Servlet3.0以上在Spring MVC里实现WebApplicationInitializer接口便可以实现等同于web.xml的配置

MvcConfig类

使用注解

@Configuraion

@EnableWebMvc,会开启一些默认配置,如一些ViewResolver或者MessageConverter等

1.视图解析器类

Web配置类:WebInitializer implements WebApplicationInitializer

1.Spring MVC常用注解

1. @Controller

   在Spring MVC声明控制器Bean的时候,只能使用@Controller

2. @RequestMapping

   映射Web请求

3. @ResponseBody

   支持将返回值放在response体内,而不是返回一个页面

4. @RequestBody

   允许request请求的参数在request体中,而不是在直接链接在地址后面

5. @PathVariable

   接收路径参数

6. @RestController

   组合注解,组合了@Controller和@ResponseBody

1.1 示例

1.传值类

添加jackson以及相关依赖

相同版本的:jackson-core,jackson-databind,jackson-annotation

jackson对对象和json做转换时一定需要空构造函数

2.Spring MVC基本配置

Spring MVC的定制配置需要我们的配置类继承一个WebMvcConfigurerAdapter类,并在此类使用@EnableWebMvc注解,开启对Spring MVC的支持

2.1 静态资源映射

重写addResourceHandlers方法

#addResourceLocations指的是文件放置的目录,addResourceHandler指的是对外暴露的访问路径

registry.addResourceHandler("/assets/**").addResourceLocations("classpath:/assets/");

2.2 拦截器配置

拦截器实现对每一个请求处理前后进行相关的业务处理,类似于Servlet的Filter

可让普通的Bean实现HandlerInterceptor接口或者继承HandlerInterceptorAdapter类来实现自定义拦截器.

通过重写WebMvcConfigurerAdapter的addInterceptors方法来注册自定义的拦截器

重写preHandler方法,在请求发生前执行
重写postHandler方法,在请求完成后执行

2.3 @ControllerAdvice

通过@ControllerAdvice,我们可以将对于控制器的全局配置放在同一个位置,注解@Controller的类的方法可使用@ExceptionHandler,@InitBinder,@ModelAttribute注解到方法上,对所有注解了@RequestMapping的控制器内方法有效

@ExceptionHandler:用于全局处理控制器里的异常

@InitBinder:用来设置WebDataBinder,WebDataBinder用来自动绑定前台请求参数到Model中

@ModelAttribute:@ModelAttribute本来的作用是绑定键值对到Model里,此处是让全局的@RequestMapping都能获得在此处设置的键值对

2.4 其他配置

1.快捷ViewController

重写addViewControllers方法

registry.addViewController("/index").setViewName("/index")

2.路径匹配参数配置

在Spring MVC中,路径参数如果带"."的话,"."后面的值将被忽略,例如:

http://localhost:8080/ch1/anno/pathvar/xx.yy

此时."."后面的yy被忽略

通过重写configurePathMatch方法可不忽略"."后面的参数:

configurer.setUseSuffixPatternMatch(false)

更多参见WebMvcCongifurerAdapter类

3. Spring MVC的高级配置

3.1 文件上传配置

Spring MVC通过配置一个MultipartResolver来上传文件

在Spring控制器中,通过MultipartFile file来接收文件,通过MultipartFile[] files接收多个文件上传.

文件上传依赖

```
@Bean
public MultipartResolver multipartResolver(){
   CommonsMultipartResolver multipartResolver = new
CommonsMultipartResolver();
   multipartResolver.setMaxUploadSize(1000000);
   return multipartResolver;
}
```

commons-fileupload
commons-io

MultipartResolver配置的Bean名称必须是multipartResolver

3.2 自定义HttpMessageConverter

自定义HttpMessageConverter需要继承AbstractHttpMessageConverter类

在Spring mvc配置类中扩展消息转换 extendMessageConverters

3.3 服务器端推送技术

服务器端推送技术以外,还有一个另外的双向通信的技术WebSocket

(1),SSE(Server Send Event 服务端发送事件),需要新式浏览器支持

(2). 基于Servlet3.0+的异步方法特性,跨浏览器的

1.SSE

服务器端:

```
@RequestMapping(value="/push",produces="text/event-stream")
```

浏览器端:

```
if(!!window.EventSource){ //EventSource是SSE的客户端
   var source = new EventSource("push");
   s='';
   source.addEventListener('message',function(e){//添加SSE客户端监听,在此获得服务器
推送的消息
      s += e.data + "<br/>";
      $("#msgFromPush").html(s);
   })

   source.addEventListener("open",function(e){
      console.log("链接打开.")
   },false);

   source.addEventListener('error',function(e){
      if (e.readyState == EventSource.CLOSED) {
         console.log("链接关闭")
      } else {
         console.log(e.readyState);
      }
   },false)
```

```
} else {
    console.log("浏览器不支持SSE")
}
```

2.Servlet3.0+异步方法处理

开启异步方法支持:

在实现WebApplicationInitializer接口的类中

```
ServletRegistration.Dynamic servlet = servletContext.addServlet("dispatcher", new
DispatcherServlet(context));
servlet.addMapping("/");
servlet.setAsyncSupported(true); #开启异步方法支持
servlet.setLoadOnStartup(1);
```

异步任务的实现方式是通过控制器从另一个线程返回一个DeferredResult

后端controller:

```
@Controller
public class AysncController {
    @Autowired
    PushService pushService;

    @RequestMapping("/defer")
    @ResponseBody
    public DeferredResult<String> deferredCall(){
        return pushService.getAsyncUpdate();
    }
}
```

后端定时任务: 需要@EnableScheduling

```
@Service
public class PushService {
    private DeferredResult<String> deferredResult;

    public DeferredResult<String> getAsyncUpdate(){
        deferredResult = new DeferredResult<String>();
        return deferredResult;
    }

    @Scheduled(fixedDelay = 5000)
    public void refresh(){
        if (deferredResult != null) {
            deferredResult.setResult(new Long(System.currentTimeMillis()).toString());
        }
    }
}
```

前端请求:

```
deferred();
function deferred(){
    $.get('defer',function(e){
```

```
        console.log(e);
        deferred()
    })
}
```

4 Spring MVC的测试

测试驱动开发(TDD):

参见git