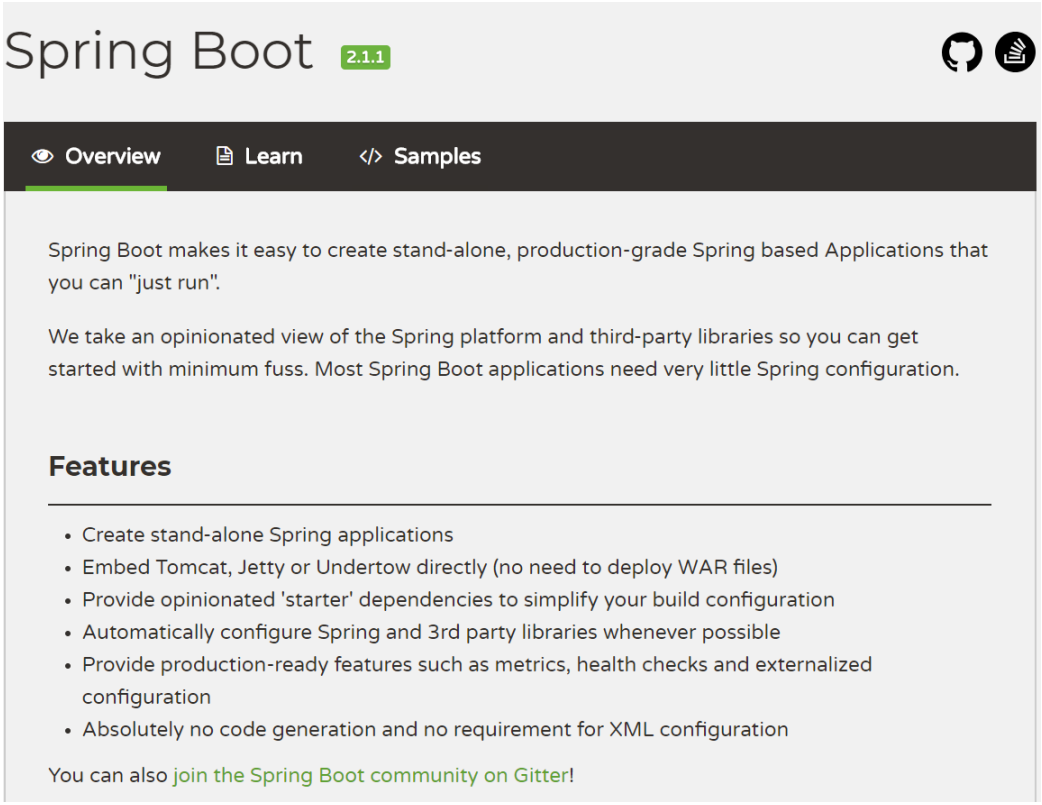
05_SpringBoot基础教程

- 1，什么是SpringBoot
  - Spring Boot是由Pivotal团队提供的全新框架，其设计目的是用来简化新Spring应用的初始搭建以及开发过程。
  - 简化搭建Spring项目的流程
  - 提供统一的父类工程，管理常见的第三方组件
  -



- 2，Eclipse开发SpringBoot
  - 开发第一个应用
    - 进入http://start.spring.io/中下载springboot的maven工程
    - http://start.spring.io/
    - 在pom.xml中加入依赖：
      -

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>2.0.6.RELEASE</version>
  <relativePath/> <!-- lookup parent from repository -->
</parent>
```

      -

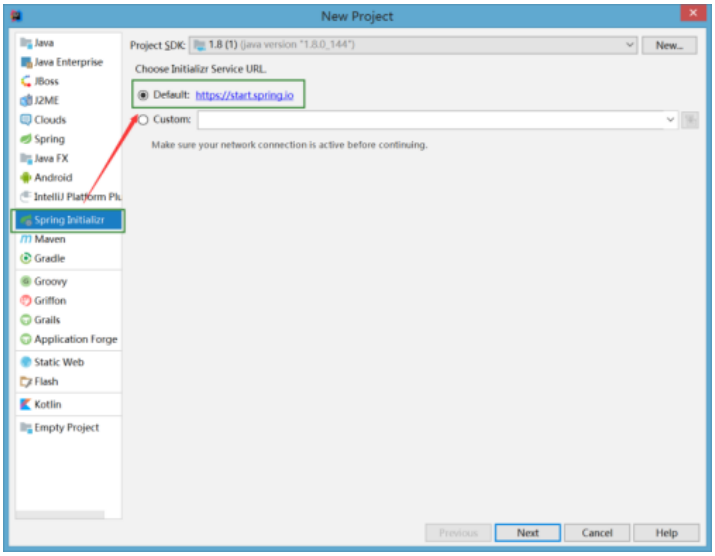
```
<dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

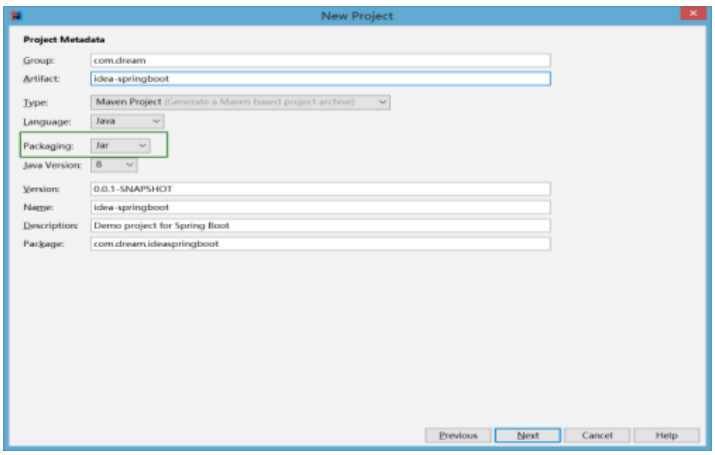
    - 创建Controller，加上相关注解

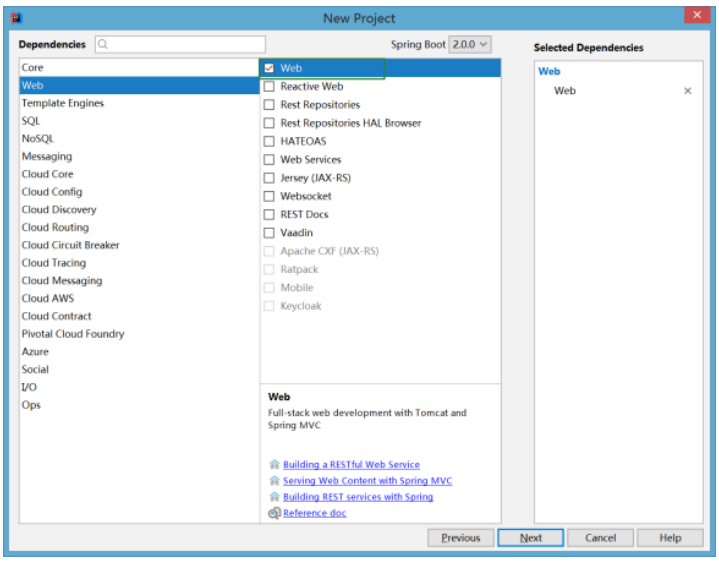
        - 启动Application
        - 访问服务
    - 配置热部署
        - 添加依赖
            -

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

- 3，IDEA开发SpringBoot
    - 创建SpringBoot项目







- 配置热部署
  - idea还需要额外做两个设置
    打开自动编译：
    File -> Settings -> Build,Execution,Deployment->Compiler下，
    打开Build project automatically选项

    打开运行时编译：
    按快捷键 Shift+Ctrl+Alt+/ ，选择 Registr
    打开
    compiler.automake.allow.when.app.running选项

- 4，配置application.properpties
  - 配置参考案例：
    - ```
      server.port=8081
      server.context-path=/springboot

      #设置对输入参数的格式化
      spring.mvc.date-format=yyyy-MM-dd
      #北京时间相对伦敦有8个小时时差所以使用GMT+8
      spring.jackson.time-zone=GMT+8
      #设置对输出参数的格式化
      spring.jackson.date-format=yyyy-MM-dd HH:mm:ss

      #fastdfs
      images.serverpath = http://images.qf.com
      ```
  - 通过@Value获取属性文件的值
    - ```
      @Value("${images.serverpath}")
      private String imageServer;
      ```

- 5，日志的配置：
  - 日志框架的作用
    -

日志

1，调试
System.out.println("start....");
System.out.println("end....");

人工维护成本高

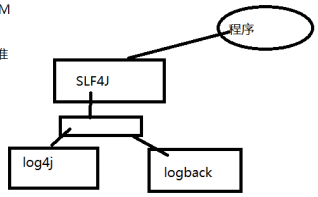
2，日志框架
log4j for java
日志级别：debug info warn error
logger.debug("start.....");
logger.error("error.....");
开发阶段：debug
上线阶段：error

配置文件：调整日志的输出级别即可
debug---->error----->debug

日志框架

1，日志框架会有很多
2，很多第三方框架，比如spring，hibernate他们会用到日志框架也有差异
SSM

标准



springboot默认整合好了

日志的作用？

开发阶段
A

上线了之后。。。。。
A
看服务器的环境的运行情况--看日志

日志输入的目的地：
控制台
文件

- 常规的设置
  - #日志的设置
    - logging.file=d://logs//all.log
  - #日志级别 debug/info/warn/error
  - #默认的日志级别为info
  - #设置全局的输入级别，root代表全局
    - logging.level.root=warn
- 通过配置文件，更加精细化控制，实现每天自动产生一个日志文件
  - 直接将logback.xml导入到resources中
  - 然后将之前的配置都注释掉
  - SpringBoot会自动采用logback中的配置
  - 配置
    -

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <!--1，定义日志保存的路径-->
    <property name="LOG_HOME" value="d://logs//" />
    <!--2，定义一个控制台输出器，名为console-->
    <appender name="console"
class="ch.qos.logback.core.ConsoleAppender">
        <!--按pattern指定的格式输出日志，编码为UTF-8-->
        <encoder
class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
            <pattern>%d{yyyy-MM-dd HH:mm:ss} %-5level [%thread]
 %logger{30} - %msg%n</pattern>
            <charset>UTF-8</charset>
        </encoder>
    </appender>
    <!--3，定义一个日滚动的日志文件-->
    <appender name="file"
class="ch.qos.logback.core.rolling.RollingFileAppender">
        <!--按pattern指定的格式输出日志，编码为UTF-8-->
        <encoder
class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
            <pattern>%d{yyyy-MM-dd HH:mm:ss} %-5level [%thread]
 %logger{30} - %msg%n</pattern>
            <charset>UTF-8</charset>
        </encoder>
        <!-- 定义保存的文件名 -->
        <rollingPolicy
class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
```

```
            <fileNamePattern>${LOG_HOME}/sprintboot_%d{yyyy-MM-dd}.log</fileNamePattern>
            <!--日志最多保存30天-->
            <maxHistory>30</maxHistory>
        </rollingPolicy>
    </appender>
    <!-- 定义日志全局最低输出级别是INFO，同时向控制台和日滚动文件输出 -->
    <root level="INFO">
        <appender-ref ref="console" />
        <appender-ref ref="file" />
    </root>
</configuration>
```

- 程序里面输入日志信息
  - private Logger logger = LoggerFactory.getLogger(MyController.class);
  - logger.debug("debug.....");
  - logger.info("info.....");
  - logger.warn("warn.....");
  - logger.error("error.....");
- 6，配置application.yml
  - 推荐的新配置方式
    - 层次感更好

```
server:
  port: 8899
  servlet:
    context-path: /haha
```

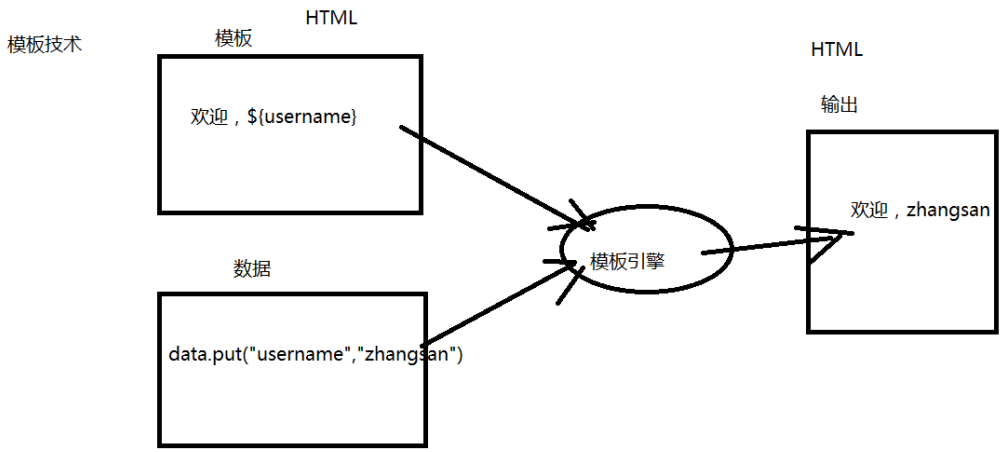
  - 多环境配置，方便动态切换
    - 根据不同的环境编写多套配置，application-develop.yml和application-producer.yml
    - 在主配置中（application.yml）编写如下代码，来实现环境的选择
      -

```
spring:
  profiles:
    active: develop
```

    - **注意，要让日志也支持环境的切换可以这么设置**
      - 1，更改默认的logback.xml为logback-spring.xml
      - 2，SpringBoot当看到logback-spring.xml文件存在的时候，才会启动日志的环境切换
      - 3，在配置文件中，增加springProfile标签
        -

```
<springProfile name="develop">
            <property name="LOG_HOME" value="d://logs//dev" />
        </springProfile>
        <springProfile name="producer">
            <property name="LOG_HOME" value="d://logs//pro" />
        </springProfile>
```

    - 另外，还可以在运行该项目时，自动切换应用的环境版本
      - 首先，打包，其次，在命令行运行该jar包
      - java -jar *.jar --spring.profiles.active=product

- 7，自定义Filter
  - 创建一个Filter
    -
      ```
      class MyFilter implements Filter{
      }
      ```
  - 创建一个注解类，通过注解的方式来配置Filter
    -
      ```
      @Configuration
      public class WebConfig {
          @Bean
          public FilterRegistrationBean init(){
              FilterRegistrationBean filterRegistrationBean = new
      FilterRegistrationBean();
              filterRegistrationBean.setFilter(new MyFilter());
              filterRegistrationBean.setName("MyFilter");
              filterRegistrationBean.addUrlPatterns("/*");
              return filterRegistrationBean;
          }
      }
      ```
- 8，自定义Listener
  - ServletListenerRegistrationBean
- 9，模板引擎-Thymeleaf
  -
    

  - 之前学的视图技术
    - HTML
    - JSP
      - 运行原理：JSP->翻译->Java(Servlet)->编译->class
      - 所以，JSP第一次运行的时候会比较慢，会经历两个步骤，将JSP转换为class
      - 之后，访问的都是class文件（除非修改JSP源文件）
      - class文件再经过tomcat解析成最终的HTML+CSS
  - 什么是thymeleaf
    - Thymeleaf 是一个跟 Velocity、FreeMarker 类似的模板引擎
    - 相较与其他的模板引擎，它有一个最大的特点是：

- Thymeleaf，它可以让美工在浏览器查看页面的静态效果，也可以让程序员在服务器查看带数据的动态页面效果。
  - 这是由于它支持 html 原型，然后在 html 标签里增加额外的属性来达到模板+数据的展示方式。
    - <a th:text="${url}">百度
  - 浏览器解释 html 时会忽略未定义的标签属性，所以 thymeleaf 的模板可以静态地运行；
  - 当有数据返回到页面时，Thymeleaf 标签会动态地替换掉静态内容，使页面动态显示。
- 引入依赖
  -
```
<dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```
- 配置thymeleaf模板参数
  -
```
spring:
  thymeleaf:
    cache: false
```
- 创建模板文件
  - 文件存放的位置：resource目录下创建templates目录
  - 创建文件,<html xmlns:th="http://www.thymeleaf.org">
    -
```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8"></meta>
<title>Insert title here</title>
</head>
<body>
      Hello Spring Boot!!!!
</body>
</html>
```
- 常用的各种传参及标签
  - 1，后台程序，依然通过model来保存值即可
  - 2，前端页面，引入标签声明：<html xmlns:th="http://www.thymeleaf.org">
    - 字符串
      - 欢迎:
      - 欢迎:
      - 欢迎:
    - 条件判断
      - 条件为真，则显示
        - 所处的年龄段
        - <span th:if="${age > 18}">老腊肉
        - <span th:if="${age <= 18}">小鲜肉
      - 条件不为真，则显示

- <span th:unless="${age>18}" th:text="未超过18岁">
- 三元运算符
  - <span th:text="${age>18？'不年轻了':'too yong'}">
- 循环 th:each="stu : ${stus}"
  -
```
<table>
    <tr>
        <td>id</td>
        <td>姓名</td>
    </tr>
    <tr th:each="stu : ${stus}">
        <td th:text="${stu.id}">id</td>
        <td th:text="${stu.name}">姓名</td>
    </tr>
</table>
```
- 日期格式化
  - <input th:value="${#dates.format(now,'yyyy-MM-dd HH:mm:ss')}"/>

- 页面读取变量的时候，会出现红色
  - 解决：File -> Settings -> Editor -> Inspections找到Thmeleaf，将后边的√取消选中就可以了。
- 如果发现页面，总是无法自动刷新，按Ctrl+F9刷新源文件

- 10，整合持久层
  - 整合MyBatis
    - 引入依赖（注意，依然是MyBatis提供的整合包，所以版本这一块需要自己来控制）
      -
```
<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>
```
    - 配置文件，配置数据库连接信息
      -
```
spring:
  datasource:
    url: jdbc:mysql://localhost:3306/qfb2c?useUnicode=true&characterEncoding=utf-8
    username: root
    password: root
    driver-class-name: com.mysql.jdbc.Driver
```
    - 运行机制：springboot会自动加载spring.datasource.*相关配置，数据源就会自动注入到sqlSessionFactory中，sqlSessionFactory会自动注入到Mapper中
    - 基于XML的方式做整合(推荐)
      - entity，mapper, mapper.xml依然需要创建，跟之前无差异

        - 在启动类加上 @MapperScan("com.qianfeng.springbootmybatisxml.mapper")
    - 基于注解的方式做整合（了解即可）
        - entity，mapper（接口+注解）
            -
```
public interface StudentMapper {
    @Select("select id,name from student")
    @Results({
            @Result(property = "name",column = "name")
    })
    public List<Student> findAll();
    @Select("select * from student where id=#{id}")
    public Student findOne(Integer id);
    @Insert("insert into student(name) values(#{name})")
    public Long save(Student student);
    @Update("update student set name=#{name} where id=#{id}")
    public Long update(Student student);
    @Delete("delete from student where id=#{id}")
    public Long delete(Integer id);
}
```
        - 在启动类加上 @MapperScan("com.qianfeng.springbootmybatisxml.mapper")
    - SpringBoot做单元测试
    - 总结：
        - SpringBoot整合MyBatis只是简化了配置，但是其他业务相关的类，配置文件还是得自己来写
        - MyBatis采用的是XML或者注解的方式，SpringBoot在整合这块的操作无差异
        - 注解=接口+XML
- 整合JPA（Hibernate，SpringData（SpringDataJPA））
    -
    - 添加依赖
        -
```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
</dependency>
```
    - 添加数据库配置
        -
```
spring:
  datasource:
    url: jdbc:mysql://localhost:3306/qfb2c?
useUnicode=true&characterEncoding=utf-8
    username: root
    password: root
    driver-class-name: com.mysql.jdbc.Driver
```

```
    jpa:
      hibernate:
        ddl-auto: update
      show-sql: true
```

- 创建entity，添加注解
  -
    ```
    @Entity
    public class Student {
        @Id
        @GeneratedValue
        private Integer id;
        @Column(nullable = false,unique = true)
        private String name;
    ```
- 创建持久层操作接口
  -
    ```
    public interface StudentRepository extends
    JpaRepository<Student,Integer>{
    }
    ```
- 在SpringBoot的单元测试中，注入接口类型对象，并测试