



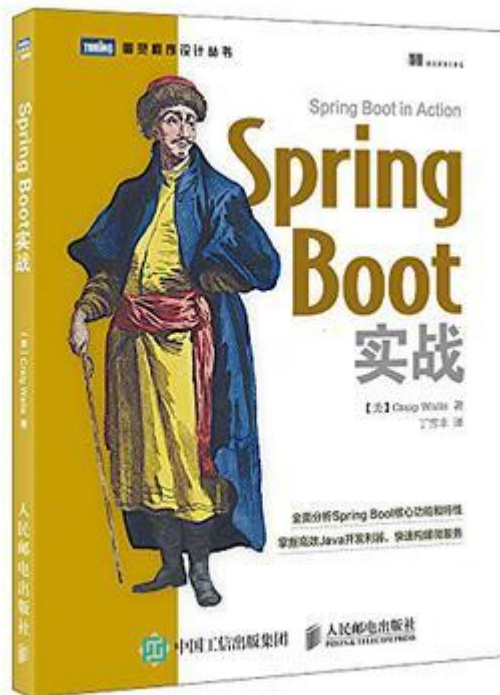
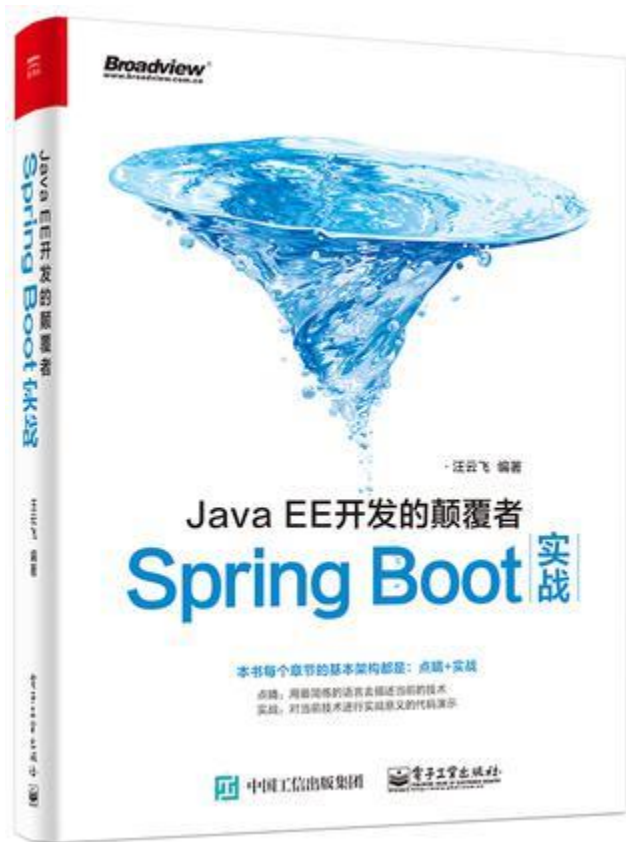
# SpringBoot 1.5.x

---

凯盛软件

# 推荐书籍

凯盛软件



- <http://projects.spring.io/spring-boot/>
- SpringBoot极大简化了基于Spring框架的应用开发，使用很少的代码就可以创建一个独立运行的、准生产基本的基于Spring框架的项目
- 核心功能
  - 独立运行的Spring项目
  - 内嵌Servlet容器
  - 提供starter简化Maven的配置
  - 自动配置Spring
  - 准生产的应用监控
  - 无代码生成和XML配置

- JDK7.0 以上版本
- Spring Framework 4.3.7 以上版本
- Maven 3.2 以上版本

Name	Servlet Version	Java Version
Tomcat 8	3.1	Java 7+
Tomcat 7	3.0	Java 6+
Jetty 9.3	3.1	Java 8+
Jetty 9.2	3.1	Java 7+
Jetty 8	3.0	Java 6+
Undertow 1.3	3.1	Java 7+

可以将SpringBoot项目部署到任何一个支持Servlet3的容器中

- 在线构建
- Maven
- IntelliJ Idea 内置构建工具
- STS构建 (<https://spring.io/tools/sts>)

- <https://start.spring.io/>

Generate a Maven Project with Spring Boot 1.5.2

### Project Metadata

Artifact coordinates

Group

Artifact

### Dependencies

Add Spring Boot Starters and dependencies to your application

Search for dependencies

Selected Dependencies

Web ×JDBC ×MyBatis ×

Generate Project

alt + ↵

Don't know what to look for? Want more options? [Switch to the full version.](#)

- 创建Maven QuickStart项目
- 继承SpringBoot的POM文件

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>1.5.2.RELEASE</version>
</parent>
```

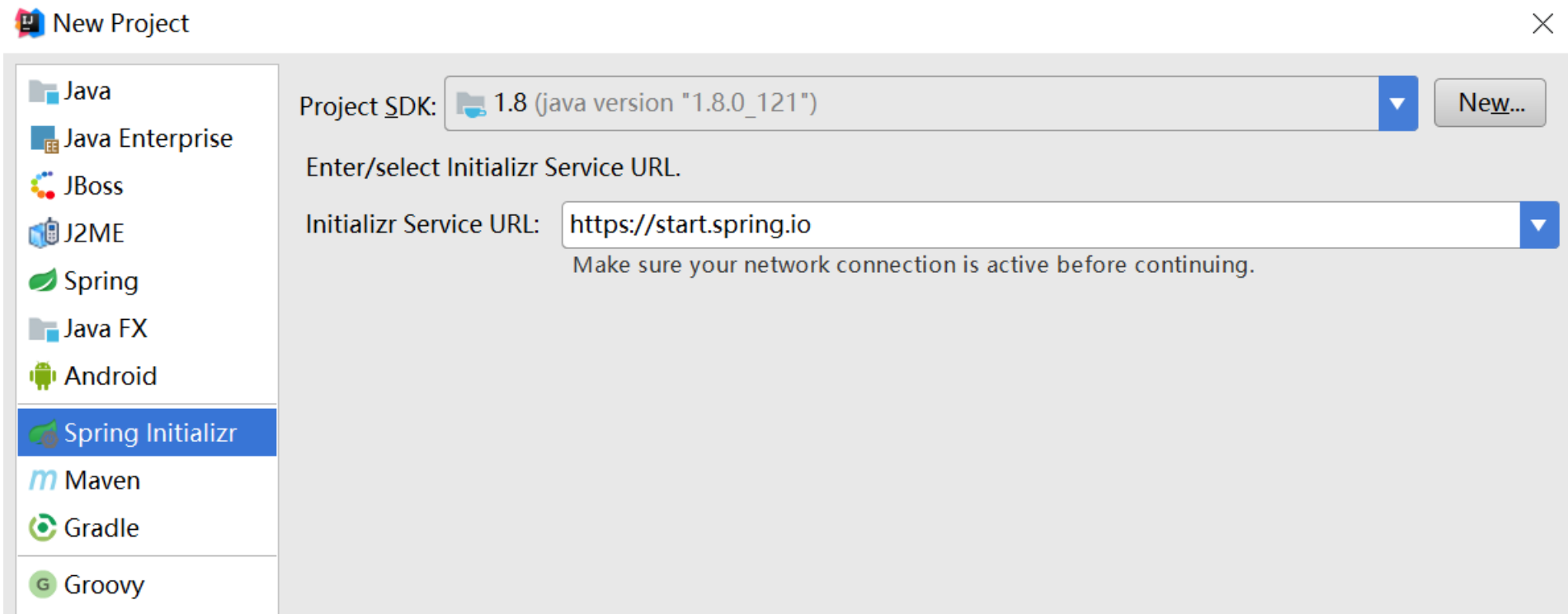
- 添加依赖

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

- 添加插件

```
<plugin>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-maven-plugin</artifactId>
</plugin>
```





**New Spring Starter Project**

Name:

Type:  Packaging:

Java Version:  Language:

Boot Version:

Group:

Artifact:

Version:

Description:

Package:

Dependencies

<input type="checkbox"/> AMQP	<input type="checkbox"/> AOP	<input type="checkbox"/> Actuator	<input type="checkbox"/> Apache Derby
<input type="checkbox"/> Atomikos (JTA)	<input type="checkbox"/> Batch	<input type="checkbox"/> Bitronix (JTA)	<input type="checkbox"/> Cloud Connectors
<input type="checkbox"/> Elasticsearch	<input type="checkbox"/> Facebook	<input type="checkbox"/> Freemarker	<input type="checkbox"/> Gemfire
<input type="checkbox"/> Groovy Templates	<input type="checkbox"/> H2	<input type="checkbox"/> HATEOAS	<input type="checkbox"/> HSQLDB
<input type="checkbox"/> Integration	<input type="checkbox"/> JDBC	<input type="checkbox"/> JMS	<input type="checkbox"/> JPA
<input type="checkbox"/> Jersey (JAX-RS)	<input type="checkbox"/> LinkedIn	<input type="checkbox"/> Mail	<input type="checkbox"/> Mobile
<input type="checkbox"/> MongoDB	<input type="checkbox"/> Mustache	<input type="checkbox"/> MySQL	<input type="checkbox"/> Redis
<input type="checkbox"/> Remote Shell	<input type="checkbox"/> Rest Repositories	<input type="checkbox"/> Security	<input type="checkbox"/> Solr
<input type="checkbox"/> Thymeleaf	<input type="checkbox"/> Twitter	<input type="checkbox"/> Velocity	<input type="checkbox"/> WS
<input checked="" type="checkbox"/> Web	<input type="checkbox"/> Websocket		

Support for full-stack web development, including Tomcat and spring-webmvc

# 第一个SpringBoot应用

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
```

```
@Configuration
```

```
@EnableAutoConfiguration
```

```
@ComponentScan
```

```
public class Application {
```

```
    public static void main(String[] args) {
```

```
        SpringApplication.run(Application.class, args);
```

```
    }
```

```
}
```

- @EnableAutoConfiguration 注解会根据类路径中的jar依赖进行自动配置，例如添加了spring-boot-starter-web，该注解就会自动配置tomcat和SpringMVC
- 每个SpringBoot应用都需要添加@EnableAutoConfiruration、@ComponentScan和@Configuration三个注解，也可以使用@SpringBootApplication注解来替代

@SpringBootApplication

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

- main方法直接运行
- Maven运行
  - mvn spring-boot:run
- 打包运行
  - mvn package
  - java -jar xxx.jar
  - 该运行方式依赖下面的插件

```
<plugin>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-maven-plugin</artifactId>  
</plugin>
```

- starter是一个依赖描述符的集合
- <http://docs.spring.io/spring-boot/docs/1.5.2.RELEASE/reference/htmlsingle/#using-boot-starter>
- <https://github.com/spring-projects/spring-boot/blob/master/spring-boot-starters/README.adoc>

名称	作用
spring-boot-starter-web	用于使用Spring MVC构建web应用，包括 RESTful。 Tomcat是默认的内嵌容器
spring-boot-starter-thymeleaf	用于使用Thymeleaf模板引擎构建MVC web应用
spring-boot-starter-data-redis	用于使用Spring Data Redis和Jedis客户端操作键- 值存储的Redis
spring-boot-starter-test	用于测试Spring Boot应用，支持常用测试类库，包括JUnit, Hamcrest和Mockito

spring-boot-starter-security	对Spring Security的支持
spring-boot-starter-data-jpa	用于使用Hibernate实现Spring Data JPA
spring-boot-starter	核心starter，包括自动配置支持，日志和YAML
spring-boot-starter-cache	用于使用Spring框架的缓存支持
spring-boot-starter-logging	用于使用Logback记录日志，默认的日志starter
spring-boot-starter-aop	用于使用Spring AOP和AspectJ实现面向切面编程
spring-boot-starter-jdbc	对JDBC的支持（使用Tomcat JDBC连接池）

# SpringBoot项目结构

凯盛软件

```
com
+- example
  +- myproject
    +- Application.java
    |
    +- domain
    |   +- Customer.java
    |   +- CustomerRepository.java
    |
    +- service
    |   +- CustomerService.java
    |
    +- web
    |   +- CustomerController.java
```

```
▼ src
  ▼ main
    ▼ java
      ▼ com.kaishengit
        ► controller
        dao
        pojo
        service
        Application
      ▼ resources
        static
        templates
        application.properties
    ► test
```



- tomcat的配置

`server.port=9090`

`server.context-path=`

`server.session.timeout=3600`

- 输出debug级别的日志

`debug=true`

- 字符编码

`spring.http.encoding.charset=UTF-8`

`spring.http.encoding.force=true`

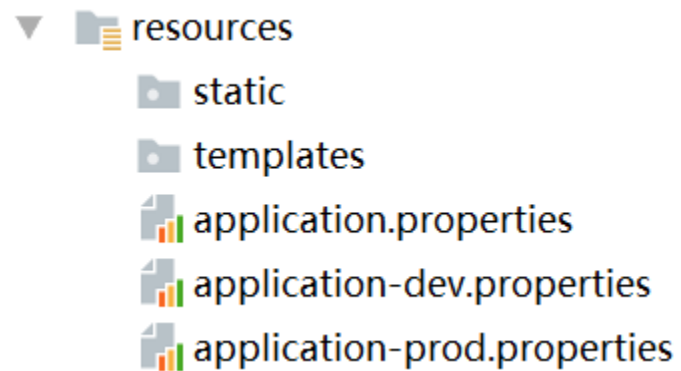
# 根据运行环境选择配置文件

- 使用application-{}.properties命名配置文件
- application.properties文件中的配置是各个环境的通用配置, 在该文件中通过以下配置方式来选择不同配置文件

```
spring.profiles.active=dev
```

该种方式在运行时会加载application-dev.properties文件

- 使用 `java -jar xxx.jar --spring.profiles.active=dev` 方式可以在命令行中选择要加载的配置文件



- 使用SpringMVC作为MVC框架
- classpath中的 /static 文件夹作为静态文件目录
- 使用以下框架作为模板引擎 (View)
  - FreeMarker
  - Groovy
  - Thymeleaf
  - Mustache
- classpath中的 /templates 文件夹作为存放视图的文件夹
- **不支持JSP**

- <http://www.thymeleaf.org/>

`<dependency>`

`<groupId>org.springframework.boot</groupId>`

`<artifactId>spring-boot-starter-thymeleaf</artifactId>`

`</dependency>`

- 默认使用的Thymeleaf版本为2.x，升级到最新的3.x

`<properties>`

`<java.version>1.8</java.version>`

`<thymeleaf.version>3.0.3.RELEASE</thymeleaf.version>`

`<thymeleaf-layout-dialect.version>2.1.1</thymeleaf-layout-dialect.version>`

`</properties>`



```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <h1 th:text="${name}">Hello, Spring Boot</h1>
</body>
</html>
```

*#禁用缓存*

```
spring.thymeleaf.cache=false
```

- 获取值

```
<h1 th:text="${name}">Hello, Spring Boot</h1>
```

```
<h1 th:text="${user.score}"></h1>
```

```
<h1 th:text="${session.sid}"></h1>
```

```
<h1 th:text="${param.q}">none</h1>
```

- 条件判断

```
<h1 th:if="${user.name == 'Rose'}">Hello, Rose</h1>
```

```
<h1 th:unless="${user.name == 'Jack'}">Hello, Jack</h1>
```

```
<ul th:switch="${user.name}">
```

```
    <li th:case="'Rose'">Rose</li>
```


```
    <li th:case="'Jack'">Jack</li>
```

```
    <li th:case="*">other</li>
```

```
</ul>
```

- 循环

```
<ul>  
  <li th:each="user : ${userList}" th:text="${user.name}"></li>  
</ul>
```

A red curved arrow originates from the 'user' part of the 'th:each="user : \${userList}"' attribute and points to the 'user' part of the 'th:text="\${user.name}"' attribute, illustrating the variable resolution in the loop.

```
<ul>  
  <li th:each="user,iterStat : ${userList}" th:text="${iterStat.count} + ' ' +user.name"  
      th:class="${iterStat.odd ? 'odd' : 'even'}"></li>  
</ul>
```

- 表单值

```
<input type="text" th:value="${user.name}">
```

- HTML属性

```
<span th:attr="data-id=${user.id}"></span>
```

```
<span th:attr="data-id=${user.id},data-name=${user.name},data-score=${user.score}"></span>
```

- 超链接

- /user?id=1001

```
<a th:href="@{/user(id=${user.id})}" th:text="${user.name}"></a>
```

- /user/1001/show

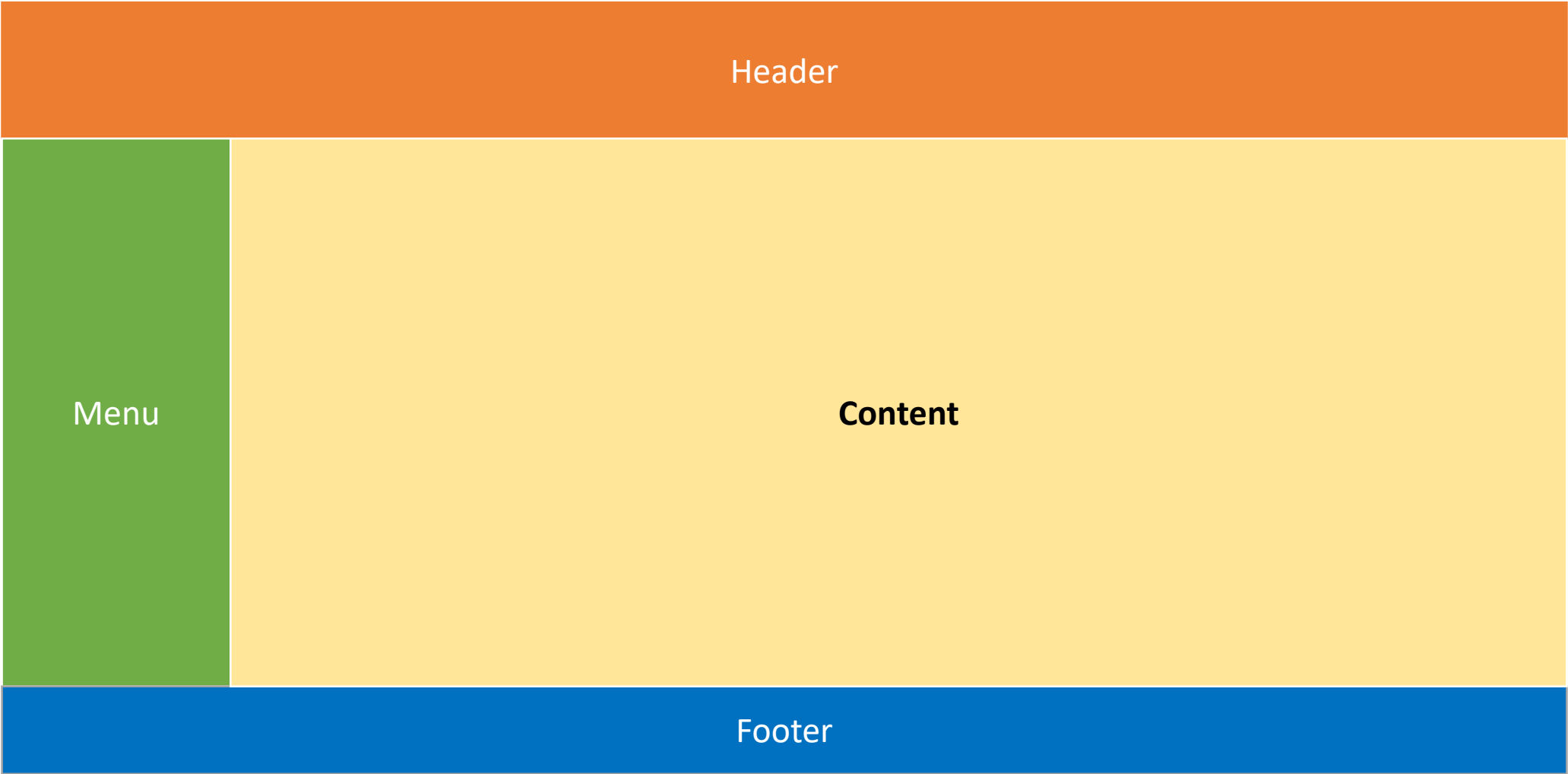
```
<a th:href="@{/user/{userId}/show(userId=${user.id})}" th:text="${user.name}"></a>
```



- 在JavaScript中使用

```
<script th:inline="javascript">
    var userId = [[${user.id}]];
    var userName = [[${user.name}]];
</script>
```

```
<script th:inline="javascript">
    [# th:if="${user.name != null}" ] Thymeleaf 3.x以上才有此特性
    var userId = [[${user.id}]];
    var userName = [[${user.name}]];
    [/]
</script>
```



- footer.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <div th:fragment="copy">
        @凯盛软件
    </div>
</body>
</html>
```

index.html

```
<div th:insert="~{footer::copy}"></div>
<div th:replace="~{footer::copy}"></div>
<div th:include="~{footer::copy}"></div>
```

- th:insert 用于将copy内容(含div)插入到当前div中
- th:replace 用于将copy内容替换当前div
- th:include 用于将copy内容(不含div)插入到当前div中
- ~{}可以省略,  
直接写th:insert="footer::copy"

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <div id="copy">
        @凯盛软件
    </div>
</body>
</html>
```

```
<div th:insert=~{footer::#copy}></div>
<div th:replace=~{footer::#copy}></div>
<div th:include=~{footer::#copy}></div>
```

- 参数传递

```
<div th:fragment="copy(name,year)">
```

```
    @ <span th:text="${name}+'-'+${year}"></span>
```

```
</div>
```

```
<div th:replace="footer::copy('kaishengit',2017)"></div>
```

```
<div th:insert="footer::copy(name=${user.name},year=${user.id})"></div>
```

# Thymeleaf Layout Dialect

---

凯盛软件

- <https://github.com/ultraq/thymeleaf-layout-dialect>
- <https://ultraq.github.io/thymeleaf-layout-dialect/>

**<dependency>**

**<groupId>**nz.net.ultraq.thymeleaf**</groupId>**

**<artifactId>**thymeleaf-layout-dialect**</artifactId>**

**<version>**2.2.0**</version>**

**</dependency>**

- 模板页 layout.html

```
<!DOCTYPE html>
<html xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>
  <h1>Header</h1>
  <div layout:fragment="content">
    <!-- 动态内容 -->
  </div>
  <h3>
    Footer
    <p layout:fragment="footer"></p>
  </h3>
</body>
</html>
```

- 使用模板页 index.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org"
      xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
      layout:decorate="~{layout}">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>
  <div layout:fragment="content">
    这是首页
  </div>
  <h3>
    <p layout:fragment="footer">@2017</p>
  </h3>
</body>
</html>
```



- 页面标题 title 的设定

- 模板页

```
<head>  
    <meta charset="UTF-8">  
    <title layout:title-pattern="$LAYOUT_TITLE - $CONTENT_TITLE">CRM系统</title>  
</head>
```

- 内容页

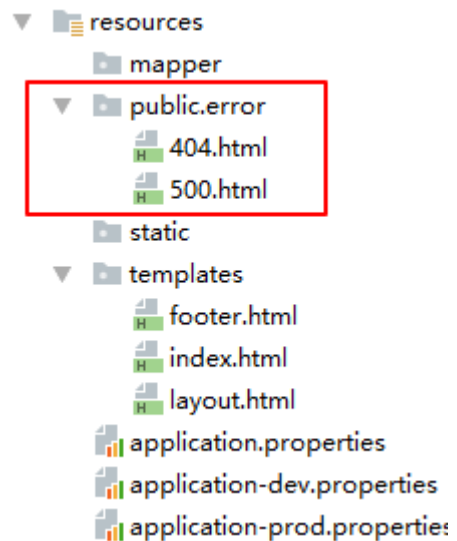
```
<head>  
    <meta charset="UTF-8">  
    <title>系统首页</title>  
</head>
```

- 结果

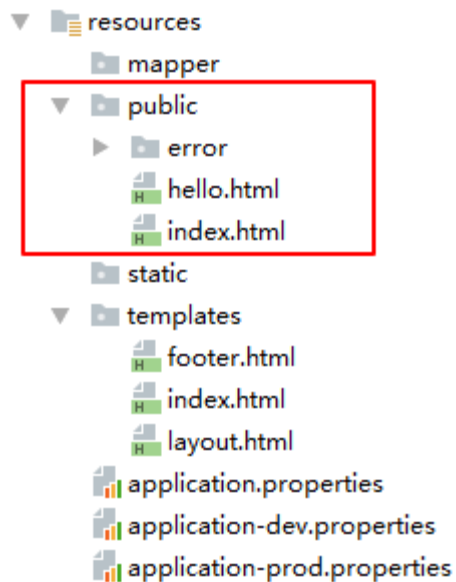
```
<title>CRM系统 - 系统首页</title>
```

# 定义错误页面

- 将错误页放到public/error/文件夹中



- 静态页面放到public文件夹中，可以通过客户端直接访问



- 当服务器端没有定义 / 的控制器是，访问 localhost:8080 即可访问 index.html
- 客户端访问 localhost:8080/hello.html 可以访问 hello.html 页面

- SpringBoot推荐使用以下连接池
  - Tomcat数据库连接池
  - HikariCP
  - Commons DBCP2

*#database*

**spring.datasource.driver-class-name=com.mysql.jdbc.Driver**

**spring.datasource.url=jdbc:mysql:///mydb**

**spring.datasource.username=root**

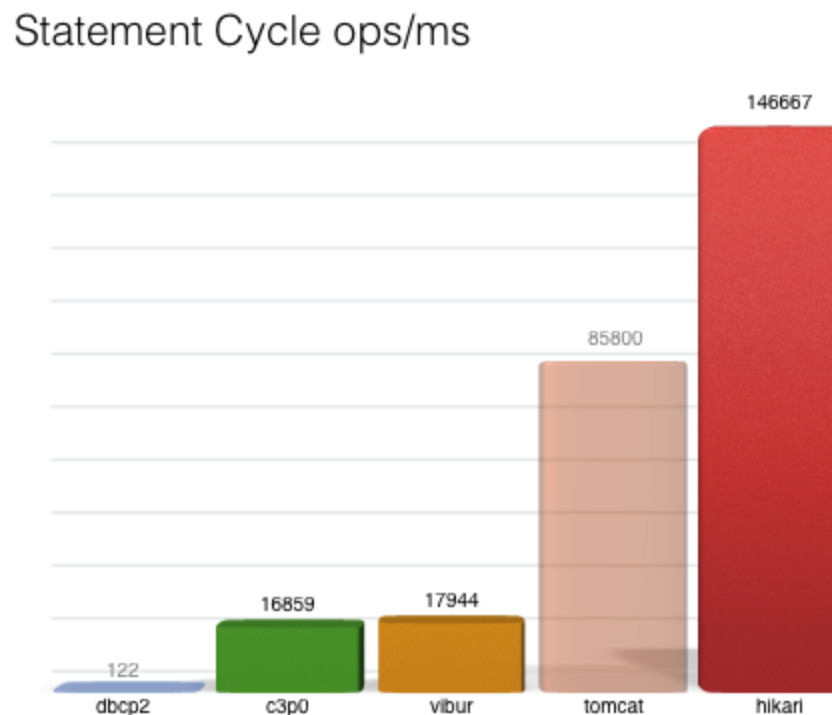
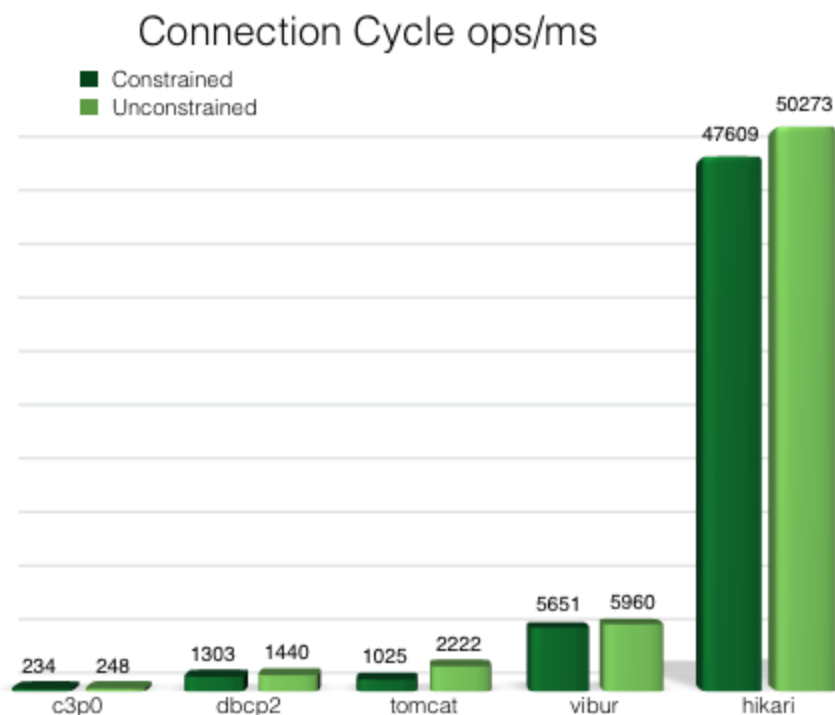
**spring.datasource.password=rootroot**

*#tomcat datasource*

**spring.datasource.tomcat.max-active=20**

**spring.datasource.tomcat.max-wait=5000**

- <https://github.com/brettwooldridge/HikariCP>
- 号称Java世界里面性能最好的数据库连接池



*#database*

spring.datasource.driver-class-name=com.mysql.jdbc.Driver

spring.datasource.url=jdbc:mysql:///mysql

spring.datasource.username=root

spring.datasource.password=rootroot

spring.datasource.type=com.zaxxer.hikari.HikariDataSource

*#hikari datasource*

spring.datasource.hikari.maximum-pool-size=20

spring.datasource.hikari.minimum-idle=5

- 添加JDBC Starter

```
<dependency>
```

```
    <groupId>org.springframework.boot</groupId>
```

```
    <artifactId>spring-boot-starter-jdbc</artifactId>
```

```
</dependency>
```

- 使用

```
@Repository
```

```
public class UserDao {
```

```
    @Autowired
```

```
    private JdbcTemplate jdbcTemplate;
```

- <https://github.com/mybatis/spring-boot-starter>

- 添加MyBatis Starter

**<dependency>**

**<groupId>**org.mybatis.spring.boot**</groupId>**

**<artifactId>**mybatis-spring-boot-starter**</artifactId>**

**<version>**1.2.0**</version>**

**</dependency>**

- 配置

*#mybatis*

**mybatis.type-aliases-package=**com.kaishengit.pojo

**mybatis.configuration.map-underscore-to-camel-case=**true

**mybatis.mapper-locations=**classpath:mapper/\*.xml



@Mapper

public interface UserMapper {

@Select("select \* from t\_user")

List<User> findAll();

}

#配置SQL 日志输出

logging.level.com.kaishengit.dao.UserMapper=debug

- Spring Boot在所有内部日志中使用Commons Logging，但是默认配置也提供了对常用日志的支持，如：Java Util Logging，Log4J（SpringBoot1.4.x起已不支持），Log4J2和Logback。每种Logger都可以通过配置使用控制台或者文件输出日志内容
- 在Spring Boot中默认配置了ERROR、WARN和INFO级别的日志输出到控制台
- 切换到debug的两种方式
  - 在配置文件中添加debug=true，配置后核心库(Spring、Hibernate、web容器)会输出debug级别的日志，但是自己的类不会输出
  - 自己的类通过logging.level.xxx=debug的方式将debug日志输出，例如logging.level.com.kaishengit=debug，是将com.kaishengit包中所有类以debug级别输出
  - 在运行时添加--debug参数，例如java -jar app.java --debug

- SpringBoot的日志默认只会输出到控制台上，如果要输出到文件中，需要指定logging.file或logging.path两个配置项

*#文件路径，可以是绝对路径也可以是相对路径*

**logging.file=D:/my.log**

*#指定日志存放目录，在当前目录中产生名称spring.log*

**logging.path=D:/log**

- 日志文件按照大小自动分割，当超过10M时会自动产生一个新的日志文件

- SpringBoot支持自定义日志系统，推荐使用带有-spring的文件名做为日志的配置文件，例如logback-spring.xml
- logback-spring.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <include resource="org/springframework/boot/logging/logback/base.xml"/>
    <logger name="org.springframework.web" level="INFO"/>
    <logger name="com.kaishengit" level="DEBUG"/>
</configuration>
```

- base.xml中配置有将日志同时输出到控制台和文件中(10M分割)，同时会自动读取application.properties配置文件中的logging.file和logging.path配置，用来将文件输出到指定位置。如果未指定，则输出到系统的临时文件夹中，文件名称为spring.log