响应处理

1、使用默认内置视图解析器(ViewResolver)

```
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
   <property name="prefix" value="/WEB-INF/"/>
   <property name="suffix" value=".jsp"></property>
</bean>
```

2、使用视图控制器<view-controller>

如果我们有些请求只是想跳转页面，不需要来后台处理什么逻辑，我们无法在Action中写一个空方法来跳转，直接在中配置一个如下的视图跳转控制器即可(不经过Action，直接跳转页面)

```
<mvc:view-controller path="/" view-name="index" />
```

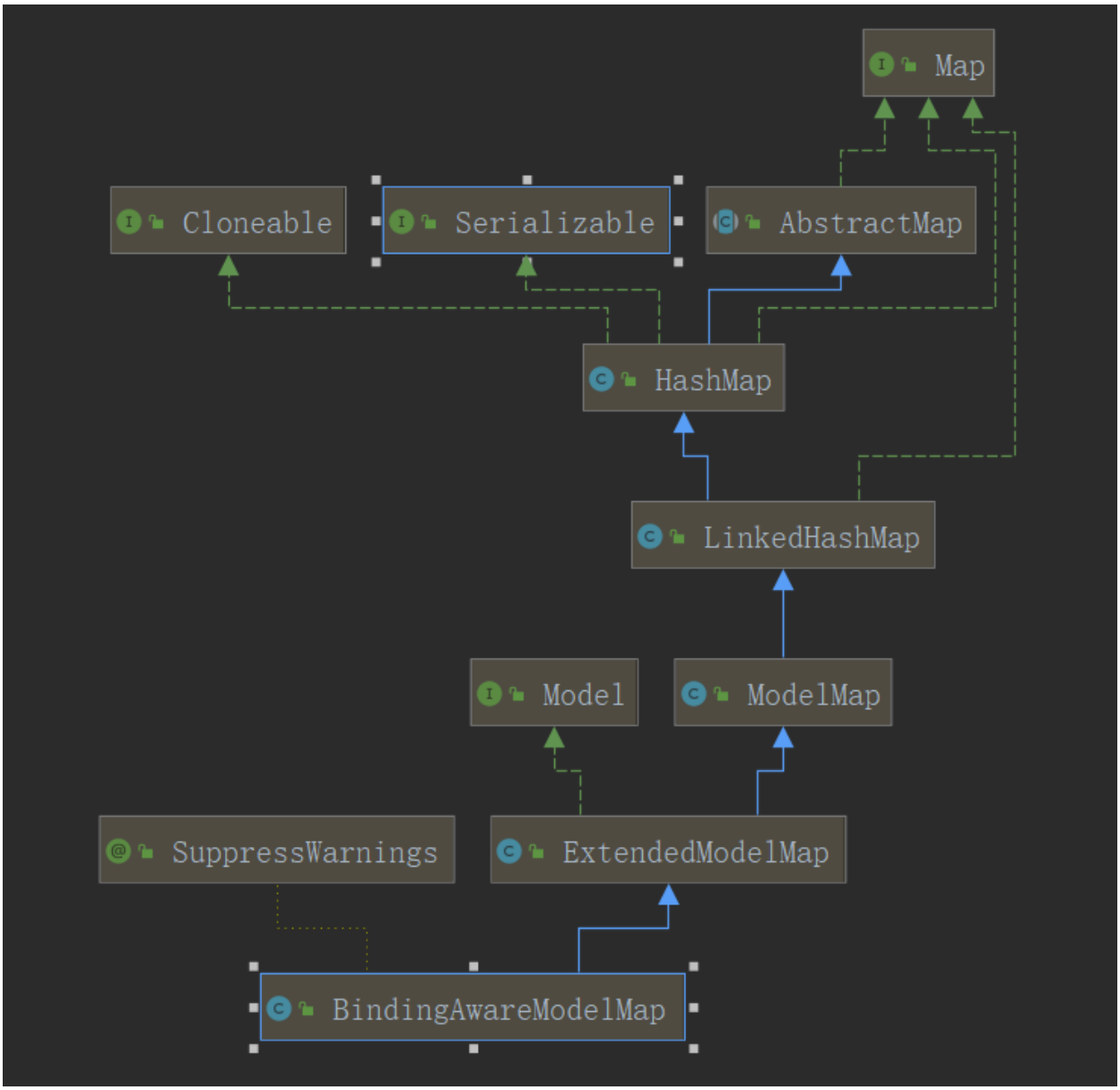
3、使用Model，Map，ModelMap传输数据到页面

在刚开始的helloworld项目中，我们传递了参数回到我们页面，但是后续的操作都只是接受用户的请求，那么在SpringMVC中除了可以使用原生servlet的对象传递数据之外，还有什么其他的方式呢?
可以在方法的参数上传入Model，ModelMap,Map类型，此时都能够将数据传送回页面
OutputController.java

```
package cn.tulingxueyuan.controller;
```

```java

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.RequestMapping;

import java.util.Map;

@Controller
public class OutputController {

    @RequestMapping("output1")
    public String output1(Model model){
        model.addAttribute("msg","hello,Springmvc");
        return "output";
    }

    @RequestMapping("output2")
    public String output2(ModelMap model){
        model.addAttribute("msg","hello,Springmvc");
        return "output";
    }

    @RequestMapping("output3")
    public String output1(Map map){
        map.put("msg","hello,Springmvc");
        return "output";
    }
}
```

当使用此方式进行设置之后，会发现所有的参数值都设置到了request作用域中，那么这三个对象是什么关系呢?



4、使用ModelAndView对象传输数据到页面

```
1  package cn.tulingxueyuan.controller;
2
3  import org.springframework.stereotype.Controller;
4  import org.springframework.web.bind.annotation.RequestMapping;
5  import org.springframework.web.servlet.ModelAndView;
6
7  @Controller
8  public class OutputController {
9
10     @RequestMapping("mv")
11     public ModelAndView mv(){
12         ModelAndView mv = new ModelAndView();
13         mv.setViewName("output");
```

```
        mv.addObject("msg","hello.modelAndView");
        return mv;
    }
}
```

发现当使用modelAndView对象的时候，返回值的类型也是此对象，可以将要跳转的页面设置成view的名称，来完成跳转的功能，同时数据也是放到request作用中。

```
ps:springmvc处理以上几种设置model的request域中的几种方式以外，
springmvc还会隐式的讲请求绑定参数自动设置到request域
```

5、使用session传输数据到页面

怎么往session设置属性?

1.通过servlet api的方式去读写session

1. 通过参数绑定的方式去获取**servlet api**
2. 通过自动注入的方式去获取**servlet api(推荐使用这种方式)**

```
@Autowired
private HttpSession session;
/**
 * 通过自动注入的方式去获取servlet api
 */
@RequestMapping("/autowired/session")
public String session02(){
    session.setAttribute("type","autowired-session");
    return "main";
}
```

2.通过springmvc提供的注解方式去读写session

1. @SessionAttributes

   用在类上面的，写入session的。

```
// 从model中获取指定的属性写入session中
// 底层会从model中去找一个叫做type的属性
// 找到了会将type设置一份到session中
// 这种方式是依赖model的
// 当前控制器下所有的处理方法 都会将model指定的属性写入session
```

2. @SessionAttribute

用在参数上面的，读取session的。

```
*  required 用来设置session中某个属性必须存在 ，  不存在则会报错：HTTP Status 400 - Missing s
*  model和session是互通的：session可以通过model中去获取写入指定的属性，  model也会从session中自
```

1. @SessionAttributes

用在类上面的，负责写入session

```
@Controller
// 通过model中指定的属性去写入到session,同时也会从session中写入指定的属性到model,
// 所以使用SessionAttributes的情况下 model和session是共同的
// 使用该方式设置session是依赖model
@SessionAttributes("type")
public class DTVController {
```

2. @SessionAttribute

用在参数上面的，负责读取session

默认指定的属性是必须要存在的，如果不存在则会报错，可以设置required =false 不需要必须存在，不存在默认绑定null

```

@RequestMapping("/getSession")
public String getSession(@SessionAttribute(value="type",required = false) String type){
    System.out.println(type);
    return "main";
}
```

6、使用@ModelAttribute来获取请求中的数据

常用的使用场景

1.写在方法上面

@ModelAttribute的方法会在当前处理器中所有的处理方法之前调用

1.通过@ModelAttribute来给全局变量赋值(不推荐)

2. 当我们调用执行全字段的更新数据库操作时，假如提供给用户的修改字段只有部分几个，这个时候就会造成其他字段更新丢失：

解决：

1.自己定制update语句， 只更新指定的那些字段

2.如果无法定制sql语句， 可以在更新之前进行查询， 怎么在更新之前查询？只能在springmvc 绑定请求参数之前查询， 利用@ModelAttribute就可以在参数绑定之前查询， 但是怎么将查询出来的对象和参数的对象进行合并？ springmvc具有该特性， 会将model中和参数名相同的属性拿出来进行合并，将参数中的新自动进行覆盖，没有的字段进行保留。这样就可以解决这个问题。
2.写在参数上面
可以省略，加上则会从model中获取一个指定的属性和参数进行合并，因为model和sessionAttribute具有共通的特性，所以如果session中有对应的属性也会进行合并

```
@ModelAttribute("user")
```

@ModelAttribute注解用于将方法的参数或者方法的返回值绑定到指定的模型属性上，并返回给web视图。首先来介绍一个业务场景，来帮助大家做理解，在实际工作中，有些时候我们在修改数据的时候可能只需要修改其中几个字段，而不是全部的属性字段都获取，那么当提交属性的时候，从form表单中获取的数据就有可能只包含了部分属性，此时再向数据库更新的时候，肯定会丢失属性，因为对象的封装是springmvc自动帮我们new的，所以此时需要先将从数据库获取的对象保存下来，当提交的时候不是new新的对象，而是在原来的对象上进行属性覆盖，此时就需要使用@ModelAttribute注解。
User.java

```
package cn.tulingxueyuan.bean;

public class User {
    private Integer id;
    private String name;
    private String password;
    private Integer age;

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

```

```
    public void setName(String name) {
        this.name = name;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public Integer getAge() {
        return age;
    }

    public void setAge(Integer age) {
        this.age = age;
    }

    @Override
    public String toString() {
        return "User{" +
                "id=" + id +
                ", name='" + name + '\'' +
                ", password='" + password + '\'' +
                ", age=" + age +
                '}';
    }
}
```

UserController.java

```
package cn.tulingxueyuan.controller;

import cn.tulingxueyuan.bean.User;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
```

```java

@Controller
public class UserController {

    Object o1 = null;
    Object o2 = null;
    Object o3 = null;

    @RequestMapping("update")
    public String update(@ModelAttribute("user") User user,Model model){
        System.out.println(user);
        o2 = model;
        //可以看到所有的model都是同一个对象
        System.out.println(o1==o2);
        //可以看到存储的user对象也是同一个
        System.out.println(user == o3);
        return "output";
    }

    @ModelAttribute
    public void MyModelAttribute(Model model){
        o1 = model;
        User user = new User();
        user.setId(1);
        user.setName("张三");
        user.setAge(12);
        user.setPassword("123");
        model.addAttribute("user",user);
        System.out.println("modelAttribute:"+user);
        o3 = user;
    }
}
```

index.jsp

```jsp
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
 <head>
   <title>$Title$</title>
 </head>
```

```
 <body>
 <form action="update" method="post">
   <input type="hidden" value="1" name="id">
  姓名：张三<br>
  密码：<input type="text" name="password"><br>
  年龄：<input type="text" name="age"><br>
   <input type="submit" value="提交">
 </form>
 </body>
</html>
```

其实在使用的时候可以简化写法，也就是说，在方法的参数上不加@ModelAttribute也不会有问题

```
    @RequestMapping("update")
    public String update(User user,Model model){
        System.out.println(user);
        o2 = model;
        //可以看到所有的model都是同一个对象
        System.out.println(o1==o2);
        //可以看到存储的user对象也是同一个
        System.out.println(user == o3);
        return "output";
    }
```

如果添加的@ModelAttribute（“”）属性的值不对，那么也是获取不到值的。同时可以添加@SessionAttributes属性，但是注意，如果没有设置值的话，会报错

```
package cn.tulingxueyuan.controller;

import cn.tulingxueyuan.bean.User;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.SessionAttributes;

@Controller
@SessionAttributes("u")
public class UserController {

```

```
    Object o1 = null;
    Object o2 = null;
    Object o3 = null;

    @RequestMapping("update")
    public String update(@ModelAttribute("u") User user,Model model){
        System.out.println(user);
        o2 = model;
        //可以看到所有的model都是同一个对象
        System.out.println(o1==o2);
        //可以看到存储的user对象也是同一个
        System.out.println(user == o3);
        return "output";
    }

    @ModelAttribute
    public void MyModelAttribute(Model model){
        o1 = model;
        User user = new User();
        user.setId(1);
        user.setName("张三");
        user.setAge(12);
        user.setPassword("123");
        model.addAttribute("user",user);
        System.out.println("modelAttribute:"+user);
        o3 = user;
    }
}
```

注意：ModelAttribute除了可以使用设置值到model中之外，还可以利用返回值。

```
package cn.tulingxueyuan.controller;

import cn.tulingxueyuan.bean.User;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.SessionAttributes;
```

```java

@Controller
public class UserController {

    Object o1 = null;
    Object o2 = null;
    Object o3 = null;

    @RequestMapping("update")
    public String update(@ModelAttribute("u") User user,Model model){
        System.out.println(user);
        o2 = model;
        //可以看到所有的model都是同一个对象
        System.out.println(o1==o2);
        //可以看到存储的user对象也是同一个
        System.out.println(user == o3);
        return "output";
    }

    @ModelAttribute("u")
    public User MyModelAttribute(Model model){
        o1 = model;
        User user = new User();
        user.setId(1);
        user.setName("张三");
        user.setAge(12);
        user.setPassword("123");
//        model.addAttribute("user",user);
        System.out.println("modelAttribute:"+user);
        o3 = user;
        return user;
    }
}
```

总结：通过刚刚的给参数赋值，大家应该能够发现，当给方法中的参数设置值的时候，如果添加了@ModelAttribute注解，那么在查找值的时候，是遵循以下方式：

1、方法的参数使用参数的类型首字母小写，或者使用@ModelAttribute("")的值

2、先看之前是否在model中设置过该属性值，如果设置过就直接获取

3、看@SessionAttributes注解标注类中的方法是否给session中赋值，如果有的话，也是直接获取，没有报异常

3种方式的获取servlet--api的线程安全问题：

线程不安全=并发问题： 同一个时间，多个线程，**同时对共享数据/变量/资源 进行读写操作**，就会产生并发问题（脏读、幻影读...）

1.通过参数绑定的方式

是线程安全的，因为参数绑定的方式变量是方法级别的，所以每次请求方法都会在内存中开辟自己独立的空间。

2.通过@AutoWired自动注入的方式

是线程安全的，特殊，虽然他是共享变量（单例类级别的变量），但是**springmvc底层 通过ThreadLocal来存储的servlet api**,所以通过自动注入进来的servlet api是线程安全的。

3.通过@ModelAtrribute的方式

不是线程安全的，因为他是共享变量

7、使用forward实现页面转发

在发送请求的时候，可以通过forward:来实现转发的功能：

```
1  package cn.tulingxueyuan.controller;
2
3  import org.springframework.stereotype.Controller;
4  import org.springframework.web.bind.annotation.RequestMapping;
5
6  @Controller
7  public class ForWardController {
8
9      /**
10      * 当使用转发的时候可以添加前缀forward:index.jsp,此时是不会经过视图解析器的，所以要添加完整的
11      *
12      * forward:也可以由一个请求跳转到另外一个请求
13      *
14      * @return
15      */
16     @RequestMapping("/forward01")
```

```
    public String forward(){
        System.out.println("1");
        return "forward:/index.jsp";
    }


    @RequestMapping("/forward02")
    public String forward2(){
        System.out.println("2");
        return "forward:/forward01";
    }
}
```

8、使用redirect来实现重定向

```
package cn.tulingxueyuan.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class RedirectController {


    /**
     * redirect :重定向的路径
     *      相当于 response.sendRedirect("index.jsp")
     *      跟视图解析器无关
     * @return
     */
    @RequestMapping("redirect")
    public String redirect(){
        System.out.println("redirect");
        return "redirect:/index.jsp";
    }

    @RequestMapping("/redirect2")
    public String redirect2(){
        System.out.println("redirect2");
        return "redirect:/redirect";
```

```
26     }
27 }
```

在javaweb的时候大家应该都接触过重定向和转发的区别，下面再详细说一下：

转发：

> 由服务器的页面进行跳转，不需要客户端重新发送请求：

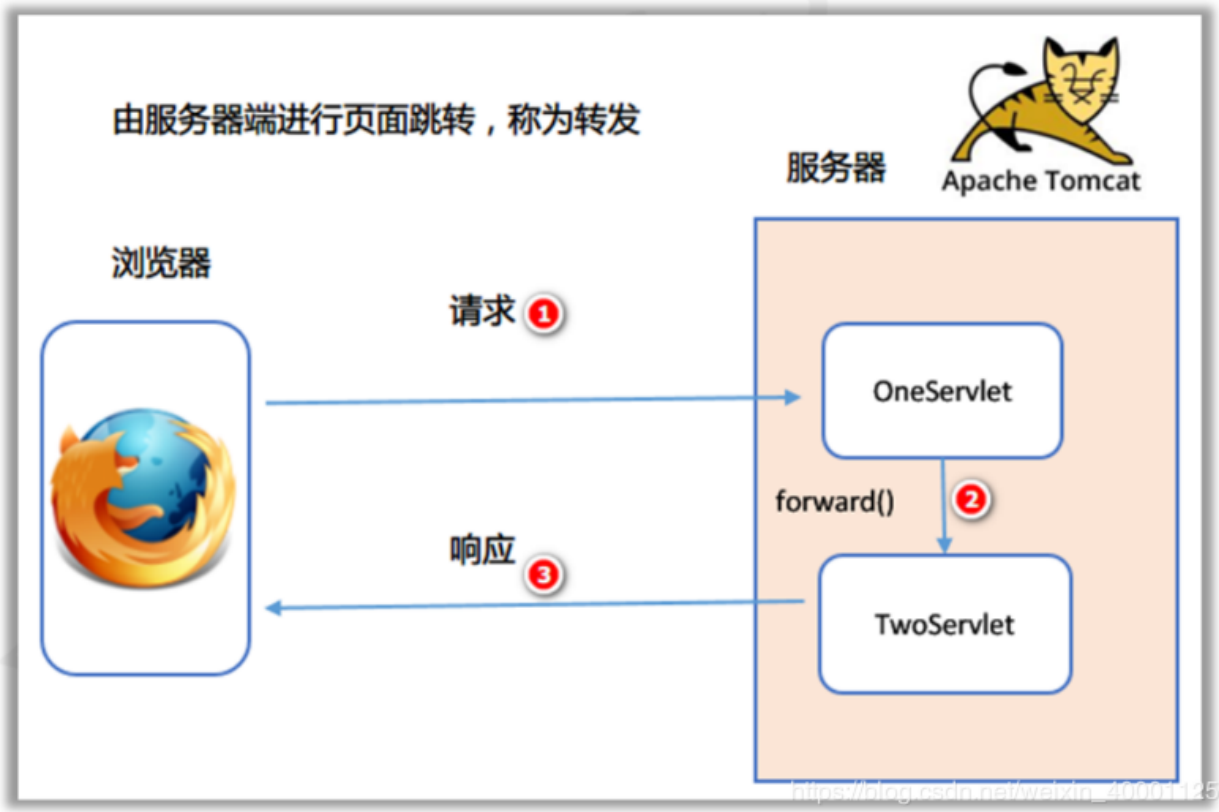
特点如下：

1、地址栏的请求不会发生变化，显示的还是第一次请求的地址

2、请求的次数，有且仅有一次请求

3、请求域中的数据不会丢失

4、根目录：localhost:8080/项目地址/,包含了项目的访问地址



重定向：

在浏览器端进行页面的跳转，需要发送两次请求（第一次是人为的，第二次是自动的）

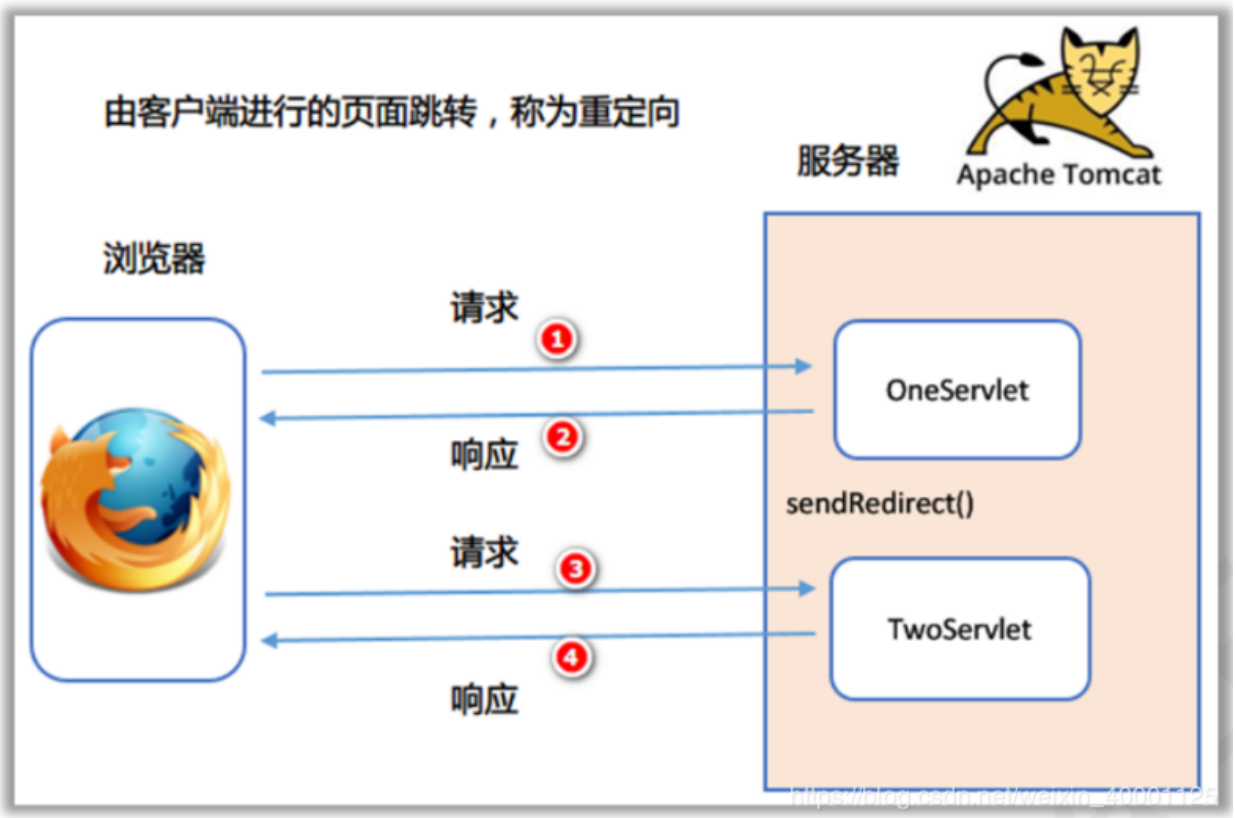
特点如下：

1、地址栏的地址发生变化，显示最新发送请求的地址

2、请求次数：2次

3、请求域中的数据会丢失，因为是不同的请求

4、根目录：localhost:8080/ 不包含项目的名称



对比：

区别	转发forward()	重定向sendRedirect()
根目录 / (在springmvc中无论转发还是重定向都会包含项目名)	包含项目访问地址	没有项目访问地址
地址栏	不会发生变化	会发生变化
哪里跳转	服务器端进行的跳转	浏览器端进行的跳转
request域中数据	不会丢失	会丢失

面试题：

springmvc 控制器是不是单例的？如果是单例的会出现什么问题？怎么解决？