配置

配置pom.xml

配置spring和springmvc的支持依赖

```
<<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc</artifactId>
        <version>5.1.10.RELEASE</version>
    </dependency>
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>servlet-api</artifactId>
        <version>3.0-alpha-1</version>
        <scope>provided</scope>
    </dependency>
    <!-- 文件上传所需依赖 -->
    <dependency>
        <groupId>commons-fileupload</groupId>
        <artifactId>commons-fileupload</artifactId>
        <version>1.3.3</version>
    </dependency>
</dependencies>
```

配置启动类

```
// 继承后可将DispatcherServlet和Spring应用上下文配置到Servlet容器中
public class WebInitializer extends
AbstractAnnotationConfigDispatcherServletInitializer {

    @Override
    // Spring配置类
    // 在Spring容器中注册
    protected Class<?>[] getRootConfigClasses() {
        return new Class[]{SpringConfig.class};
    }

    @Override
    // SpringMVC配置类
    // 在SpringMVC容器中注册
    protected Class<?>[] getServletConfigClasses() {
        return new Class[]{MvcConfig.class};
    }

    @Override
    // dispatcherServlet的urlpattern
```

```
    // 将DispatcherServlet映射到“/”
    protected String[] getServletMappings() {
        return new String[]{"/"};
    }

}
```

SpringMVC 配置

```
// mvc注解
@EnableWebMvc
// 下面两个基本都是同时出现
// 定义bean
@Configuration
// 指定Spring扫描注解的package
@ComponentScan("com.cskaoyan.homework.controller")
// 实现WebMvcConfigurer后可以在类中直接定义拦截器、转换器等
public class MvcConfig implements WebMvcConfigurer
```

Spring配置

```
@Configuration
// value为扫描注解的package
// excludeFilters: 指定不适合组件扫描的类型
@ComponentScan(value = "com.cskaoyan",excludeFilters = @ComponentScan.Filter(type
= FilterType.ANNOTATION /*按照注解过滤*/,
        value ={org.springframework.stereotype.Controller.class,
                EnableWebMvc.class}))
public class SpringConfig {
}
```

其他组件

Encoding Filter (拦截器)

web.xml中：

```
<filter>
    <filter-name>characterEncodingFilter</filter-name>
    <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-
class>
    <!-- 拦截器中的内容 -->
    <init-param>
        <param-name>encoding</param-name>
        <param-value>utf-8</param-value>
```

```
    </init-param>
</filter>
<filter-mapping>
    <filter-name>characterEncodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

java config中：

```
public class WebInitializer extends
AbstractAnnotationConfigDispatcherServletInitializer {
    @Override
    //拦截器
    protected Filter[] getServletFilters() {
        CharacterEncodingFilter characterEncodingFilter = new
CharacterEncodingFilter();
        characterEncodingFilter.setEncoding("utf-8");
        characterEncodingFilter.setForceEncoding(true);
        return new Filter[]{characterEncodingFilter};
    }
}
```

Converter (转换器)

xml中：

```
<mvc:annotation-driven conversion-service="customConversionService"/>
<bean id="customConversionService"
class="org.springframework.format.support.FormattingConversionServiceFactoryBean">
    <property name="converters">
        <set>
            <ref bean="stringToDateConverter"/>
        </set>
    </property>
</bean>
```

java config中：

```
@Configuration
@EnableWebMvc
@ComponentScan("com.cskaoyan.homework.controller")
public class MvcConfig implements WebMvcConfigurer {

    @Autowired
    // 首先从容器取出原有的conversionService
    ConfigurableConversionService conversionService;
    // 给他增加上我们自定义转换器
```

```
    public void addCustomConverter(){
        // StringToDateConverter为自定义转换器
        conversionService.addConverter(new StringToDateConverter());
    }

    @Bean
    @Primary
    // 重新注册进去
    public ConfigurableConversionService conversionService(){
        return conversionService;
    }
}
```

File upload

xml方式：

```
<bean id="multipartResolver"
class="org.springframework.web.multipart.commons.CommonsMultipartResolver">

</bean>
```

java config中：

```
@Bean("multipartResolver")
public CommonsMultipartResolver multipartResolver() {
    return new CommonsMultipartResolver();
}
```

HandlerInterceptor (拦截器)

```
@Override
public void addInterceptors(InterceptorRegistry registry) {
    // 自定义拦截器，拦截的url为/hello/**
    registry.addInterceptor(new
CustomHandlerInterceptor()).addPathPatterns("/hello/**");
}
```

resources静态资源处理

```
@Override
public void addResourceHandlers(ResourceHandlerRegistry registry) {
    // addResourceHandler()表示需访问的url，addResourceLocations()表示文件所在的位置
    registry.addResourceHandler("/pic/**").addResourceLocations("/WEB-INF/jpg/");
```

```
registry.addResourceHandler("/pic2/**").addResourceLocations("classpath:/picture/"
);

registry.addResourceHandler("/pic3/**").addResourceLocations("file:D:/spring");
}
```

异常处理器

```
// 在config中定义异常处理太麻烦，直接注册会更好
@Bean
public CustomHandlerExceptionResolver handlerExceptionResolver(){
    return new CustomHandlerExceptionResolver();
}
```