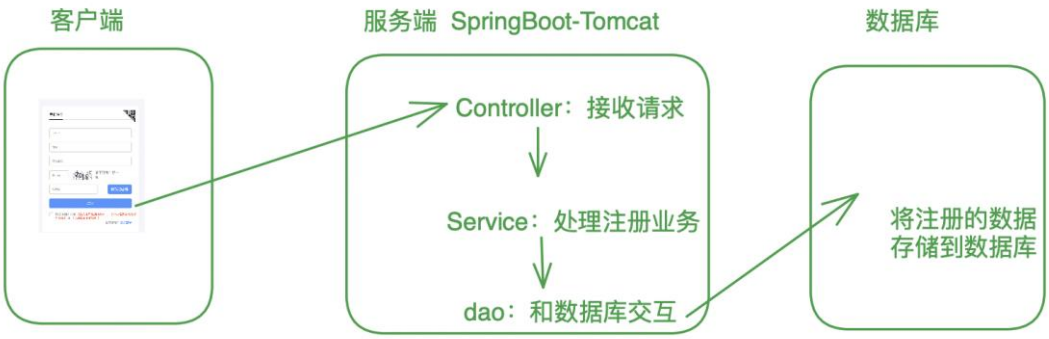


4.3 用户注册流程



4.4 URL 路径分析





Controller--直接和前端页面交互 （接收请求，发送响应）

Service--处理业务

dao---数据库交互

Java 基础面试题：

JAVA 基础高频题

线程相关面试题：

Java 面试题-分专题

Java 基础常识题：

JAVA 基础常识题

1. String，StringBuffer，StringBuilder
2. Object、正则，包装类
3. 读写文件--有文件 File ， IO 读写数据 字节流，字符流，对象流，缓冲流
4. 集合，ArrayList
5. 线程
6. 手写 SpringBoot