SpringBoot高级篇

一、SpringBoot基础篇

1.SpringBoot初探

SpringBoot的初衷**简化配置**

2.SpringBoot项目的构建方式

2.1 通过官网自动生成

https://start.spring.io/ 快速生成

2.2 IDE 在线模板生成

本质上和上面是一样的，只是简化了我们的操作

2.3 IDE通过maven项目构建

1.创建一个独立的web项目

2.引入对应依赖

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.1.15.RELEASE</version>
    <relativePath/> <!-- lookup parent from repository -->
</parent>

<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
</dependencies>
```

3.添加对应的启动器

```
@SpringBootApplication
public class GpSpringBootApplication {

    public static void main(String[] args) {
        SpringApplication.run(GpSpringBootApplication.class,args);
    }
}
```

4.启动启动器的主方法

测试即可

3.SpringBoot中的常规配置

3.1 入口类和相关注解

```
@SpringBootApplication
public class GpSpringbootDemo02Application {

    public static void main(String[] args) {
        // Spring IoC 容器的初始化
        ApplicationContext ac =
SpringApplication.run(GpSpringbootDemo02Application.class, args);
    }
}
```

main方法：

其实完成的就是一个SpringIoC容器的初始化操作

@SpringBootApplication注解

1.在IoC初始化的时候会加载该注解

2.是一个组合注解

```
@Target({ElementType.TYPE}) // 注解可以写在哪些地方
@Retention(RetentionPolicy.RUNTIME) // 该注解的作用域  RESOURCES CLASS RUNTIME
@Documented // 该注解会被API抽取
@Inherited  // 可继承
// 以上四个是Java中提供的元注解
@SpringBootConfiguration // 本质上就是一个Configuration注解
@EnableAutoConfiguration // 自动装配的注解
@ComponentScan( // 扫描 会自动扫描 @SpringBootApplication所在的类的同级包
(com.gupaoedu)以及子包中的Bean，所有一般我们建议将入口类放置在 groupId+artifcatID的组合包
下
    excludeFilters = {@Filter(
    type = FilterType.CUSTOM,
    classes = {TypeExcludeFilter.class}
), @Filter(
    type = FilterType.CUSTOM,
    classes = {AutoConfigurationExcludeFilter.class}
)}
)
```

3.2 Banner

http://patorjk.com/software/taag

在resources目录下创建一个 `banner.txt` 文件

关闭Banner

```
public static void main(String[] args) {
    // Spring IoC 容器的初始化
    //ApplicationContext ac =
SpringApplication.run(GpSpringbootDemo02Application.class, args);
    SpringApplication springApplication = new
SpringApplication(GpSpringbootDemo02Application.class);
    springApplication.setBannerMode(Banner.Mode.OFF);// 关闭Banner
    springApplication.run(args);
}
```

3.3 常规配置

在SpringBoot中给我们提供的有两个配置文件 applicationContext.properties,applicationContext.yml 作用是一样的，一个项目中只需要其中的一个就可以了。

Tomcat配置修改

```
server.port=8082
server.servlet.context-path=/springboot
```

自定义的属性

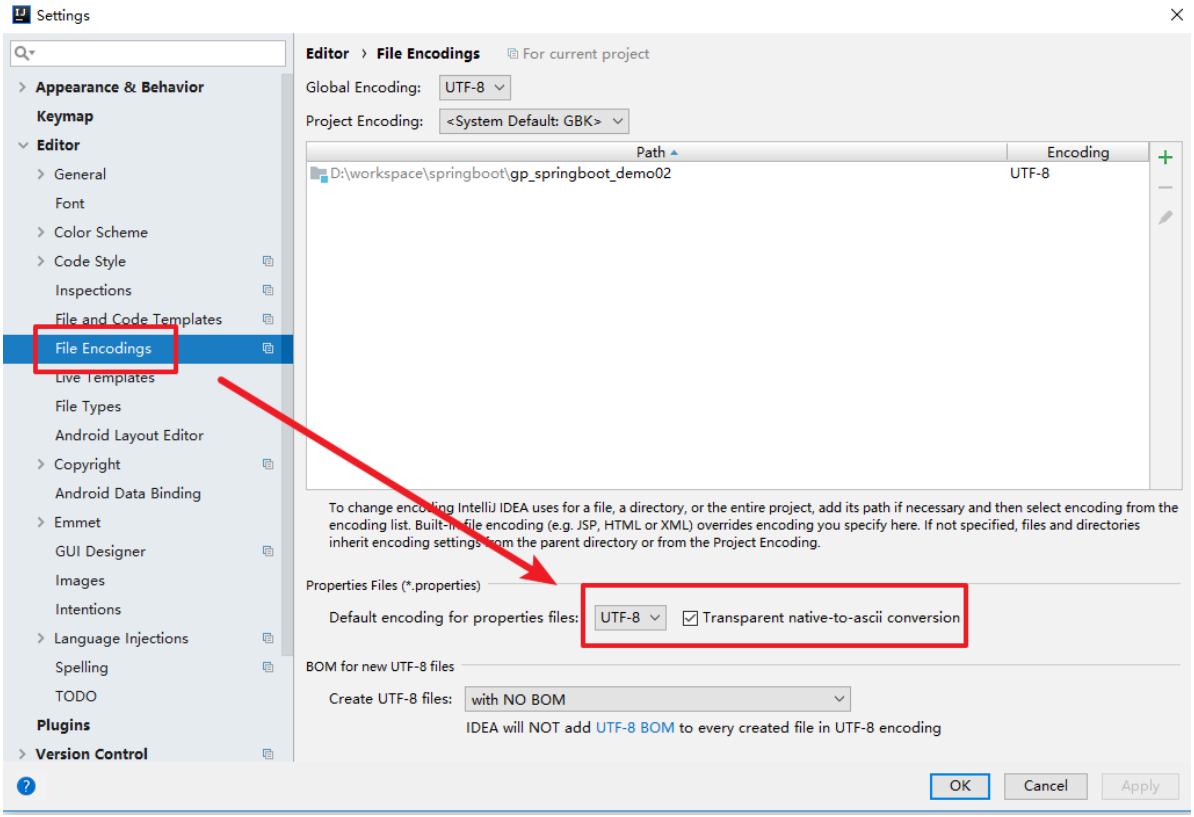
```
# 自定义的配置信息
user.username=bobo
user.age=18
user.address=湖南长沙
```

获取

```
@Value("${user.username}")
private String userName;

@Value("${user.age}")
private Integer age;
@Value("${user.address}")
private String address;
```

中文乱码问题

```
server.tomcat.uri-encoding=UTF-8
spring.http.encoding.charset=UTF-8
spring.http.encoding.enabled=true
spring.http.encoding.force=true
spring.messages.encoding=UTF-8
```



类型安全配置

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-configuration-processor</artifactId>
    <optional>true</optional>
</dependency>
```

```
package com.gupaoedu.bean;

import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.stereotype.Component;

/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 * @Description: ${todo}
 * @date 2020/7/21 10:26
 */
@Component
// 属性文件中的属性和User对象中的成员变量映射
@ConfigurationProperties(prefix = "user")
```

```java
public class User {

    private String username;

    private Integer age;

    private String address;


    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public Integer getAge() {
        return age;
    }

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

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

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

3.4 Logback日志

SpringBoot内置的有Logback的依赖

直接在属性文件中简单配置

```
# logback的配置
logging.file=d:/log.log
logging.level.org.springframework.web=DEBUG
```

单独提供一个`logback.xml`

3.5 Profile

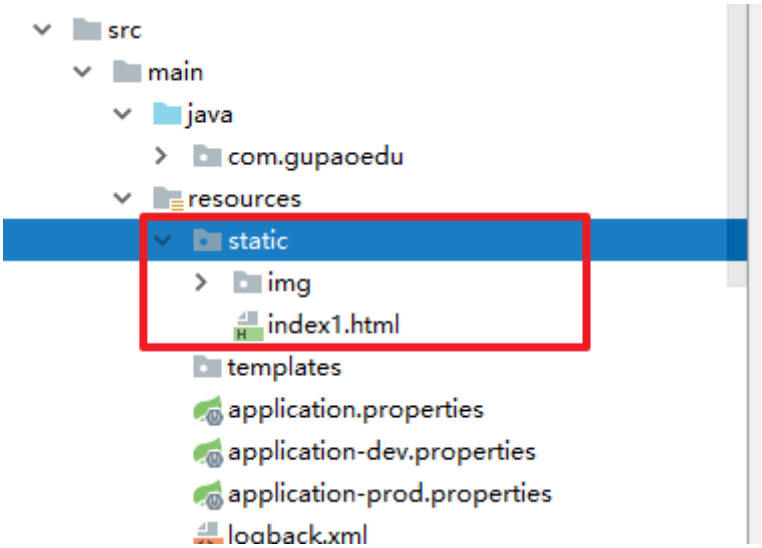
命名规则`application-xxx.properties`

```
spring.profiles.active=xxx # 指定对应的环境
```

4.SpringBoot中的静态资源

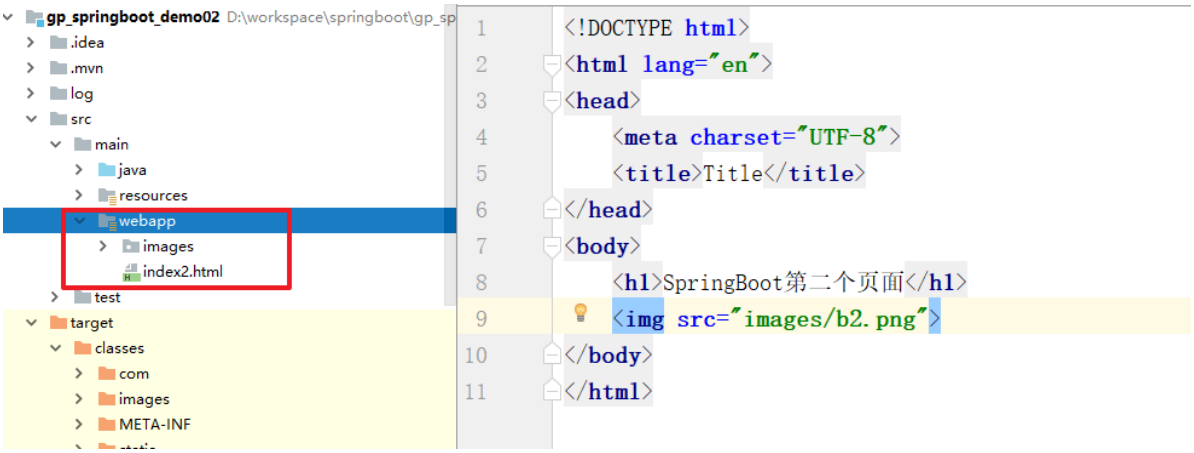
4.1 static目录

SpringBoot默认的存放静态资源的目录



4.2 webapp目录

在resources同级目录下创建一个webapp目录，该目录的类型必须是ResourcesRoot



4.3 自定义静态资源路径

自定义目录后，创建对应的相关资源，然后在属性文件中去覆盖静态资源的路径配置即可

```
# 表示所有的访问都经过静态资源路径
spring.webflux.static-path-pattern=/**

# 覆盖默认的配置，所有需要将默认的static public等这些路径将不能作为静态资源的访问
spring.resources.static-locations=classpath:/META-INF/resources/,classpath:/resources/,classpath:/static/,classpath:/custom
```

5.自动装配的原理

5.1 starter

名称	描述
spring-boot-starter-thymeleaf	使MVC Web applications 支持Thymeleaf
spring-boot-starter-data-couchbase	使用Couchbase 文件存储数据库、Spring Data Couchbase
spring-boot-starter-artemis	为JMS messaging使用Apache Artemis
spring-boot-starter-web-services	使用Spring Web Services
spring-boot-starter-mail	Java Mail、Spring email为邮件发送工具
spring-boot-starter-data-redis	通过Spring Data Redis 、Jedis client使用Redis键值存储数据库
spring-boot-starter-web	构建Web，包含RESTful风格框架SpringMVC和默认的嵌入式容器Tomcat
spring-boot-starter-activemq	为JMS使用Apache ActiveMQ
spring-boot-starter-data-elasticsearch	使用Elasticsearch、analytics engine、Spring Data Elasticsearch
spring-boot-starter-integration	使用Spring Integration
spring-boot-starter-test	测试 Spring Boot applications包含JUnit、 Hamcrest、Mockito
spring-boot-starter-jdbc	通过 Tomcat JDBC 连接池使用JDBC
spring-boot-starter-mobile	通过Spring Mobile构建Web应用
spring-boot-starter-validation	通过Hibernate Validator使用 Java Bean Validation
spring-boot-starter-hateoas	使用Spring MVC、Spring HATEOAS构建 hypermedia-based RESTful Web 应用
spring-boot-starter-jersey	通过 JAX-RS、Jersey构建 RESTful web applications；spring-boot-starter-web的另一替代方案
spring-boot-starter-data-neo4j	使用Neo4j图形数据库、Spring Data Neo4j
spring-boot-starter-websocket	使用Spring WebSocket构建 WebSocket 应用
spring-boot-starter-aop	通过Spring AOP、AspectJ面向切面编程
spring-boot-starter-amqp	使用Spring AMQP、Rabbit MQ

名称	描述
spring-boot-starter-data-cassandra	使用Cassandra分布式数据库、Spring Data Cassandra
spring-boot-starter-social-facebook	使用 Spring Social Facebook
spring-boot-starter-jta-atomikos	为 JTA 使用 Atomikos
spring-boot-starter-security	使用 Spring Security
spring-boot-starter-mustache	使MVC Web applications 支持Mustache
spring-boot-starter-data-jpa	通过 Hibernate 使用 Spring Data JPA （Spring-data-jpa依赖于Hibernate）
spring-boot-starter	Core starter,包括 自动配置支持、 logging and YAML
spring-boot-starter-groovy-templates	使MVC Web applications 支持Groovy Templates
spring-boot-starter-freemarker	使MVC Web applications 支持 FreeMarker
spring-boot-starter-batch	使用Spring Batch
spring-boot-starter-social-linkedin	使用Spring Social LinkedIn
spring-boot-starter-cache	使用 Spring caching 支持
spring-boot-starter-data-solr	通过 Spring Data Solr 使用 Apache Solr
spring-boot-starter-data-mongodb	使用 MongoDB 文件存储数据库、Spring Data MongoDB
spring-boot-starter-jooq	使用JOOQ链接SQL数据库；spring-boot-starter-data-jpa、spring-boot-starter-jdbc的另一替代方案
spring-boot-starter-jta-narayana	Spring Boot Narayana JTA Starter
spring-boot-starter-cloud-connectors	用连接简化的 Spring Cloud 连接器进行云服务就像Cloud Foundry、Heroku那样
spring-boot-starter-jta-bitronix	为JTA transactions 使用 Bitronix
spring-boot-starter-social-twitter	使用 Spring Social Twitter

名称	描述
spring-boot-starter-data-rest	使用Spring Data REST 以 REST 方式暴露 Spring Data repositories
spring-boot-starter-actuator	使用Spring Boot Actuator 的 production-ready 功能来帮助你监视和管理应用
spring-boot-starter-undertow	使用 Undertow 作为嵌入式服务容器；spring-boot-starter-tomcat 的另一替代方案
spring-boot-starter-jetty	使用 Jetty 作为嵌入式服务容器；spring-boot-starter-tomcat的另一替代方案
spring-boot-starter-logging	为 logging 使用Logback.默认 logging starter
spring-boot-starter-tomcat	使用 Tomcat 作为嵌入式服务容器；作为默认嵌入式服务容器被 spring-boot-starter-web使用
spring-boot-starter-log4j2	使用Log4j2记录日志；spring-boot-starter-logging的另一替代方案

5.2 自动装配

@EnableAutoConfiguration

```
@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Inherited
@AutoConfigurationPackage
@Import({AutoConfigurationImportSelector.class})
public @interface EnableAutoConfiguration {
    String ENABLED_OVERRIDE_PROPERTY = "spring.boot.enableautoconfiguration";

    Class<?>[] exclude() default {};

    String[] excludeName() default {};
}
```

通过@EnableAutoConfiguration注解发现。其本身就是一个组合注解，有一个注解我们必须要弄清楚`@Import`注解

@Import

在Spring中我们将类型交给SpringIoC管理的方式有哪些?

> 1.基于xml配置文件
>
> 2.基于xml配置文件@Component
>
> 3.基于Java配置类【@Configuration】 @Bean

> 4.基于Java配置类+@ComponentScan+@Component
>
> 5.FactoryBean接口【getObject()】
>
> 6.@Import注解

第一种使用方式

静态使用方式

```
@Configuration
@Import(UserService.class)
public class JavaConfig {

    /*@Bean
    public UserService getUserSerivce(){
        return new UserService();
    }*/
}
```

在@Import注解中直接指定要添加的类型

缺点：直接在@Import中写死要注入的类型，不太灵活

第二种使用方式

ImportSelector接口

```
/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 * @Description: ${todo}
 * @date 2020/7/21 15:29
 */
public class GpImportSelector implements ImportSelector {
    /**
     *  动态获取IoC要加载的类型
     * @param annotationMetadata 注解的元数据
     * @return
     *     IoC要加载的类型的数组
     */
    @Override
    public String[] selectImports(AnnotationMetadata annotationMetadata) {
        // 根据不同的业务逻辑 实现动态添加IoC加载的类型
        /*if (){

        }*/
        return new String[]
{LoggerService.class.getName(),CacheService.class.getName()};
    }
}
```

```
@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Inherited
@Import(GpImportSelector.class)
public @interface EnableGpImport {
}
```

是将@Import注解中添加的 ImportSelector的实现类中的`selectImports`这个方法返回的字符串数组加载到IoC容器中

第三种实现方式

实现ImportBeanDefinitionRegistrar接口，其实和第二种方式很类似，都是在源码设计层面用的比较多。

```
/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 * @Description: ${todo}
 * @date 2020/7/21 15:37
 */
public class GpImportBeanDefinition implements ImportBeanDefinitionRegistrar {
    /**
     *  提供了一个beanDefinition的注册器，我直接把需要IoC加载的类型注册到容器中去
     * @param annotationMetadata
     * @param beanDefinitionRegistry beanDefinition的注册器
     */
    @Override
    public void registerBeanDefinitions(AnnotationMetadata annotationMetadata,
BeanDefinitionRegistry beanDefinitionRegistry) {
        // 将我们需要添加的类型统一封装为RootBeanDefinition对象
        RootBeanDefinition cache = new RootBeanDefinition(CacheService.class);
        beanDefinitionRegistry.registerBeanDefinition("cache",cache);
        RootBeanDefinition logger = new RootBeanDefinition(LoggerService.class);
        beanDefinitionRegistry.registerBeanDefinition("logger",logger);
    }
}
```

原理分析

```
public String[] selectImports(AnnotationMetadata annotationMetadata) {
    if (!this.isEnabled(annotationMetadata)) {
        return NO_IMPORTS;
    } else {
        // 加载META-INF/spring-autoconfigure-metadata.properties
        AutoConfigurationMetadata autoConfigurationMetadata =
AutoConfigurationMetadataLoader.loadMetadata(this.beanClassLoader);
        AutoConfigurationImportSelector.AutoConfigurationEntry
autoConfigurationEntry =
this.getAutoConfigurationEntry(autoConfigurationMetadata, annotationMetadata);
        // 返回需要IoC加载的类型数组
        return
StringUtils.toStringArray(autoConfigurationEntry.getConfigurations());
    }
}
```

```
protected AutoConfigurationImportSelector.AutoConfigurationEntry
getAutoConfigurationEntry(AutoConfigurationMetadata autoConfigurationMetadata,
AnnotationMetadata annotationMetadata) {
    if (!this.isEnabled(annotationMetadata)) {
        return EMPTY_ENTRY;
    } else {
        AnnotationAttributes attributes =
this.getAttributes(annotationMetadata);
        // 获取候选的配置信息 META-INF/spring.factories 加载了很多的 类路径
        List<String> configurations =
this.getCandidateConfigurations(annotationMetadata, attributes);
        // 去掉重复的
        configurations = this.removeDuplicates(configurations);
        // 去掉要排除掉的类型
        Set<String> exclusions = this.getExclusions(annotationMetadata,
attributes);
        this.checkExcludedClasses(configurations, exclusions);
        configurations.removeAll(exclusions);
        // 过滤器
        configurations = this.filter(configurations, autoConfigurationMetadata);
        // 广播
        this.fireAutoConfigurationImportEvents(configurations, exclusions);
        return new
AutoConfigurationImportSelector.AutoConfigurationEntry(configurations,
exclusions);
    }
}
```

自动装配的原理：

1.在SpringBoot项目启动的时候，会加载SpringBootApplication这个注解

2.会解析@EnableAutoConfiguration注解

3.与之对应的解析@Import注解

4.执行ImportSelector接口的的实现

5.加载META-INF/spring-autoconfigure-metadata.properties中的注解元数据信息

6.加载META-INF/spring.factories各种类路径【第三方扩展也同样的会加载对应的文件 SPI扩展机制】

二、SpringBoot集成篇

1.SpringBoot整合Servlet

1.1 第一种方式

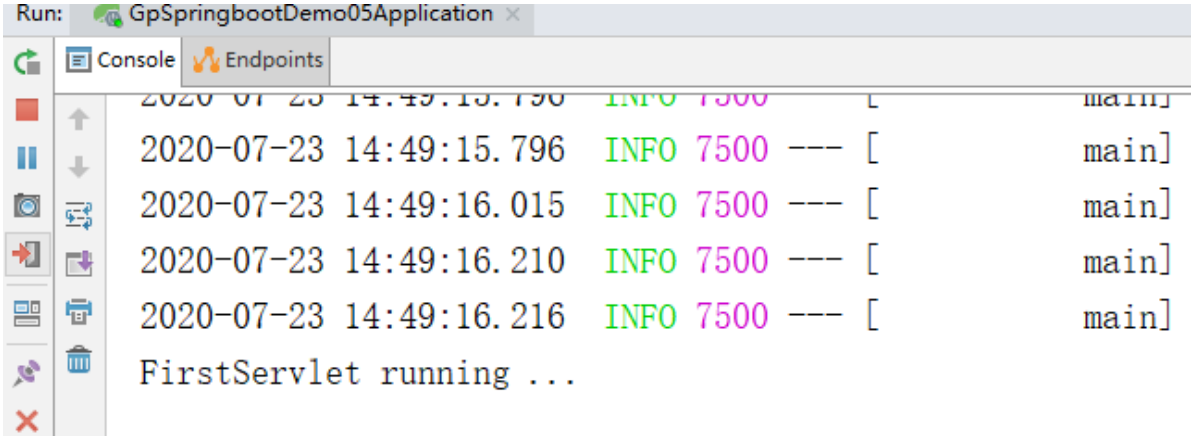
1.添加自定义的Servlet

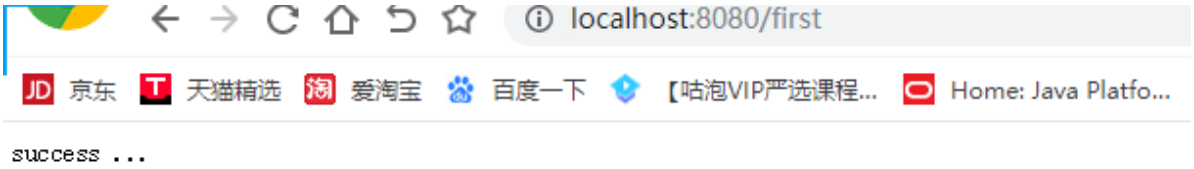
```
package com.gupaoedu.servlet;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;

/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 * @Description: ${todo}
 * @date 2020/7/23 14:45
 */
@WebServlet(name = "FirstServlet",urlPatterns = "/first")
public class FirstServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        System.out.println("FirstServlet running ... ");
        PrintWriter out = resp.getWriter();
        out.write("success ... ");
        out.flush();
        out.close();
    }
}
```

2.在启动类中添加扫描注解

```
@SpringBootApplication
// 在SpringBoot启动的时候会扫描@WebServlet注解
@ServletComponentScan()
public class GpSpringbootDemo05Application {

    public static void main(String[] args) {
        SpringApplication.run(GpSpringbootDemo05Application.class, args);
    }

}
```

3.启动测试





1.2 第二种方式

1.创建自定义的Servlet，不需要添加`@WebServlet`

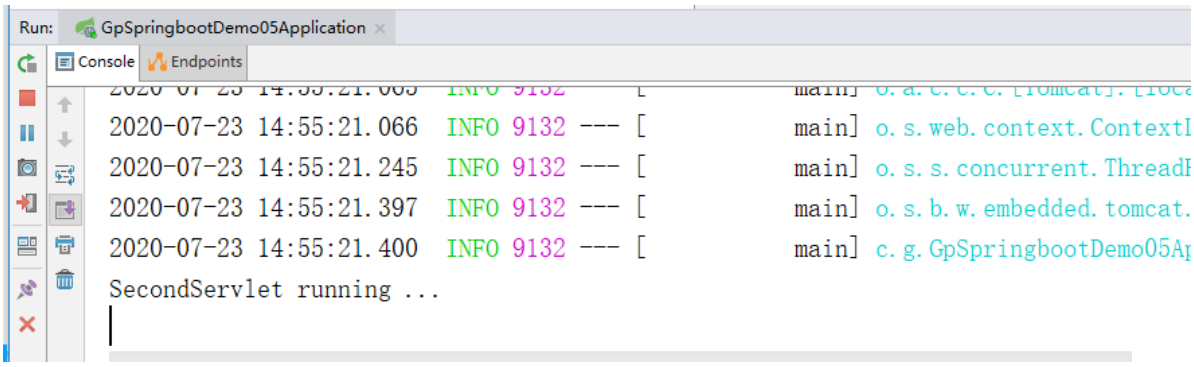
```
package com.gupaoedu.servlet;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;

/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 * @Description: ${todo}
 * @date 2020/7/23 14:52
 */
public class SecondServlet extends HttpServlet {
```

```
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        System.out.println("SecondServlet running ... ");
        PrintWriter out = resp.getWriter();
        out.write("success ... ");
        out.flush();
        out.close();
    }
}
```

2.在启动类中显示在注册

```
@SpringBootApplication
// 在SpringBoot启动的时候会扫描@WebServlet注解
@ServletComponentScan()
public class GpSpringbootDemo05Application {

    public static void main(String[] args) {
        SpringApplication.run(GpSpringbootDemo05Application.class, args);
    }


    @Bean
    public ServletRegistrationBean getRegistrationBean(){
        // 将要添加的Servlet封装为一个ServletRegistrationBean对象
        ServletRegistrationBean registrationBean = new
ServletRegistrationBean(new SecondServlet());
        // 设置映射信息
        registrationBean.addUrlMappings("/second");
        return registrationBean;
    }
}
```

3.测试



2.SpringBoot整合Filter

2.1 第一种方式

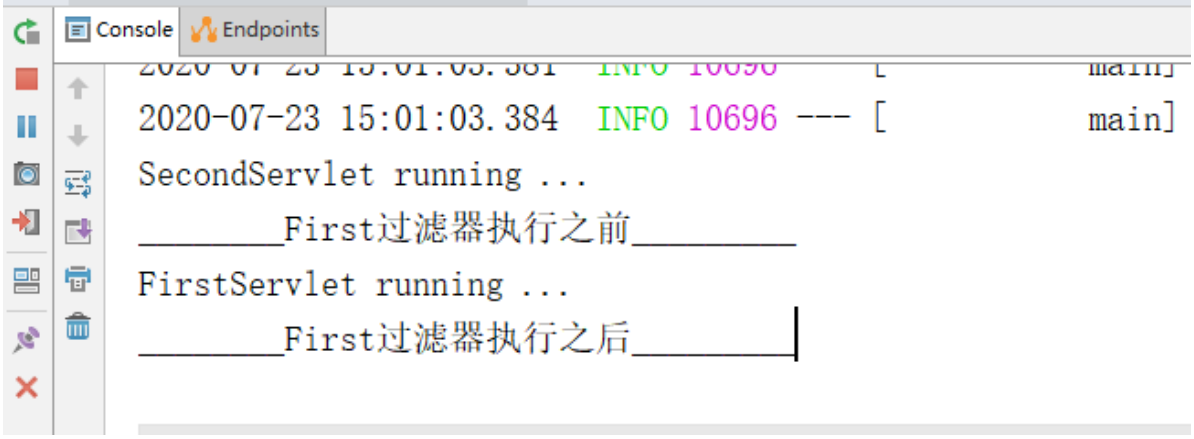
直接在过滤器中添加@WebFilter注解

```
package com.gupaoedu.filter;

import javax.servlet.*;
import javax.servlet.annotation.WebFilter;
import java.io.IOException;

/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 * @Description: ${todo}
 * @date 2020/7/23 14:58
 */
@WebFilter(urlPatterns = "/first")
public class FirstFilter implements Filter {
    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
        System.out.println("----init----");
    }

    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse
servletResponse, FilterChain filterChain) throws IOException, ServletException {
        System.out.println("________First过滤器执行之前_________");
        filterChain.doFilter(servletRequest,servletResponse);
        System.out.println("________First过滤器执行之后_________");
    }

    @Override
    public void destroy() {
        System.out.println("****destroy****");
    }
}
```

在启动器中添加@ServletCompoenentScan

```
@SpringBootApplication
// 在SpringBoot启动的时候会扫描@WebServlet注解
@ServletComponentScan()
public class GpSpringbootDemo05Application {

    public static void main(String[] args) { SpringApplication.run(GpSpringbootDemo05Application.class, args)


    @Bean
    public ServletRegistrationBean getRegistrationBean(){
        // 将要添加的Servlet封装为一个ServletRegistrationBean对象
        ServletRegistrationBean registrationBean = new ServletRegistrationBean(new SecondServlet());
        // 设置映射信息
        registrationBean.addUrlMappings("/second");
        return registrationBean;
    }
}
```

测试



2.2 第二种方式

1.创建自定义的过滤器，不需要添加@WebFilter注解

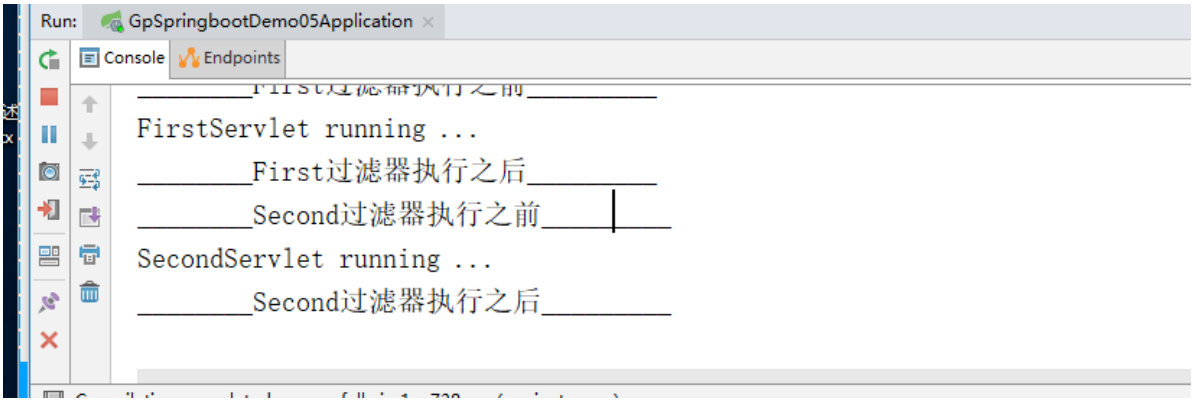
```
package com.gupaoedu.filter;

import javax.servlet.*;
import java.io.IOException;

/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 * @Description: ${todo}
 * @date 2020/7/23 15:03
 */
public class SecondFilter implements Filter {
    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
        System.out.println("--second--init----");
    }

    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse
servletResponse, FilterChain filterChain) throws IOException, ServletException {
        System.out.println("_____Second过滤器执行之前_____");
        filterChain.doFilter(servletRequest,servletResponse);
        System.out.println("_____Second过滤器执行之后_____");
    }

    @Override
    public void destroy() {
        System.out.println("****destroy****");
    }
}
```

2.在启动类中显示的注册

```
package com.gupaoedu;
```

```java
import com.gupaoedu.filter.SecondFilter;
import com.gupaoedu.servlet.SecondServlet;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.web.servlet.FilterRegistrationBean;
import org.springframework.boot.web.servlet.ServletComponentScan;
import org.springframework.boot.web.servlet.ServletRegistrationBean;
import org.springframework.context.annotation.Bean;

@SpringBootApplication
// 在SpringBoot启动的时候会扫描@WebServlet注解
@ServletComponentScan()
public class GpSpringbootDemo05Application {

    public static void main(String[] args) {
        SpringApplication.run(GpSpringbootDemo05Application.class, args);
    }


    @Bean
    public ServletRegistrationBean getRegistrationBean(){
        // 将要添加的Servlet封装为一个ServletRegistrationBean对象
        ServletRegistrationBean registrationBean = new
ServletRegistrationBean(new SecondServlet());
        // 设置映射信息
        registrationBean.addUrlMappings("/second");
        return registrationBean;
    }

    @Bean
    public FilterRegistrationBean getRegistractionBean(){
        FilterRegistrationBean bean = new FilterRegistrationBean(new
SecondFilter());
        bean.addUrlPatterns("/second");
        return bean;
    }
}
```

3.测试



3.SpringBoot整合Listener

3.1 第一种方式

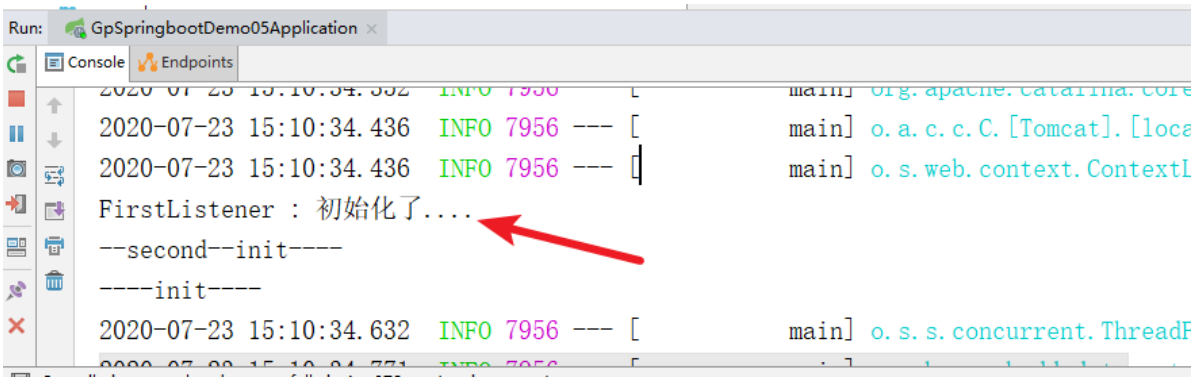
1.创建自定义的Listener

```
package com.gupaoedu.listener;

import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;
import javax.servlet.annotation.WebListener;

/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 * @Description: ${todo}
 * @date 2020/7/23 15:08
 */
@WebListener
public class FirstListener implements ServletContextListener {

    @Override
    public void contextInitialized(ServletContextEvent sce) {
        System.out.println("FirstListener : 初始化了....");
    }

    @Override
    public void contextDestroyed(ServletContextEvent sce) {
        System.out.println("FirstListener : 销毁了....");
    }
}
```

2.添加扫描注解

```
// 在SpringBoot启动的时候会扫描@WebServlet注解
@ServletComponentScan()
public class GpSpringbootDemo05Application {

    public static void main(String[] args) { SpringApplication.

    @Bean
    public ServletRegistrationBean getRegistrationBean() {
        // 将要添加的Servlet封装为一个ServletRegistrationBean对象
        ServletRegistrationBean registrationBean = new ServletReg
```

3.启动测试



3.2 第二种方式

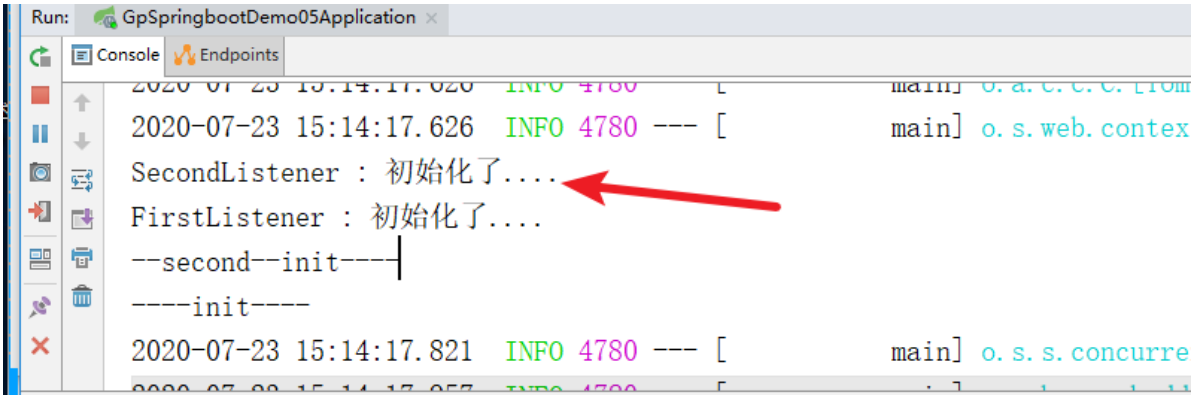
1.创建自定义Listener

```
package com.gupaoedu.listener;

import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;

/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 * @Description: ${todo}
 * @date 2020/7/23 15:12
 */
public class SecondListener implements ServletContextListener {
    @Override
    public void contextInitialized(ServletContextEvent sce) {
        System.out.println("SecondListener : 初始化了....");
    }

    @Override
    public void contextDestroyed(ServletContextEvent sce) {
        System.out.println("SecondListener : 销毁了....");
    }
}
```

2.显示的在启动类中注册

```
 @Bean
public ServletListenerRegistrationBean getListenerRegistrationBean(){
    ServletListenerRegistrationBean bean = new
ServletListenerRegistrationBean(new SecondListener());
    return bean;
}
```

3.测试



4.SpringBoot如何实现文件上传和下载

4.1 文件上传

1.创建提交的表单

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <h1>文件上传案例</h1>
    <form action="/user/upload" method="post" enctype="multipart/form-data">
        <label>账号:</label><input type="text" name="username"><br/>
        <label>照片:</label><input type="file" name="upload"><br/>
        <input type="submit" value="提交">
    </form>
</body>
</html>
```

2.服务处理上传请求

```
package com.gupaoedu.controller;

import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.multipart.MultipartFile;

import java.io.File;
import java.io.IOException;

/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 * @Description: ${todo}
 * @date 2020/7/23 15:47
```

```
 */
@RestController("/user")
public class UserController {

    @RequestMapping("/upload")
    public String upload(MultipartFile upload,String username) throws
IOException {
        System.out.println("userName:" + username + " 文件名称:" +
upload.getOriginalFilename());
        upload.transferTo(new File("d:/" ,upload.getOriginalFilename()));
        return "success ...";
    }
}
```

3.配置相关的上传参数

```
spring.servlet.multipart.enabled=true
# 设置单个文件上传的大小
spring.servlet.multipart.max-file-size=200MB
# 设置一次上传文件总的大小
spring.servlet.multipart.max-request-size=200MB
```

4.2 文件下载

1.页面提供一个下载按钮

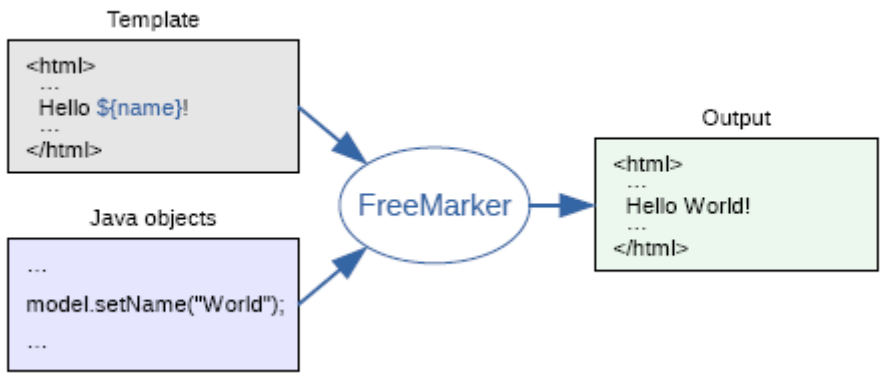
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <h1>文件上传案例</h1>
    <h2><a href="/user/download">文件下载</a></h2>
    <form action="user/upload" method="post" enctype="multipart/form-data">
        <label>账号:</label><input type="text" name="username"><br/>
        <label>照片:</label><input type="file" name="upload"><br/>
        <input type="submit" value="提交">
    </form>
</body>
</html>
```

2.服务端处理下载请求

```
@RequestMapping("/user/download")
public void downloadFile(HttpServletRequest request, HttpServletResponse
response){
    File file = new File("d://1.png");
```

```
    // 设置响应的头和客户端保存的文件名
    response.setCharacterEncoding("utf-8");
    response.setContentType("multipart/form-data");
    response.setHeader("Content-Disposition", "attachment;fileName=" +
file.getName());
    InputStream in = null;
    ServletOutputStream out = null;
    try {
        // 文件的复制
        in = new FileInputStream(file);
        out = response.getOutputStream();
        // 循环读取
        byte[] b = new byte[1024];
        int length = 0;
        while((length = in.read(b)) > 0){
            out.write(b,0,length);
        }

    }catch (Exception e){
        e.printStackTrace();
    }finally {
        try {
            in.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
        try {
            out.flush();
            out.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

5.SpringBoot整合Freemarker



1.添加对应的依赖

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-freemarker</artifactId>
</dependency>
```

2.添加一个自定义的控制器

```
package com.gupaoedu.controller;

import com.gupaoedu.bean.User;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;

import java.util.ArrayList;
import java.util.List;

/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 * @Description: ${todo}
 * @date 2020/7/23 17:07
 */
@Controller
public class UserController {

    @RequestMapping("/showUser")
    public String showUser(Model model){
        List<User> list = new ArrayList<>();
        list.add(new User(1,"zhangsan",22));
        list.add(new User(2,"lisi",23));
        list.add(new User(3,"wangwu",24));
        model.addAttribute("list",list);
        return "user";
    }
}
```

3.属性文件配置

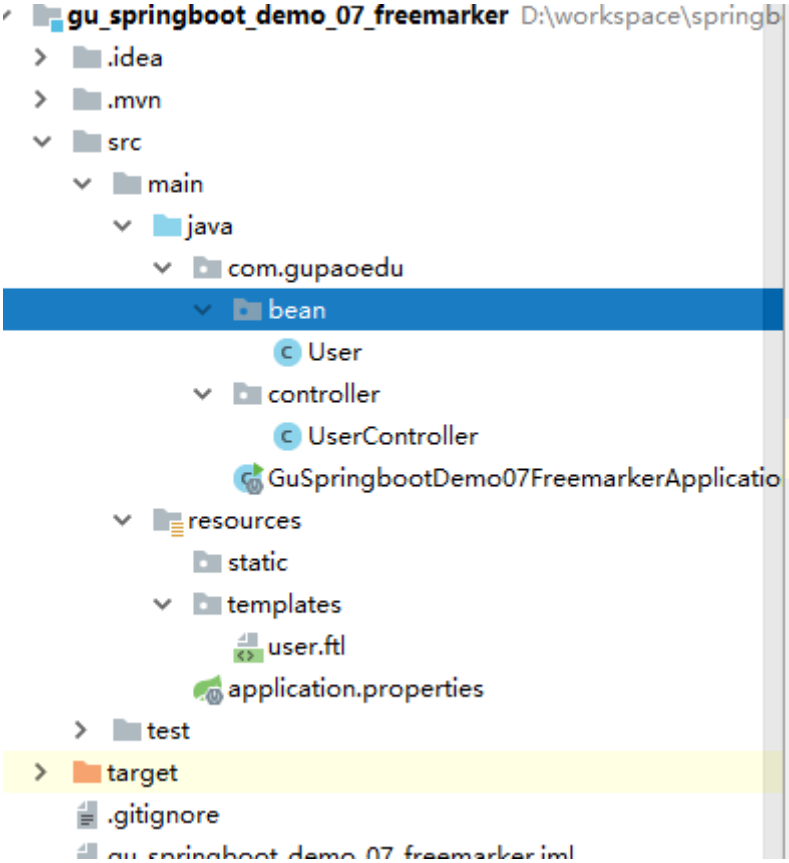
```
spring.freemarker.suffix=.ftl
```

4.模板页面

是一个ftl文件

```
<html>
    <head>
```

```
        <title>用户信息</title>
        <meta charset="UTF-8">
    </head>
    <body>
        <table border="1" align="center" width="50%">
            <tr>
                <th>ID</th>
                <th>姓名</th>
                <th>年龄</th>
            </tr>
            <#list list as user>
                <tr>
                    <td>${user.id}</td>
                    <td>${user.userName}</td>
                    <td>${user.age}</td>
                </tr>
            </#list>
        </table>
    </body>
</html>
```



6.SpringBoot整合Thymeleaf

1.添加相关的依赖

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```

2.创建自定义的控制器

```
package com.gupaoedu.controller;

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

/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 * @Description: ${todo}
 * @date 2020/7/23 17:22
 */
@Controller
public class UserController {

    @RequestMapping("/show")
    public String showInfo(Model model){
        model.addAttribute("msg","Thymeleaf Hello ....");
        return "index";
    }
}
```

3.创建对应的模板页面

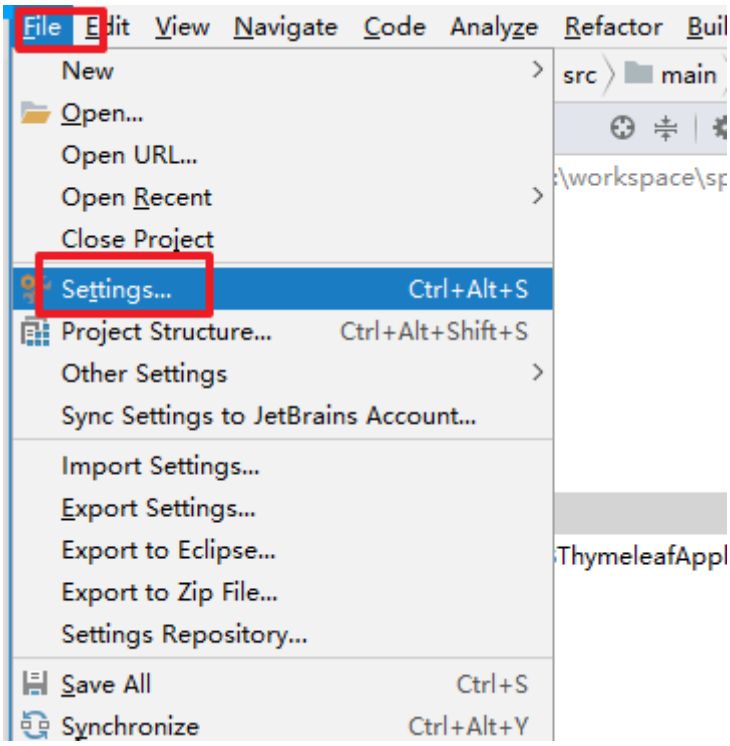
Thymeleaf的模板页面的后缀是`.html`和我们讲的html页面的后缀是一样，但可以写标签

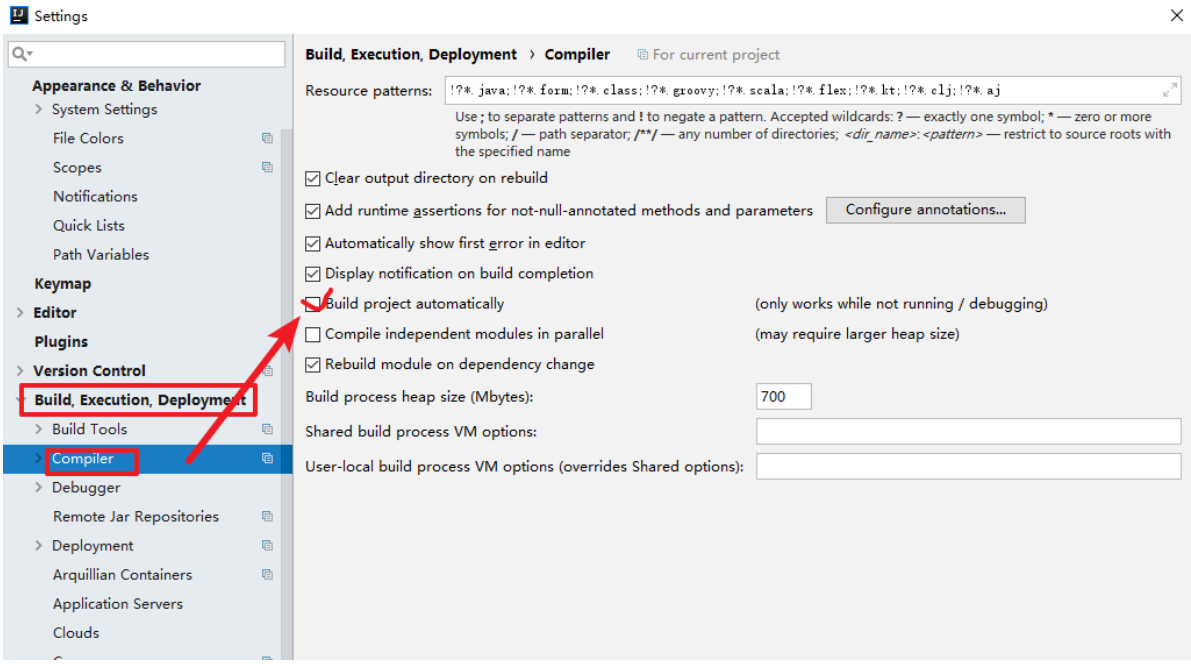
```
<!DOCTYPE html>
<html lang="en"  xmlns="http://www.w3.org/1999/xhtml"
xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <h1>Thymeleaf整合</h1>
    <hr>
    <span th:text="${msg}"></span>
</body>
</html>
```

4.测试

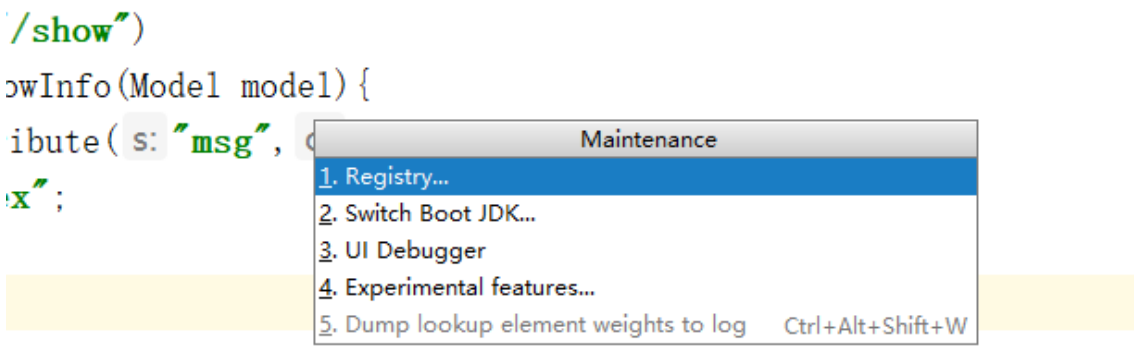
7.SpringBoot如何实现热部署操作

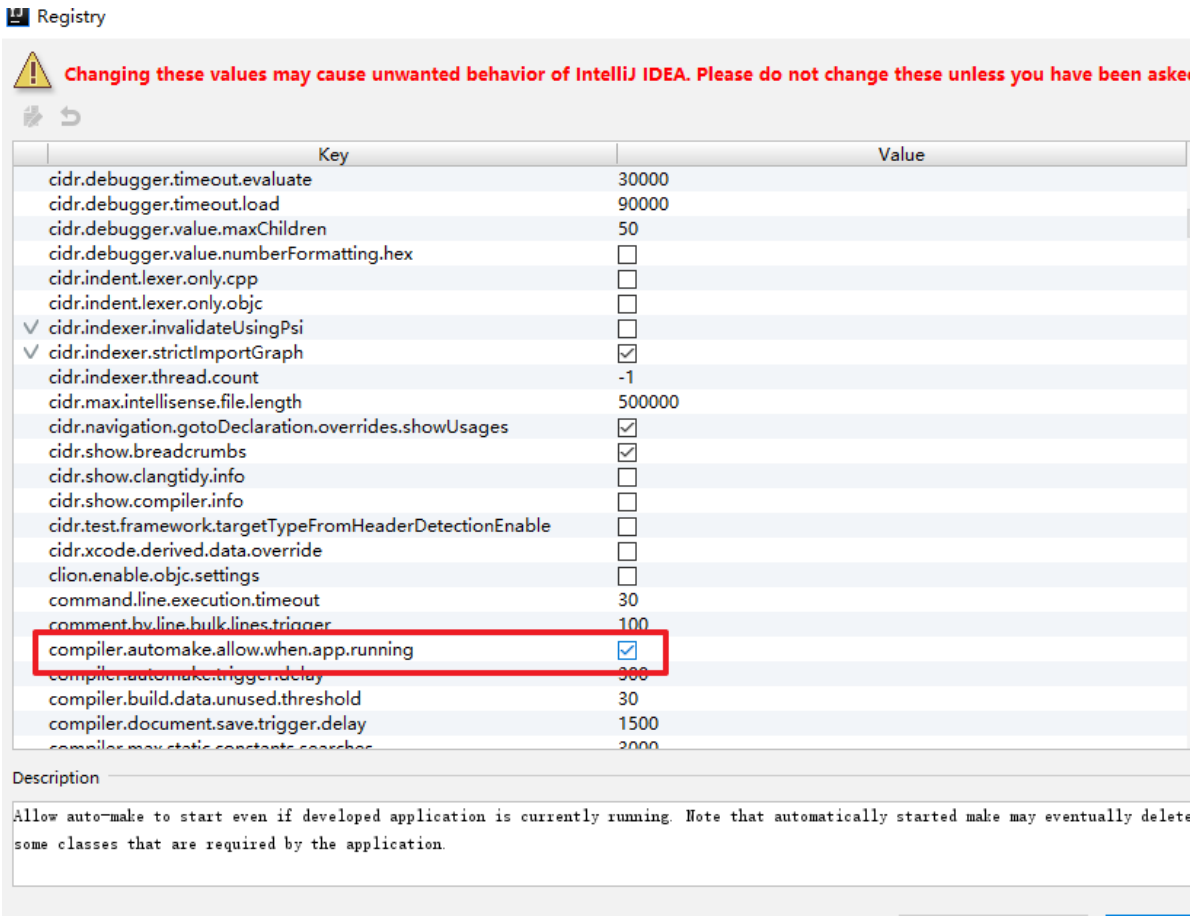
1.放开配置





2. ctrl+shift+alt+'/'





3.添加spring-boot-devtools

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
</dependency>

----
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
            <configuration>
                <fork>true</fork>
            </configuration>
        </plugin>
    </plugins>
</build>
```

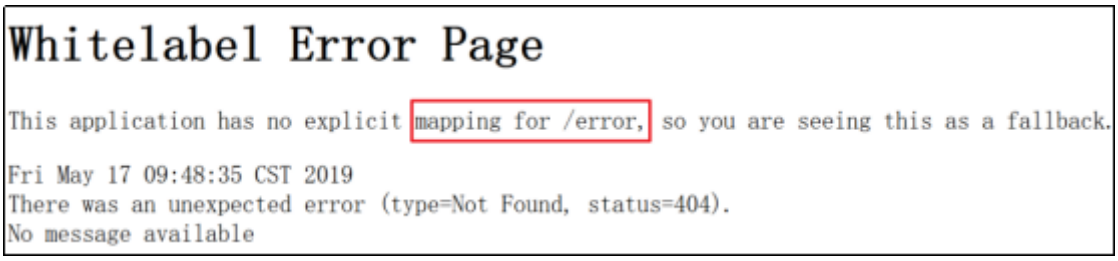
4.测试

修改Java代码，IDEA就会帮助我们重新加载对应的文件，这时我们就不需要人为的去重启服务

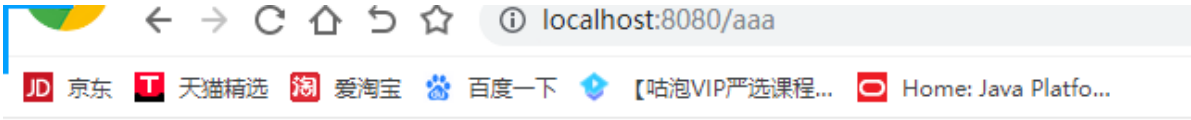
8.SpringBoot中的异常处理

8.1 自定义错误页面

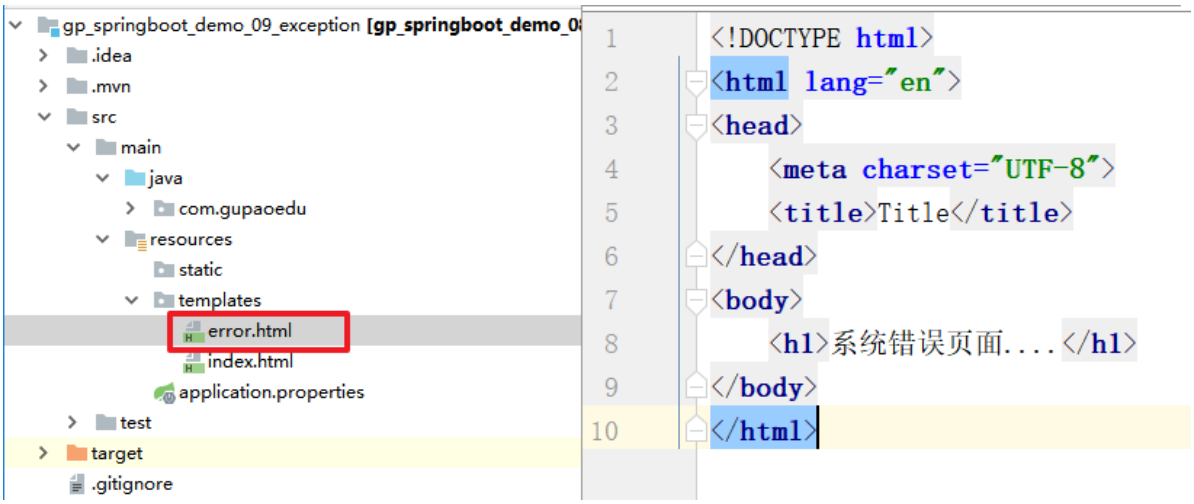
SpringBoot 默认的处理异常的机制：SpringBoot 默认的已经提供了一套处理异常的机制。一旦程序中出现了异常 SpringBoot 会像/error 的 url 发送请求。在 springBoot 中提供了一个叫 BasicExceptionController 来处理/error 请求，然后跳转到默认显示异常的页面来展示异常信息



我们只需要在 `resources/templates` 中添加一个 `error.html` 页面即可



系统错误页面....



8.2 @ExceptionHandler处理

针对特定的异常处理。

控制器：

```
@RequestMapping("/show1")
    public String showInfo1(){
        String msg = null;
        msg.length(); // NullPointerException
        return "success";
    }
```

```
    /**
     * 如果当前类中出现了NullPointerException异常就会跳转到本方法对应的view中
     * @return
     */
    @ExceptionHandler(value = {NullPointerException.class})
    public ModelAndView nullPointerExceptionHandler(Exception e){
        ModelAndView view = new ModelAndView();
        view.addObject("error",e.toString());
        view.setViewName("error1");
        return view;
    }

    /**
     * 如果当前类中出现了ArithmeticException异常就会跳转到本方法对应的view中
     * @return
     */
    @ExceptionHandler(value = {ArithmeticException.class})
    public ModelAndView arithmeticExceptionHandler(Exception e){
        ModelAndView view = new ModelAndView();
        view.addObject("error",e.toString());
        view.setViewName("error2");
        return view;
    }

    @RequestMapping("/show2")
    public String showInfo2(){
        int i = 0;
        int b = 100;
        System.out.println(b/i); // ArithmeicExpetion
        return "success";
    }
```

异常处理代码和业务代码耦合性比较强

8.3 @ControllerAdvice处理

实现业务代码和系统异常处理代码解耦

```
package com.gupaoedu.handler;

import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.servlet.ModelAndView;

/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 * @Description: ${todo}
 * @date 2020/7/27 16:06
 */
@ControllerAdvice
public class GlobalException {
    /**
```

```
     * 如果当前类中出现了NullPointerException异常就会跳转到本方法对应的view中
     * @return
     */
    @ExceptionHandler(value = {NullPointerException.class})
    public ModelAndView nullPointerExceptionHandler(Exception e){
        ModelAndView view = new ModelAndView();
        view.addObject("error",e.toString());
        view.setViewName("error1");
        return view;
    }

    /**
     * 如果当前类中出现了ArithmeticException异常就会跳转到本方法对应的view中
     * @return
     */
    @ExceptionHandler(value = {ArithmeticException.class})
    public ModelAndView arithmeticExceptionHandler(Exception e){
        ModelAndView view = new ModelAndView();
        view.addObject("error",e.toString());
        view.setViewName("error2");
        return view;
    }
}
```

8.4 SimpleMappingExceptionResolver处理

通过系统提供的异常映射处理实现

```
package com.gupaoedu;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.web.servlet.handler.SimpleMappingExceptionResolver;
import  java.util.Properties;

@SpringBootApplication
public class GpSpringbootDemo08ThymeleafApplication {

    public static void main(String[] args) {
        SpringApplication.run(GpSpringbootDemo08ThymeleafApplication.class,
args);
    }

    /**
     * 异常信息和对应的 处理地址的 映射
     * @return
     */
    @Bean
    public SimpleMappingExceptionResolver getSimpleMappingExceptionResolver(){
```

```
        SimpleMappingExceptionResolver mapping = new
SimpleMappingExceptionResolver();
        Properties mappings = new Properties();
        mappings.setProperty("java.lang.NullPointerException","error1");
        mappings.setProperty("java.lang.ArithmeticException","error2");
        mapping.setExceptionMappings(mappings);
        return mapping;
    }
}
```

8.5 自定义HandlerExceptionResolver

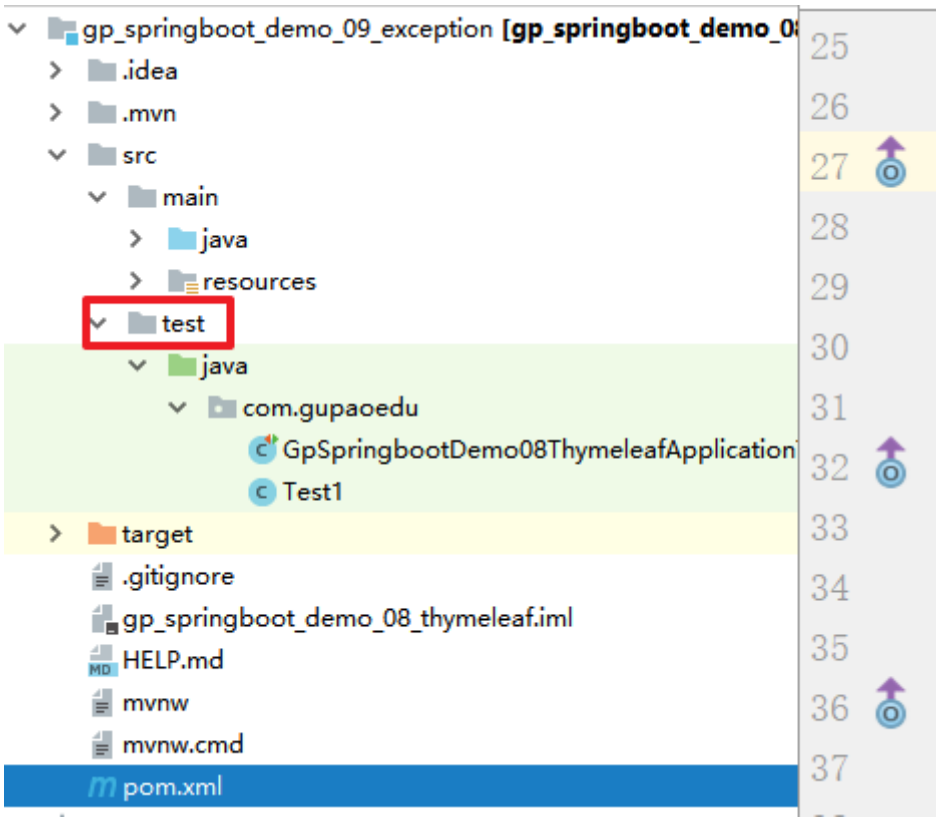
```
package com.gupaoedu.handler;

import org.springframework.stereotype.Component;
import org.springframework.web.servlet.HandlerExceptionResolver;
import org.springframework.web.servlet.ModelAndView;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 * @Description: ${todo}
 * @date 2020/7/27 16:17
 */
@Component
public class MyHandlerExceptionResolver implements HandlerExceptionResolver {
    /**
     * 自定义的全局异常
     * @param httpServletRequest
     * @param httpServletResponse
     * @param o
     * @param e
     * @return
     */
    @Override
    public ModelAndView resolveException(HttpServletRequest httpServletRequest
            , HttpServletResponse httpServletResponse
            , Object o, Exception e) {
        System.out.println("全局的自定义异常处理触发了....");
        ModelAndView mv = new ModelAndView();
        if(e instanceof  NullPointerException){
            mv.setViewName("error1");
            mv.addObject("error","空指针异常");
        }else if(e instanceof  ArithmeticException){
            mv.setViewName("error2");
            mv.addObject("error","算数异常");
        }
        return mv;
    }
}
```

9.SpringBoot中的单元测试

9.1 添加依赖

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
```

9.2 单元测试




```
@RunWith(SpringRunner.class)
@SpringBootTest
public class GpSpringbootDemo08ThymeleafApplicationTests {

    @Autowired
    private IUserService userService;

    @Test
    public void contextLoads() {
        System.out.println(userService.query());
    }

}
```

10.SpringBoot整合MyBatis

整合 MyBatis同时结合SpringMVC+Thymeleaf完成CRUD操作

10.1 整合操作

依赖

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.gupaoedu</groupId>
    <artifactId>gp_springboot_mybatis_demo</artifactId>
    <version>1.0-SNAPSHOT</version>

    <!-- 配置依赖的父类 -->
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.1.5.RELEASE</version>
    </parent>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-thymeleaf</artifactId>
        </dependency>
        <dependency>
            <groupId>org.mybatis.spring.boot</groupId>
            <artifactId>mybatis-spring-boot-starter</artifactId>
            <version>1.3.2</version>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
        </dependency>
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>druid</artifactId>
            <version>1.0.14</version>
        </dependency>
    </dependencies>

</project>
```

配置文件

```
# jdbc的相关配置
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/gp?serverTimezone=UTC
spring.datasource.username=root
spring.datasource.password=123456

# 连接池
spring.datasource.type=com.alibaba.druid.pool.DruidDataSource

## mybatis的package别名
mybatis.type-aliases-package=com.gupaoedu.pojo

# 指定MyBatis的映射文件的路径
mybatis.mapper-locations=classpath:mapper/*.xml
```

启动器

```
package com.gupaoedu;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 * @Description: ${todo}
 * @date 2020/7/27 17:37
 */
@SpringBootApplication
public class StartApp {

    public static void main(String[] args) {
        SpringApplication.run(StartApp.class,args);
    }
}
```

数据库表结构

```
CREATE TABLE `users` (
`id` int(11) NOT NULL AUTO_INCREMENT,
`name` varchar(255) DEFAULT NULL,
`age` int(11) DEFAULT NULL, PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

创建对应的Pojo对象

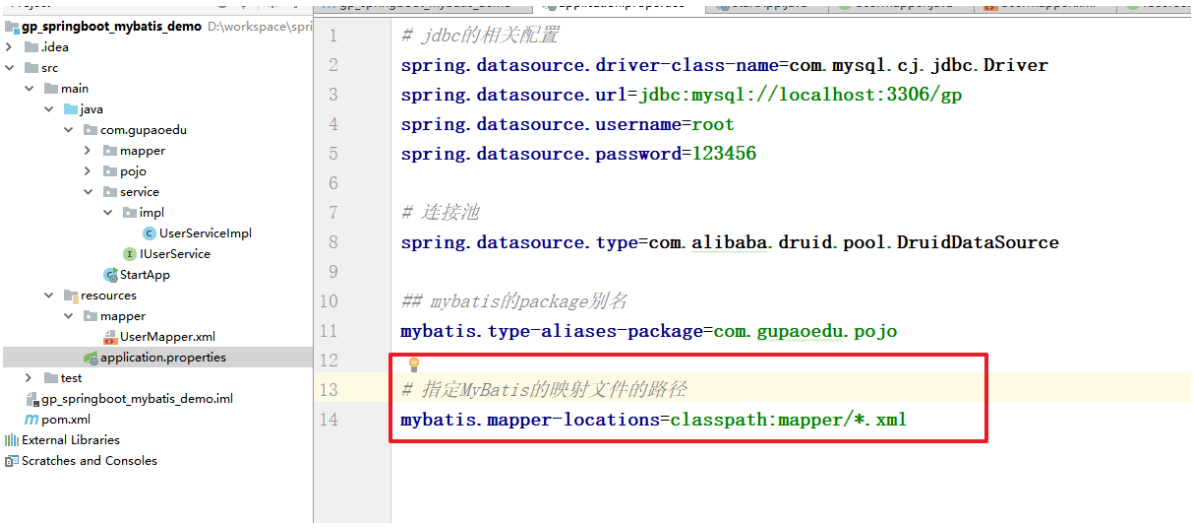
```java
package com.gupaoedu.pojo;

/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 * @Description: ${todo}
 * @date 2020/7/27 17:43
 */
public class User {

    private Integer id;

    private String name;

    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 Integer getAge() {
        return age;
    }

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

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

mapper接口

```
/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 * @Description: ${todo}
 * @date 2020/7/27 17:45
 */
public interface UserMapper {

    public List<User> query(User user);
}
```

mapper映射文件

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper
        PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.gupaoedu.mapper.UserMapper">
    <select id="query" resultType="User">
        select * from users
    </select>
</mapper>
```



service信息

```
package com.gupaoedu.service;

import com.gupaoedu.pojo.User;

import java.util.List;

/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 * @Description: ${todo}
```

```
 * @date 2020/7/27 17:46
 */
public interface IUserService {

    public List<User> query(User user);
}
```

```
package com.gupaoedu.service.impl;

import com.gupaoedu.mapper.UserMapper;
import com.gupaoedu.pojo.User;
import com.gupaoedu.service.IUserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;

/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 * @Description: ${todo}
 * @date 2020/7/27 17:47
 */
@Service
public class UserServiceImpl implements IUserService {

    @Autowired
    private UserMapper mapper;
    @Override
    public List<User> query(User user) {
        return mapper.query(user);
    }
}
```

controller信息

```
package com.gupaoedu.controller;

import com.gupaoedu.pojo.User;
import com.gupaoedu.service.IUserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import java.util.List;

/**
 * 让每一个人的职业生涯不留遗憾
```

```
 *
 * @author 波波老师【咕泡学院】
 * @Description: ${todo}
 * @date 2020/7/27 17:49
 */
@RestController
public class UserController {

    @Autowired
    private IUserService service;

    @RequestMapping("/user/query")
    public List<User> query(){
        return service.query(null);
    }
}
```

最后启动器中我们的制定Mapper的扫描路径

```
9   *
10  * @author 波波老师【咕泡学院】
11  * @Description: ${todo}
12  * @date 2020/7/27 17:37
13  */
14  @SpringBootApplication
15  @MapperScan("com.gupaoedu.mapper") // 用户扫描Mapper接口
16  public class StartApp {
17
18      public static void main(String[] args) {
19          SpringApplication.run(StartApp.class,args);
20      }
21  }
22
23
```

10.2 用户信息查询

1.控制器修改

```java
@Controller
public class UserController {

    @Autowired
    private IUserService service;

    @RequestMapping("/user/query")
    public String query(Model model){
        model.addAttribute("list",service.query(null));
        return "user";
    }
}
```

2.模板页面

```html
<!DOCTYPE html>
<html lang="en" xmlns="http://www.w3.org/1999/xhtml"
xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>用户信息</title>
</head>
<body>
    <h1>用户管理</h1>
    <table border="1" style="width: 300px">
        <tr>
            <th>用户ID</th>
            <th>用户姓名</th>
            <th>用户年龄</th>
        </tr>
        <tr th:each="user:${list}">
            <td th:text="${user.id}"></td>
            <td th:text="${user.name}"></td>
            <td th:text="${user.age}"></td>
        </tr>
    </table>
</body>
</html>
```

3.效果

localhost:8080/user/query

京东 天猫精选 爱淘宝 百度一下 [咕泡VIP严选课程... Home: Java Platfo...

用户管理

用户ID	用户姓名	用户年龄
1	bobo	18
2	gupao	3

10.3 用户信息的添加

mapper

```
public interface UserMapper {

    public List<User> query(User user);

    public Integer addUser(User user);
}
```

```
<insert id="addUser" parameterType="User">
    INSERT INTO users (name,age)VALUES(#{name},#{age})
</insert>
```

service

```
public interface IUserService {

    public List<User> query(User user);

    public Integer addUser(User user);
}
```

```
@Service
@Transactional
public class UserServiceImpl implements IUserService {

    @Autowired
    private UserMapper mapper;
    @Override
    public List<User> query(User user) {
        return mapper.query(user);
```

```
    }

    @Override
    public Integer addUser(User user) {
        return mapper.addUser(user);
    }
}
```

controller

```
@Controller
public class UserController {

    @Autowired
    private IUserService service;

    @RequestMapping("/user/query")
    public String query(Model model){
        model.addAttribute("list",service.query(null));
        return "user";
    }
    @RequestMapping("/user/save")
    public String addUser(User user){
       service.addUser(user);
       return "redirect:/user/query";
    }
}
```

页面

```
<!DOCTYPE html>
<html lang="en" xmlns="http://www.w3.org/1999/xhtml"
xmlns:th="http://www.thymeleaf.org">
    <head>
        <meta charset="UTF-8">
        <title>用户信息</title>
    </head>
    <body>
        <h1>添加用户</h1>
        <form th:action="@{/user/save}" method="post">
            <label>姓名:</label><input type="text" name="name"><br>
            <label>年龄:</label><input type="text" name="age"><br>
            <input type="submit" value="提交">
        </form>

    </body>
</html>
```

基础跳转请求

```
/**
     * 基础页面的请求
     * @param page
     * @return
     */
@RequestMapping("/{page}")
public String showPage(@PathVariable String page){
    return page;
}
```

10.4 修改用户信息

修改按钮

```
<!DOCTYPE html>
<html lang="en" xmlns="http://www.w3.org/1999/xhtml"
xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>用户信息</title>
</head>
<body>
    <h1>用户管理</h1>
    <table border="1" style="width: 300px">
        <tr>
            <th>用户ID</th>
            <th>用户姓名</th>
            <th>用户年龄</th>
            <th>操作</th>
        </tr>
        <tr th:each="user:${list}">
            <td th:text="${user.id}"></td>
            <td th:text="${user.name}"></td>
            <td th:text="${user.age}"></td>
            <td>
                <a th:href="@{/user/updateInfo(id=${user.id})}">修改</a>
            </td>
        </tr>
    </table>
</body>
</html>
```

用户管理

用户ID	用户姓名	用户年龄	操作
1	bobo	18	修改
2	gupao	3	修改
3	zhangsan	22	修改

mapper

```
package com.gupaoedu.mapper;

import com.gupaoedu.pojo.User;

import java.util.List;

/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 * @Description: ${todo}
 * @date 2020/7/27 17:45
 */
public interface UserMapper {

    public List<User> query(User user);

    public Integer addUser(User user);

    /**
     * 根据id查询用户信息
     * @param id
     * @return
     */
    public User queryById(Integer id);

    public Integer updateUser(User user);
}
```

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper
        PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.gupaoedu.mapper.UserMapper">
    <select id="query" resultType="User">
```

```
        select * from users
    </select>

    <insert id="addUser" parameterType="User">
        INSERT INTO users (name,age)VALUES(#{name},#{age})
    </insert>

    <select id="queryById" resultType="User" >
        select * from users where id = #{id}
    </select>

    <update id="updateUser" parameterType="User">
        update users set name=#{name},age=#{age} where id =#{id}
    </update>
</mapper>
```

```
package com.gupaoedu.controller;

import com.gupaoedu.pojo.User;
import com.gupaoedu.service.IUserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import java.util.List;

/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 * @Description: ${todo}
 * @date 2020/7/27 17:49
 */
@Controller
public class UserController {

    @Autowired
    private IUserService service;

    /**
     * 基础页面的请求
     * @param page
     * @return
     */
    @RequestMapping("/{page}")
    public String showPage(@PathVariable String page){
        return page;
    }

    @RequestMapping("/user/query")
    public String query(Model model){
        model.addAttribute("list",service.query(null));
```

```
        return "user";
    }
    @RequestMapping("/user/save")
    public String addUser(User user){
       service.addUser(user);
       return "redirect:/user/query";
    }
    @RequestMapping("/user/updateInfo")
    public String updateInfo(Integer id,Model model){
        User user = service.queryById(id);
        model.addAttribute("user",user);
        return "updateUser";
    }

    @RequestMapping("/user/update")
    public String updateUser(User user){
        service.updateUser(user);
        return "redirect:/user/query";
    }
}
```

跳转到修改界面

```
<!DOCTYPE html>
<html lang="en" xmlns="http://www.w3.org/1999/xhtml"
xmlns:th="http://www.thymeleaf.org">
    <head>
        <meta charset="UTF-8">
        <title>用户信息</title>
    </head>
    <body>
        <h1>更新用户</h1>
        <form th:action="@{/user/update}" method="post">
            <input type="hidden" name="id" th:value="${user.id}">
            <label>姓名:</label><input type="text" name="name"
th:value="${user.name}"><br>
            <label>年龄:</label><input type="text" name="age"
th:value="${user.age}"><br>
            <input type="submit" value="提交">
        </form>

    </body>
</html>
```

提交修改数据

显示更新的信息

10.5删除用户信息

添加删除按钮

```
<!DOCTYPE html>
<html lang="en" xmlns="http://www.w3.org/1999/xhtml"
xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>用户信息</title>
</head>
<body>
    <h1>用户管理</h1>
    <table border="1" style="width: 300px">
        <tr>
            <th>用户ID</th>
            <th>用户姓名</th>
            <th>用户年龄</th>
            <th>操作</th>
        </tr>
        <tr th:each="user:${list}">
            <td th:text="${user.id}"></td>
            <td th:text="${user.name}"></td>
            <td th:text="${user.age}"></td>
            <td>
                <a th:href="@{/user/updateInfo(id=${user.id})}">修改</a>

                <a th:href="@{/user/deleteUser(id=${user.id})}">删除</a>
            </td>
        </tr>
    </table>
</body>
</html>
```

提交删除的编号

```
@RequestMapping("/user/deleteUser")
public String deleteUser(Integer id){
    service.deleteUserById(id);
    return "redirect:/user/query";
}
```

删除数据

11. SpringBoot整合Shiro

SpringBoot整合Shiro认证

在上面案例的基础上操作

表结构

```
CREATE TABLE `t_user` (
  `id` int(20) NOT NULL AUTO_INCREMENT,
  `username` varchar(20) DEFAULT NULL,
  `password` varchar(100) DEFAULT NULL,
  `salt` varchar(100) DEFAULT NULL,
  `create_time` datetime DEFAULT NULL,
  `state` int(1) DEFAULT NULL,
  `last_login_time` datetime DEFAULT NULL,
  `nickname` varchar(30) DEFAULT NULL,
  `realname` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=7 DEFAULT CHARSET=utf8;
```

整合步骤:

1.添加依赖

```
<dependency>
    <groupId>org.apache.shiro</groupId>
    <artifactId>shiro-spring</artifactId>
    <version>1.3.2</version>
</dependency>
```

2.自定义的realm

```
package com.gupaoedu.realm;

import org.apache.shiro.authc.AuthenticationException;
import org.apache.shiro.authc.AuthenticationInfo;
import org.apache.shiro.authc.AuthenticationToken;
import org.apache.shiro.authz.AuthorizationInfo;
import org.apache.shiro.realm.AuthorizingRealm;
import org.apache.shiro.subject.PrincipalCollection;

/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 * @Description: ${todo}
 * @date 2020/7/27 20:24
 */
public class AuthcRealm extends AuthorizingRealm {


    /**
     * 认证的方法
     * @param authenticationToken
     * @return
     * @throws AuthenticationException
     */
    @Override
    protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken
authenticationToken) throws AuthenticationException {
        return null;
    }

    /**
     * 授权的方法
     * @param principalCollection
     * @return
     */
    @Override
    protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection
principalCollection) {
        return null;
    }
}
```

3.Shiro的配置类

将我们原来写在xml文件中的配置添加到了Java类中

```
package com.gupaoedu.config;

import com.gupaoedu.realm.AuthcRealm;
import org.apache.shiro.authc.credential.HashedCredentialsMatcher;
import org.apache.shiro.mgt.SecurityManager;
import org.apache.shiro.spring.web.ShiroFilterFactoryBean;
import org.apache.shiro.web.mgt.DefaultWebSecurityManager;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.DependsOn;

import java.util.HashMap;
import java.util.Map;

/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 * @Description: ${todo}
 * @date 2020/7/27 20:27
 */
@Configuration
public class ShiroConfig {

    // 散列算法
    private String hashAlgorithmName = "md5";
    // 迭代次数
    private Integer hashIterations = 1024;

    /**
     * 获取凭证匹配器
     * @return
     */
    @Bean
    public HashedCredentialsMatcher hashedCredentialsMatcher(){
        HashedCredentialsMatcher matcher = new HashedCredentialsMatcher();
        matcher.setHashAlgorithmName(hashAlgorithmName);
        matcher.setHashIterations(hashIterations);
        return matcher;
    }

    /**
     * 获取自定义的Realm
     * @return
     */
    @Bean
    public AuthcRealm authcRealm(HashedCredentialsMatcher matcher){
        AuthcRealm realm = new AuthcRealm();
        realm.setCredentialsMatcher(matcher);
        return  realm;
```

```
    }

    /**
     *  获取SecurityManager对象
     * @param realm
     * @return
     */
    @Bean
    public SecurityManager securityManager(AuthcRealm realm){
        DefaultWebSecurityManager manager = new DefaultWebSecurityManager();
        manager.setRealm(realm);
        return manager;
    }

    /**
     * 注册ShiroFilterFactoryBean
     * @param manager
     * @return
     */
    @Bean
    public ShiroFilterFactoryBean shiroFilterFactoryBean(SecurityManager
manager){
        ShiroFilterFactoryBean filter = new ShiroFilterFactoryBean();
        filter.setSecurityManager(manager);
        filter.setLoginUrl("/login.do");
        filter.setSuccessUrl("/success.html");
        filter.setUnauthorizedUrl("/refuse.html");
        // 设置过滤器链
        Map<String,String> map = new HashMap<>();
        map.put("/css/*","anon");
        map.put("/js/**","anon");
        map.put("/img/**","anon");
        map.put("/js/**","anon");
        map.put("/login","anon");
        map.put("/login.do","authc");
        map.put("/**","authc");
        filter.setFilterChainDefinitionMap(map);
        return filter;
    }
}
```

4.认证配置

```
package com.gupaoedu.realm;

import com.gupaoedu.pojo.User;
import com.gupaoedu.service.IUserService;
import org.apache.shiro.authc.*;
import org.apache.shiro.authz.AuthorizationInfo;
import org.apache.shiro.realm.AuthorizingRealm;
import org.apache.shiro.subject.PrincipalCollection;
import org.apache.shiro.util.SimpleByteSource;
import org.springframework.beans.factory.annotation.Autowired;
```

```java
import java.util.List;

/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 * @Description: ${todo}
 * @date 2020/7/27 20:24
 */
public class AuthcRealm extends AuthorizingRealm {

    @Autowired
    private IUserService service;

    /**
     * 认证的方法
     * @param authenticationToken
     * @return
     * @throws AuthenticationException
     */
    @Override
    protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken
authenticationToken) throws AuthenticationException {
        UsernamePasswordToken token = (UsernamePasswordToken)
authenticationToken;
        String userName = token.getUsername();
        System.out.println("开始认证: " + userName);
        User user = new User();
        user.setUsername(userName);
        // 根据账号认证
        List<User> list = service.query(user);
        if(list == null || list.size() != 1){
            // 账号不存在或者异常
            return  null;
        }
        user = list.get(0);
        return new SimpleAuthenticationInfo(user
                ,user.getPassword() // 密码
                ,new SimpleByteSource(user.getSalt()) // salt
                ,"authcRealm" // 自定义的Realm名称
        );
    }

    /**
     * 授权的方法
     * @param principalCollection
     * @return
     */
    @Override
    protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection
principalCollection) {
        return null;
    }
}
```

控制器

```
package com.gupaoedu.controller;

import org.apache.shiro.SecurityUtils;
import org.apache.shiro.web.filter.authc.FormAuthenticationFilter;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

import javax.servlet.http.HttpServletRequest;

/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 * @Description: ${todo}
 * @date 2020/7/27 21:15
 */
@Controller
public class AuthcController {

    @RequestMapping("/login.do")
    public String login(HttpServletRequest request){
        // 认证失败的异常信息
        Object obj =
request.getAttribute(FormAuthenticationFilter.DEFAULT_ERROR_KEY_ATTRIBUTE_NAME);
        System.out.println("认证失败的信息：" + obj);
        return "login";
    }
    @RequestMapping("/logout.do")
    public String logout(){
        SecurityUtils.getSubject().logout();
        return "redirect:/login";
    }
}
```

登录界面

```
<!DOCTYPE html>
<html lang="en" xmlns="http://www.w3.org/1999/xhtml"
xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <h1>登录管理</h1>
    <form th:action="/login.do" method="post">
        <label>账号：</label><input type="text" name="username"><br>
        <label>密码：</label><input type="password" name="password"><br>
```

```
        <input type="submit" value="提交">
    </form>
</body>
</html>
```

SpringBoot整合Shiro授权

注解的使用

注解授权的开启

```
/**
     * 开始Shiro 注解授权操作
     * @param manager
     * @return
     */
@Bean
public AuthorizationAttributeSourceAdvisor
authorizationAttributeSourceAdvisor(SecurityManager manager){
    AuthorizationAttributeSourceAdvisor advisor = new
AuthorizationAttributeSourceAdvisor();
    advisor.setSecurityManager(manager);
    return advisor;
}

@Bean
public DefaultAdvisorAutoProxyCreator defaultAdvisorAutoProxyCreator(){
    DefaultAdvisorAutoProxyCreator proxyCreator = new
DefaultAdvisorAutoProxyCreator();
    proxyCreator.setProxyTargetClass(true);
    return proxyCreator;
}
```

自定义Realm中完成授权操作

```
/**
     * 授权的方法
     * @param principalCollection
     * @return
     */
@Override
protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection
principalCollection) {
    User user = (User) principalCollection.getPrimaryPrincipal();
    System.out.println("授权的账号是: " + user.getUsername());
    SimpleAuthorizationInfo info = new SimpleAuthorizationInfo();
    info.addRole("role1");
    return info;
}
```

控制器

```java
package com.gupaoedu.controller;

import com.gupaoedu.pojo.User;
import com.gupaoedu.service.IUserService;
import org.apache.shiro.authz.annotation.Logical;
import org.apache.shiro.authz.annotation.RequiresRoles;
import org.apache.shiro.crypto.hash.Md5Hash;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import java.util.HashMap;
import java.util.List;

/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 * @Description: ${todo}
 * @date 2020/7/27 17:49
 */
@Controller
public class UserController {

    @Autowired
    private IUserService service;



    @RequiresRoles(value = {"role1"},logical=Logical.OR)
    @RequestMapping("/user/query")
    public String query(Model model){
        model.addAttribute("list",service.query(null));
        return "user";
    }
    @RequiresRoles(value = {"role2"},logical=Logical.OR)
    @RequestMapping("/user/query1")
    public String query1(Model model){
        model.addAttribute("list",service.query(null));
        return "user";
    }

/*    public static void main(String[] args) {
        Md5Hash md5Hash = new Md5Hash("123456","aaa",1024);
        System.out.println(md5Hash);
    }*/
}
```

自定义没有授权的跳转页面

```java
package com.gupaoedu;
```

```java
import org.mybatis.spring.annotation.MapperScan;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.web.servlet.handler.SimpleMappingExceptionResolver;

import java.util.Properties;

/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 * @Description: ${todo}
 * @date 2020/7/27 17:37
 */
@SpringBootApplication
@MapperScan("com.gupaoedu.mapper") // 用户扫描Mapper接口
public class StartApp {

    public static void main(String[] args) {
        SpringApplication.run(StartApp.class,args);
    }

    @Bean
    public SimpleMappingExceptionResolver simpleMappingExceptionResolver(){
        SimpleMappingExceptionResolver resolver = new
SimpleMappingExceptionResolver();
        Properties properties = new Properties();
        properties.setProperty("AuthorizationException","/refuse");
        resolver.setExceptionMappings(properties);
        return resolver;
    }
}
```

标签库的使用

注意是扩展Thymeleaf中Shiro标签库怎么使用

```pom.dependency
<dependency>
    <groupId>com.github.theborakompanioni</groupId>
    <artifactId>thymeleaf-extras-shiro</artifactId>
    <version>2.0.0</version>
</dependency>
```

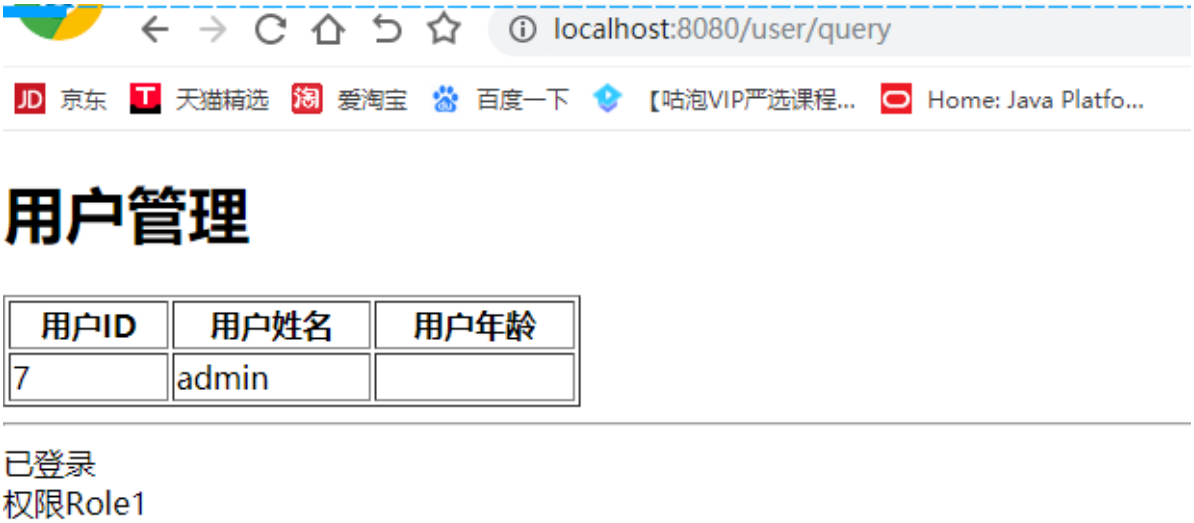
容器中注册 ShiroDialect对象

```
// 添加一个Thymeleaf的模板
@Bean
public ShiroDialect shiroDialect(){
    return new ShiroDialect();
}
```

页面头部引入一下信息

```
xmlns:shiro="http://www.pollix.at/thymeleaf/shiro"
```

```
<!DOCTYPE html>
<html lang="en" xmlns="http://www.w3.org/1999/xhtml"
      xmlns:shiro="http://www.pollix.at/thymeleaf/shiro"
      xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>用户信息</title>
</head>
<body>
    <h1>用户管理</h1>
    <table border="1" style="width: 300px">
        <tr>
            <th>用户ID</th>
            <th>用户姓名</th>
            <th>用户年龄</th>
        </tr>
        <tr th:each="user:${list}">
            <td th:text="${user.id}"></td>
            <td th:text="${user.username}"></td>
            <td th:text="${user.realname}"></td>

        </tr>
    </table>
<hr>
    <shiro:authenticated>
        已登录<br>
    </shiro:authenticated>
    <span shiro:hasRole="role1">权限Role1</span><br>
    <span shiro:hasRole="role2">权限Role2</span><br>
    <shiro:guest>游客</shiro:guest>
</body>
</html>
```



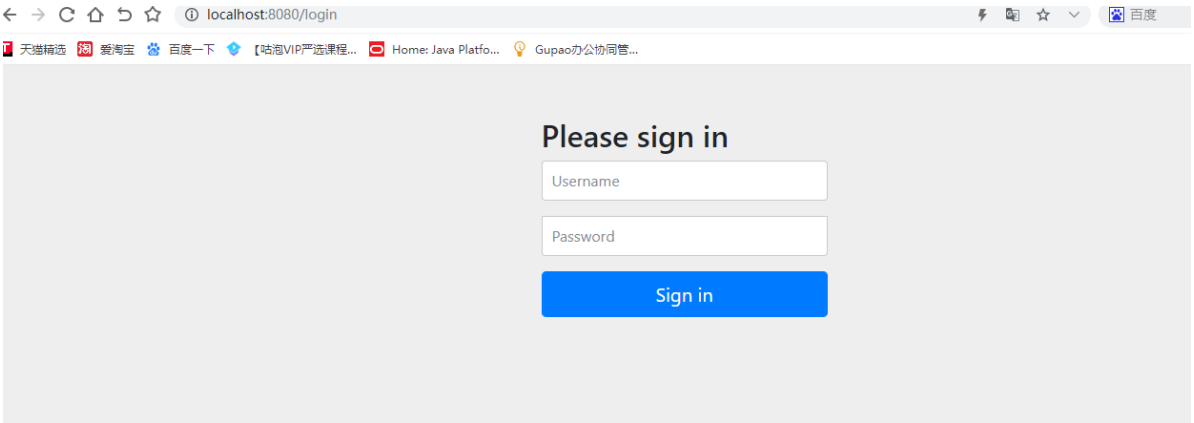
12.SpringBoot整合SpringSecurity

1.基本整合

添加SpringSecurity的依赖即可

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

重启访问即可跳转到对应的登录界面



> 系统启动的时候会帮我们创建一个随机的密码，账号是`user`

```
2020-07-29 10:25:36.252  INFO 6368 --- [  restartedMain] o.s.b.a.w.s.WelcomePageHandlerMapping    : Adding
2020-07-29 10:25:36.469  INFO 6368 --- [  restartedMain] .s.s.UserDetailsServiceAutoConfiguration :

Using generated security password: befe7c64-25c1-4647-9427-daecb9a3b399

2020-07-29 10:25:36.570  INFO 6368 --- [  restartedMain] o.s.s.web.DefaultSecurityFilterChain     : Creati
2020-07-29 10:25:36.620  INFO 6368 --- [  restartedMain] o.s.b.d.a.OptionalLiveReloadServer       : LiveRe
2020-07-29 10:25:36.758  INFO 6368 --- [  restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat
2020-07-29 10:25:36.760  INFO 6368 --- [  restartedMain] g.GpSpringbootDemo08ThymeleafApplication : Starte
2020-07-29 10:25:38.568  INFO 6368 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initia
```

2.自定义登录界面

准备一个登录的HTML页面

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>登录</title>
</head>
<body>
<h2>自定义登录页面</h2>
<form action="/authentication/form" method="post">
    <table>
        <tr>
            <td>用户名:</td>
            <td><input type="text" name="username"></td>
        </tr>
        <tr>
            <td>密码:</td>
            <td><input type="password" name="password"></td>
        </tr>
        <tr>
            <td colspan="2">
                <button type="submit">登录</button>
            </td>
        </tr>
    </table>
</form>
</body>
</html>
```

自定义SpringSecurity的配置类

```java
package com.gupaoedu.config;

import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.authentication.builders.Authentic
ationManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
```

```java
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecuri
ty;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityConf
igurerAdapter;

/**
 * 让每一个人的职业生涯不留遗憾
 *   SpringSecurity的配置类
 * @author 波波老师【咕泡学院】
 * @Description: ${todo}
 * @date 2020/7/29 10:30
 */
@Configuration
@EnableWebSecurity // 方法SpringSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    /**
     * 认证的配置
     * @param auth
     * @throws Exception
     */
    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception
{
        // 配置自定义的账号密码
        auth.inMemoryAuthentication()
                .withUser("zhang")
                .password("{noop}123")
                .roles("USER");// 用户具有的角色
    }


    /**
     * http请求的配置
     * @param http
     * @throws Exception
     */
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.formLogin()
                .loginPage("/login.html") // 指定自定义的登录界面
                .loginProcessingUrl("/login.do") // 必须和登录表单的 action一致
                .and()
                .authorizeRequests() // 定义哪些资源被保护
                .antMatchers("/login.html")
                .permitAll() // login.html可以匿名访问
                .anyRequest()
                .authenticated(); //出来登录页面其他都需要认证
        http.csrf().disable();// 禁用跨越攻击
    }
}
```

3. 数据库认证

创建一个service继承UserDetailService

```
/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 * @Description: ${todo}
 * @date 2020/7/29 10:43
 */
public interface UserService extends UserDetailsService {
}
```

service实现中冲load***方法

```
package com.gupaoedu.service.impl;

import com.gupaoedu.service.UserService;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;

import java.util.ArrayList;
import java.util.List;

/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 * @Description: ${todo}
 * @date 2020/7/29 10:44
 */
@Service
public class UserServiceImpl implements UserService {
    @Override
    public UserDetails loadUserByUsername(String s) throws
UsernameNotFoundException {
        // 模拟数据库操作 根据账号查询
        String password = "456";
        // 假设查询出来的用户的角色
        List<SimpleGrantedAuthority> list = new ArrayList<>();
        list.add(new SimpleGrantedAuthority("USER1"));
        UserDetails userDetails = new User(s,"{noop}"+password,list);
        return userDetails;
    }
}
```

在SpringSecurity的配置类添加配置信息

```
package com.gupaoedu.config;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.authentication.builders.Authentic
ationManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecuri
ty;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityConf
igurerAdapter;
import org.springframework.security.core.userdetails.UserDetailsService;

/**
 * 让每一个人的职业生涯不留遗憾
 *   SpringSecurity的配置类
 * @author 波波老师【咕泡学院】
 * @Description: ${todo}
 * @date 2020/7/29 10:30
 */
@Configuration
@EnableWebSecurity // 方法SpringSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    private UserDetailsService userDetailsService;

    /**
     * 认证的配置
     * @param auth
     * @throws Exception
     */
    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception
{
        // 配置自定义的账号密码
        /*auth.inMemoryAuthentication()
                .withUser("zhang")
                .password("{noop}123")
                .roles("USER");// 用户具有的角色*/
        // 关联自定义的认证的Service
        auth.userDetailsService(userDetailsService);
    }


    /**
     * http请求的配置
     * @param http
     * @throws Exception
     */
    @Override
    protected void configure(HttpSecurity http) throws Exception {
```

```
        http.formLogin()
                .loginPage("/login.html") // 指定自定义的登录界面
                .loginProcessingUrl("/login.do") // 必须和登录表单的 action一致
                .and()
                .authorizeRequests() // 定义哪些资源被保护
                .antMatchers("/login.html")
                .permitAll() // login.html可以匿名访问
                .anyRequest()
                .authenticated(); //出来登录页面其他都需要认证
        http.csrf().disable();// 禁用跨越攻击
    }
}
```

加密认证

在配置类中指定解密器

```
    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception
{
        // 配置自定义的账号密码
        /*auth.inMemoryAuthentication()
                .withUser("zhang")
                .password("{noop}123")
                .roles("USER");// 用户具有的角色*/
        // 关联自定义的认证的Service
        auth.userDetailsService(userDetailsService).passwordEncoder(new
BCryptPasswordEncoder());
    }
```

在service获取对应的加密的密文

```
package com.gupaoedu.service.impl;

import com.gupaoedu.service.UserService;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;

import java.util.ArrayList;
import java.util.List;

/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 * @Description: ${todo}
 * @date 2020/7/29 10:44
 */
```

```
@Service
public class UserServiceImpl implements UserService {
    @Override
    public UserDetails loadUserByUsername(String s) throws
UsernameNotFoundException {
        // 模拟数据库操作 根据账号查询
        String password =
"$2a$10$9tzTU0L5cM7e25RPo.KGnOfzUzeulD0CzOoawooYSiUlrPABkCPXG";
        // 假设查询出来的用户的角色
        List<SimpleGrantedAuthority> list = new ArrayList<>();
        list.add(new SimpleGrantedAuthority("USER1"));
        UserDetails userDetails = new User(s,password,list);
        return userDetails;
    }
}
```

13. SpringBoot整合Ehcache

jar包依赖

```
<dependency>
    <groupId>error</groupId>
    <artifactId>spring-boot-starter-cache</artifactId>
</dependency>
<dependency>
    <groupId>net.sf.ehcache</groupId>
    <artifactId>ehcache</artifactId>
</dependency>
```

添加Ehcache的配置

```
<ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation="http://ehcache.org/ehcache.xsd"
         updateCheck="false">

    <diskStore path="java.io.tmpdir"/>
    <!--defaultCache:echcache 的默认缓存策略 -->
    <defaultCache
            maxElementsInMemory="10000"
            eternal="false"
            timeToIdleSeconds="120"
            timeToLiveSeconds="120"
            maxElementsOnDisk="10000000"
            diskExpiryThreadIntervalSeconds="120"
            memoryStoreEvictionPolicy="LRU">
        <persistence strategy="localTempSwap"/>
    </defaultCache>
    <!-- 自定义缓存策略 -->
    <cache name="users"
           maxElementsInMemory="10000"
           eternal="false"
```

```
            timeToIdleSeconds="120"
            timeToLiveSeconds="120"
            maxElementsOnDisk="10000000"
            diskExpiryThreadIntervalSeconds="120"
            memoryStoreEvictionPolicy="LRU">
        <persistence strategy="localTempSwap"/>
    </cache>
</ehcache>
```

在application.properties中关联Ehcache的配置文件

```
# jdbc的相关配置
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/gp?serverTimezone=UTC
spring.datasource.username=root
spring.datasource.password=123456

# 连接池
spring.datasource.type=com.alibaba.druid.pool.DruidDataSource

## mybatis的package别名
mybatis.type-aliases-package=com.gupaoedu.pojo

# 指定MyBatis的映射文件的路径
mybatis.mapper-locations=classpath:mapper/*.xml

# 关联Ehcache的配置文件
spring.cache.ehcache.config=ehcache.xml
```

在需要开启换的位置通过 `Cacheable` 设置

```
    @Override
    public Integer addUser(User user) {
        return mapper.addUser(user);
    }

    @Cacheable(value = {"userQuery"})
    @Override
    public User queryById(Integer id) {
        System.out.println("用户查询: " + id);
        return mapper.queryById(id);
    }

    @Override
    public Integer updateUser(User user) { return mapper.updateUser(user); }
```

单元测试，放开缓存

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
</dependency>
```

```
package com.gupaoedu.test;

import com.gupaoedu.service.IUserService;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.cache.annotation.EnableCaching;
import org.springframework.test.context.junit4.SpringRunner;

/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 * @Description: ${todo}
 * @date 2020/7/29 14:39
 */
@RunWith(SpringRunner.class)
@SpringBootTest
@EnableCaching// 放开缓存
public class Test01 {

    @Autowired
    private IUserService userService;

    @Test
    public void test01(){
        userService.queryById(1);
        userService.queryById(1);
    }
}
```

14.SpringBoot整合SpringDataRedis

14.1 添加相关的依赖

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
<dependency>
    <groupId>redis.clients</groupId>
    <artifactId>jedis</artifactId>
    <version>2.9.0</version>
</dependency>
```

14.2 配置信息

```
spring.redis.jedis.pool.max-idle=10
spring.redis.jedis.pool.min-idle=5
spring.redis.jedis.pool.max-active=20
spring.redis.host=192.168.187.120
spring.redis.port=6379
```

14.3 Redis的配置类

```
package com.gupaoedu.config;

import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.redis.connection.jedis.JedisConnectionFactory;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.data.redis.serializer.StringRedisSerializer;
import redis.clients.jedis.JedisPoolConfig;

/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 * @Description: ${todo}
 * @date 2020/7/29 15:24
 */
@Configuration
public class RedisConfig {

    /**
     * 创建JedisPoolConfig对象
     * @return
     */
    @Bean
    @ConfigurationProperties(prefix = "spring.redis.pool")
    public JedisPoolConfig jedisPoolConfig(){
        JedisPoolConfig config = new JedisPoolConfig();
        System.out.println("默认值: " + config.getMaxIdle());
        System.out.println("默认值: " + config.getMinIdle());
        System.out.println("默认值: " + config.getMaxTotal());
```

```
        return config;
    }

    @Bean
    @ConfigurationProperties(prefix = "spring.redis.pool")
    public JedisConnectionFactory jedisConnectionFactory(JedisPoolConfig config)
{
        JedisConnectionFactory factory = new JedisConnectionFactory();
        factory.setPoolConfig(config);
        return factory;
    }

    @Bean
    public RedisTemplate<String,Object> redisTemplate(JedisConnectionFactory
jedisConnectionFactory){
        RedisTemplate<String,Object> template = new RedisTemplate<>();
        template.setConnectionFactory(jedisConnectionFactory);
        // 设置 key的序列号器
        template.setKeySerializer(new StringRedisSerializer());
        // 设置 value的序列化器
        template.setValueSerializer(new StringRedisSerializer());
        return template;

    }
}
```

14.4 单元测试

```
package com.gupaoedu;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.test.context.junit4.SpringRunner;
import redis.clients.jedis.Jedis;

@RunWith(SpringRunner.class)
@SpringBootTest
public class GpSpringbootRedisDemoApplicationTests {

    @Autowired
    private RedisTemplate<String,Object> template;

    /**
     * 添加一个简单的字符串
     */
    @Test
    public void test01() {
        this.template.opsForValue().set("name","bobo");
    }

}
```

```
package com.gupaoedu;

import com.gupaoedu.pojo.User;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.data.redis.serializer.Jackson2JsonRedisSerializer;
import
org.springframework.data.redis.serializer.JdkSerializationRedisSerializer;
import org.springframework.test.context.junit4.SpringRunner;
import redis.clients.jedis.Jedis;

@RunWith(SpringRunner.class)
@SpringBootTest
public class GpSpringbootRedisDemoApplicationTests {

    @Autowired
    private RedisTemplate<String,Object> template;

    /**
     * 添加一个简单的字符串
     */
    @Test
    public void test01() {
        this.template.opsForValue().set("name","bobo");
    }

    @Test
    public void test02() {
        System.out.println(template.opsForValue().get("name"));
    }

    /**
     * 将User对象序列化为一个字符串存储
     */
    @Test
    public void test03(){
        User user = new User(1,"张三","湖南长沙");
        // 设置序列化器
        template.setValueSerializer(new JdkSerializationRedisSerializer());
        template.opsForValue().set("user",user);

    }

    /**
     * 将Redis中存储的User对象反序列化出来
     */
    @Test
    public void test04(){
        template.setValueSerializer(new JdkSerializationRedisSerializer());
        User user = (User) this.template.opsForValue().get("user");
```

```
        System.out.println(user);
    }

    /**
     * 将User对象转换为JSON对象存储
     */
    @Test
    public void test05(){
        User user = new User(2,"李四","湖南长沙");
        template.setValueSerializer(new Jackson2JsonRedisSerializer<>
(User.class));
        template.opsForValue().set("userJson",user);

    }

    /**
     * 将Redis中存储的JSON数据取出转换为User对象
     */
    @Test
    public void test06(){
        template.setValueSerializer(new Jackson2JsonRedisSerializer<>
(User.class));
        User user = (User) template.opsForValue().get("userJson");
        System.out.println(user);
    }

}
```

15.SpringBoot整合Scheduled

Scheduled定时任务，Spring3.0之后就提供的有

15.1 添加相关的依赖

```
<!-- 添加 Scheduled 坐标 -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context-support</artifactId>
            <version>5.1.7.RELEASE</version>
        </dependency>
```

15.2 创建定时任务的方法

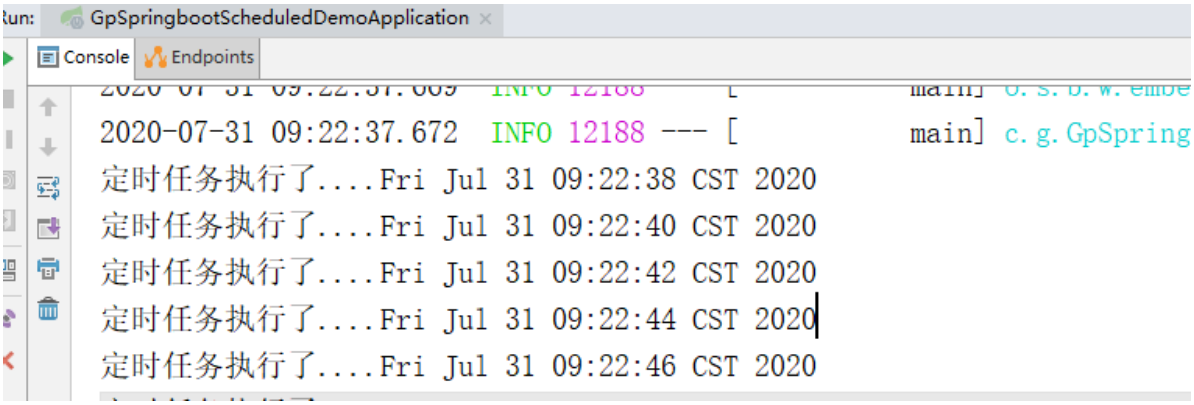
```
package com.gupaoedu.task;

import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Component;

import java.util.Date;

/**
 * 让每一个人的职业生涯不留遗憾
 *
```

```
 * @author 波波老师【咕泡学院】
 * @Description: ${todo}
 * @date 2020/7/31 9:20
 */
@Component
public class MyScheduledTask {

    /**
     * 定时任务的方法
     */
    @Scheduled(cron = "0/2 * * * * ?")
    public void doSome(){
        System.out.println("定时任务执行了...." + new Date());
    }
}
```

15.3 在启动器中放开Scheduled

```
@SpringBootApplication
@EnableScheduling // 放开Scheduled定时任务
public class GpSpringbootScheduledDemoApplication {

    public static void main(String[] args) {
        SpringApplication.run(GpSpringbootScheduledDemoApplication.class, args);
    }

}
```

效果



cron表达式 长度6/7位

> Seconds Minutes Hours Day Month Week Year
>
> Seconds Minutes Hours Day Month Week

例子

> @Scheduled(cron = “0 0 1 1 1 ?”)//每年一月的一号的 1:00:00 执行一次
>
> @Scheduled(cron = “0 0 1 1 1,6 ?”) //一月和六月的一号的 1:00:00 执行一次
>
> @Scheduled(cron = “0 0 1 1 1,4,7,10 ?”) //每个季度的第一个月的一号的 1:00:00 执行一次

> @Scheduled(cron = “0 0 1 1 * ?”)//每月一号 1:00:00 执行一次
>
> @Scheduled(cron=“0 0 1 * * *”) //每天凌晨 1 点执行一次

16.SpringBoot整合Quartz

组成	描述
Job--任务	你要做什么事?
Trigger--触发器	你什么时候去做?
Scheduler--任务调度	你什么时候需要去做什么事情?

16.1 Quartz基本使用

1.依赖

```
<!-- Quartz 坐标 -->
<dependency>
    <groupId>org.quartz-scheduler</groupId>
    <artifactId>quartz</artifactId>
    <version>2.2.1</version>
</dependency>
```

2.创建Job

```
package com.gupaoedu.Job;

import org.quartz.Job;
import org.quartz.JobExecutionContext;
import org.quartz.JobExecutionException;

import java.util.Date;

/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 * @Description: ${todo}
 * @date 2020/7/31 9:33
 */
public class MyJob implements Job {
    /**
     * 自定义的Job
     * @param jobExecutionContext
     * @throws JobExecutionException
     */
    @Override
```

```
    public void execute(JobExecutionContext jobExecutionContext) throws
JobExecutionException {
        System.out.println("quartz任务执行了..." + new Date());
    }
}
```

3.测试

```
package com.gupaoedu.Job;

import org.quartz.*;
import org.quartz.impl.StdSchedulerFactory;

/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 * @Description: ${todo}
 * @date 2020/7/31 9:34
 */
public class JobMain {
    public static void main(String[] args) throws SchedulerException {
        // 1.创建Job对象
        JobDetail job = JobBuilder.newJob(MyJob.class).build();

        // 2.创建Trigger
        Trigger trigger = TriggerBuilder
                .newTrigger()
                .withSchedule(CronScheduleBuilder.cronSchedule("0/2 * * * * ?"))
                .build();
        // 3.创建Scheduler对象
        Scheduler scheduler = StdSchedulerFactory.getDefaultScheduler();
        scheduler.scheduleJob(job,trigger);
        // 启动
        scheduler.start();
    }
}
```

```
09:38:08.001 [DefaultQuartzScheduler_Worker-2] DEBUG org.quartz.core.JobRunShell - Calling execute
09:38:08.001 [DefaultQuartzScheduler_QuartzSchedulerThread] DEBUG org.quartz.core.QuartzSchedulerTh
quartz任务执行了...Fri Jul 31 09:38:08 CST 2020
09:38:08.210 [Timer-0] DEBUG org.quartz.utils.UpdateChecker - Quartz version update check failed: S
09:38:10.001 [DefaultQuartzScheduler_QuartzSchedulerThread] DEBUG org.quartz.simpl.PropertySettingJ
09:38:10.002 [DefaultQuartzScheduler_QuartzSchedulerThread] DEBUG org.quartz.core.QuartzSchedulerTh
09:38:10.002 [DefaultQuartzScheduler_Worker-3] DEBUG org.quartz.core.JobRunShell - Calling execute
quartz任务执行了...Fri Jul 31 09:38:10 CST 2020
09:38:12.001 [DefaultQuartzScheduler_QuartzSchedulerThread] DEBUG org.quartz.simpl.PropertySettingJ
09:38:12.002 [DefaultQuartzScheduler_QuartzSchedulerThread] DEBUG org.quartz.core.QuartzSchedulerTh
09:38:12.002 [DefaultQuartzScheduler_Worker-4] DEBUG org.quartz.core.JobRunShell - Calling execute
quartz任务执行了...Fri Jul 31 09:38:12 CST 2020
09:38:14.001 [DefaultQuartzScheduler_QuartzSchedulerThread] DEBUG org.quartz.simpl.PropertySettingJ
09:38:14.002 [DefaultQuartzScheduler_QuartzSchedulerThread] DEBUG org.quartz.core.QuartzSchedulerTh
09:38:14.002 [DefaultQuartzScheduler_Worker-5] DEBUG org.quartz.core.JobRunShell - Calling execute
quartz任务执行了...Fri Jul 31 09:38:14 CST 2020

Process finished with exit code -1
```

16.2 SpringBoot整合Quartz

添加对应的依赖

```
<!-- Quartz 坐标 -->
    <dependency>
        <groupId>org.quartz-scheduler</groupId>
        <artifactId>quartz</artifactId>
        <version>2.2.1</version>
        <exclusions>
            <exclusion>
                <artifactId>slf4j-api</artifactId>
                <groupId>org.slf4j</groupId>
            </exclusion>
        </exclusions>
    </dependency>
    <!-- 添加 Scheduled 坐标 -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context-support</artifactId>
    </dependency>
    <!-- Sprng tx 坐标 -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-tx</artifactId>
    </dependency>
```

创建对应的Quartz配置类

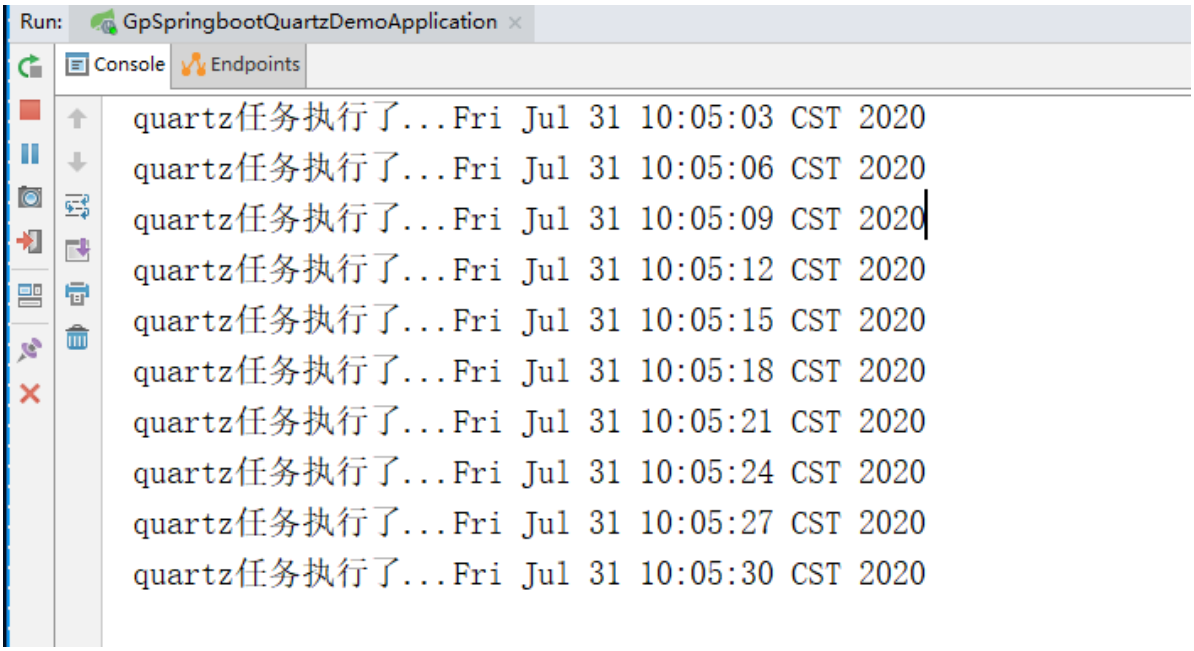
```
package com.gupaoedu.config;

import com.gupaoedu.Job.MyJob;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.scheduling.quartz.CronTriggerFactoryBean;
```

```java
import org.springframework.scheduling.quartz.JobDetailFactoryBean;
import org.springframework.scheduling.quartz.SchedulerFactoryBean;
import org.springframework.scheduling.quartz.SimpleTriggerFactoryBean;

/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 * @Description: ${todo}
 * @date 2020/7/31 9:52
 */
@Configuration
public class QuartzConfig {

    /**
     * 创建Job对象
     * @return
     */
    @Bean
    public JobDetailFactoryBean jobDetailFactoryBean(){
        JobDetailFactoryBean factoryBean = new JobDetailFactoryBean();
        // 关联 Job类
        factoryBean.setJobClass(MyJob.class);
        return factoryBean;
    }

    /**
     * 创建Trigger对象
     * @return
     */
    @Bean
    public SimpleTriggerFactoryBean
simpleTriggerFactoryBean(JobDetailFactoryBean jobDetailFactoryBean){
        SimpleTriggerFactoryBean factoryBean = new SimpleTriggerFactoryBean();
        // 关联JobDetail对象
        factoryBean.setJobDetail(jobDetailFactoryBean.getObject());
        // 设置间隔时间
        factoryBean.setRepeatInterval(2000);
        // 设置重复次数
        factoryBean.setRepeatCount(3);
        return factoryBean;
    }
    /**
     * 创建Trigger对象 Cron表达式
     * @return
     */
    @Bean
    public CronTriggerFactoryBean cronTriggerFactoryBean(JobDetailFactoryBean
jobDetailFactoryBean){
        CronTriggerFactoryBean factoryBean = new CronTriggerFactoryBean();
        factoryBean.setJobDetail(jobDetailFactoryBean.getObject());
        // 设置触发的时间
        factoryBean.setCronExpression("0/3 * * * * ?");
        return factoryBean;
    }

    /**
     * 创建对应的Scheduler对象
```

```
     * @param cronTriggerFactoryBean
     * @return
     */
    @Bean
    public SchedulerFactoryBean schedulerFactoryBean(CronTriggerFactoryBean
cronTriggerFactoryBean){
        SchedulerFactoryBean factoryBean = new SchedulerFactoryBean();
        factoryBean.setTriggers(cronTriggerFactoryBean.getObject());
        return factoryBean;
    }
}
```

启动器中放开

```
package com.gupaoedu;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.scheduling.annotation.EnableScheduling;

@SpringBootApplication
@EnableScheduling
public class GpSpringbootQuartzDemoApplication {

    public static void main(String[] args) {
        SpringApplication.run(GpSpringbootQuartzDemoApplication.class, args);
    }

}
```



17. SpringBoot整合SpringDataJPA

17.1添加依赖

```
<dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>

        <!-- springBoot的启动器 -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>

        <!-- mysql -->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.47</version>
        </dependency>

        <!-- druid连接池 -->
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>druid</artifactId>
            <version>1.0.9</version>
        </dependency>
    </dependencies>
```

17.2 配置文件

```
# jdbc的相关配置
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/gp?serverTimezone=UTC
spring.datasource.username=root
spring.datasource.password=123456

# 配置连接池信息
spring.datasource.type=com.alibaba.druid.pool.DruidDataSource

# 配置JPA的相关参数
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```

17.3 创建POJO对象

```
package com.gupaoedu.pojo;

import javax.persistence.*;

/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 * @Description: ${todo}
 * @date 2020/7/31 10:17
 */
@Entity
@Table(name = "users")
public class User {

    @Id
    @GeneratedValue(strategy =  GenerationType.IDENTITY)
    @Column(name = "id")
    private Integer id;

    @Column(name = "name")
    private String name;

    @Column(name = "age")
    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 Integer getAge() {
        return age;
    }

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

17.4 创建Repository接口

```
package com.gupaoedu.dao;

import com.gupaoedu.pojo.User;
import org.springframework.data.jpa.repository.JpaRepository;

/**
 * 让每一个人的职业生涯不留遗憾
 *   JpaRepository<User,Integer>
 *       User 映射的实体对象
 *       Integer Id的类型
 * @author 波波老师【咕泡学院】
 * @Description: ${todo}
 * @date 2020/7/31 10:19
 */
public interface UserRepository extends JpaRepository<User,Integer> {
}
```

17.5 单元测试

```
package com.gupaoedu;

import com.gupaoedu.dao.UserRepository;
import com.gupaoedu.pojo.User;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit4.SpringRunner;

@RunWith(SpringRunner.class)
@SpringBootTest
public class GpSpringbootJpaDemoApplicationTests {

    @Autowired
    private UserRepository repository;

    @Test
    public void contextLoads() {
        User user = new User();
        user.setName("gupao");
        user.setAge(4);
        repository.save(user);
    }

}
```

GpSpringbootJpaDemoApplicationTests.contextLoads

Tests passed: 1 of 1 test – 375 ms

GpSpringbootJpaDemoApplicationTest 375 ms

contextLoads 375 ms

```
2020-07-31 10:22:16.497  INFO 8248 --- [          main] org.hibernate.dialect.Dialect
2020-07-31 10:22:17.565  INFO 8248 --- [          main] j.LocalContainerEntityManagerFactoryB
2020-07-31 10:22:18.283  INFO 8248 --- [          main] o.s.s.concurrent.ThreadPoolTaskExecut
2020-07-31 10:22:18.359  WARN 8248 --- [          main] aWebConfiguration$JpaWebMvcConfigurat
2020-07-31 10:22:19.157  INFO 8248 --- [          main] c.g.GpSpringbootJpaDemoApplicationTes
Hibernate: insert into users (age, name) values (?, ?)
2020-07-31 10:22:19.574  INFO 8248 --- [      Thread-2] o.s.s.concurrent.ThreadPoolTaskExecut
2020-07-31 10:22:19.575  INFO 8248 --- [      Thread-2] j.LocalContainerEntityManagerFactoryB
2020-07-31 10:22:19.580  INFO 8248 --- [      Thread-2] com.alibaba.druid.pool.DruidDataSourc
```