## 培训总结

第一周的技术培训完成了一个基于ssm框架的员工管理系统，使我对ssm框架的理解有了巨大的提升，能独立的担任开发任务。

第二周，中心的北极星培训，使我对行内的开发流程、开发工具有了一定的认识，对技术流程有个概况把握。

第三周组内的培训，更是加快了我对springboot的熟悉和理解，基于springboot框架独自进行后端的开发，并且熟悉了k12的项目概况。

## 生成时间戳

1、System.*currentTimeMillis*()

2、new Date().getTime()

## @RequestParam和@RequestBody和@PathVariable

1、使用@PathVariable接收参数，参数值需要在url进行占位

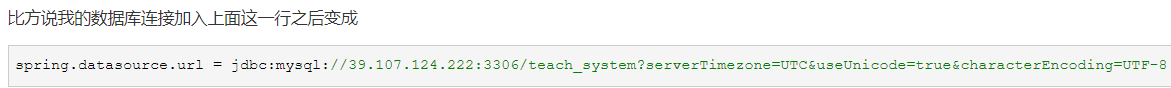
2、注解@RequestParam接收的参数是来自requestHeader中，即请求头。

3、注解@RequestBody接收的参数是来自requestBody中，即请求体。一般用于处理非Content-Type: application/x-www-form-urlencoded编码格式的数据，比如：application/json、application/xml等类型的数据。

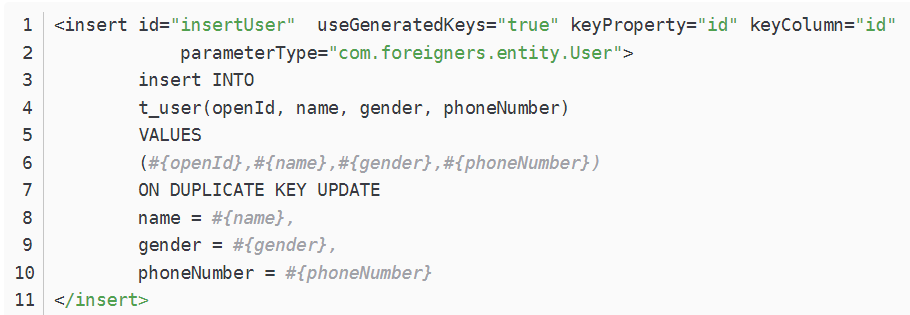
## springboot写入数据库汉字变问号？？？

在数据库连接中加入：





## Mybatis 有则更新，无则插入的实现



使用ON DUPLICATE KEY UPDATE。其中insert中的一些属性参数可以不要。

## springboot的搭建

**springboot的优势：**

* 继承了 Spring 框架自身优秀的特性。
* 能够让配置更简单，通过自动配置快速搭建 Spring 应用项目。
* 能够以 jar 文件形式独立运行项目。
* 更强大的注解，让业务实现更简化。
* 内嵌了常见的 Web 服务器，让你随时使用 Tomcat、Jetty 等。
* 提供了企业生产级项目的服务监控方案，让监控更加简单。
* 还有一些非功能性的通用配置，适应更多的开发需求。
* 可进行分布式开发，结合 Spring Cloud 进行微服务开发。

**搭建springboot项目：**

1.创建maven项目

2.创建好之后需要添加配置和代码

首先，使用 <parent> 标签指定  spring-boot-starter-parent 依赖模块，单从标签名字上来看，就知道它想指定（继承）父类级别的东西。

在构建 Spring Boot 应用时设置它，也就意味着会自动包含能简化我们工作的自动配置、日志和 YAML 等大量的配置。它作为 Spring Boot 的核心启动器，提供一些 Maven 默认的配置和 dependency-management，能让你快速使用 Spring Boot 进行开发。

<parent>

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

    <artifactId>spring-boot-starter-parent</artifactId>

    <version>2.2.1.RELEASE</version>

</parent>

如果，我们需要进行 Web 开发，还需要指定 spring-boot-starter-web 依赖模块，需要在 pom.xml 文件中添加 <dependency…/> 元素来进行指定，具体如下：

<dependency>

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

    <artifactId>spring-boot-starter-web</artifactId>

</dependency>

最后，还可以通过 <build …/> 元素来添加插件，比如设置 Spring Boot 的 Maven 插件 Spring Boot Maven plugin，它能够为 Spring Boot 应用提供执行 Maven 操作。此插件，能够将 Spring Boot 应用进行打包为可执行的 jar 或 war 形式的文件，然后以通常的方式来运行。具体配置如下：

<build>

    <plugins>

        <plugin>

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

            <artifactId>spring-boot-maven-plugin</artifactId>

        </plugin>

    </plugins>

</build>

完整的 pom.xml 文件，具体如下：

<?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.nx</groupId>

    <artifactId>SpringDemo</artifactId>

    <version>1.0-SNAPSHOT</version>

    <!-- 指定 parent 版本 -->

    <parent>

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

        <artifactId>spring-boot-starter-parent</artifactId>

        <version>2.2.1.RELEASE</version>

    </parent>

    <!-- 指定 web 依赖模块 -->

    <dependencies>

        <dependency>

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

            <artifactId>spring-boot-starter-web</artifactId>

        </dependency>

    </dependencies>

    <!-- 设置 maven 插件 -->

    <build>

        <plugins>

            <plugin>

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

              <artifactId>spring-boot-maven-plugin</artifactId>

            </plugin>

        </plugins>

    </build>

</project>

3.添加Controller类

创建包，并创建控制类

package com.nx.controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

@RestController

public class HelloController {

    @RequestMapping("/hello")

    public String sayHello(){

        return "I Love Spring Boot.";

    }

}

@RestController注解是一个组合注解，它包含了@Controller和@ResponseBody两个注解，说明它能以 JSON 格式进行响应数据。注解的源码如下：

4.添加springboot启动类

package com.nx;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

// 用于指定 Spring Boot 应用的启动类

@SpringBootApplication

public class SpringBootApp {

    public static void main(String[] args) {

        // 从 main 方法中进行启动 Spring 应用中的类

        SpringApplication.run(SpringBootApp.class, args);

    }

}

**启动boot，请参考企业级博客笔记。**

### 注解详解

SpringBoot的启动类，也就是入口类，需要使用@SpringBootApplication注解来标注。在启动类中，我们的 main 方法就是 Java 应用程序的入口方法。

@SpringBootApplication 是一个组合注解，比较重要的三个注解是：@SpringBootConfiguration、@EnableAutoConfiguration、@ComponentScan。

**@SpringBootConfiguration注解**

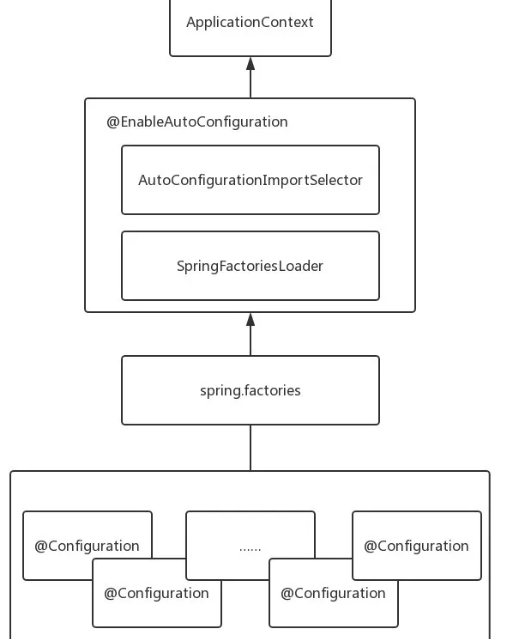
主要是负责 Spring Boot 应用配置相关的注解，它也是组合注解。通过源码，可以看到它也使用了 @Configuration 注解，它们两个都是将当前类标注为配置类，能将类中使用 @Bean 注解标记的方法对应的实例注入到 Spring 容器中，那实例名就是方法名。

另外在 @Configuration 注解源码中，还看到有一个 @Component 注解，做了再次封装，主要是把普通 POJO 实例化到 Spring 容器中。

**@EnableAutoConfiguration注解**

主要用来启动自动配置，Spring Boot 就能够根据依赖信息自动实现应用的相关配置，总体分为两个部分：一是收集所有 spring.factories 中EnableAutoConfiguration 相关 bean 的类，二是将得到的类注册到 Spring 容器中。将符合的配置都加载到 IoC 容器中。

组件调用关系图：



其实是这样的，涉及到了 BeanFactory 的创建。Spring 框架中会调用 ApplicationContext 的 refresh 方法来启动 Spring 容器，然后就会创建 BeanFactory，接着扫描各种包，读取使用到了 @Configuration、@Import、@SpringBootApplication 等注解标注的类，然后生成 BeanDefinition 最终注册到 BeanFactory 中。

然后就交给 BeanFactoryPostProcessor 来执行，BeanFactory 后置处理器会处理 BeanDefinition，比如在 BeanFactoryPostProcessor 接口中，提供了 postProcessBeanFactory 方法来接收 ConfigurableListableBeanFactory 对象来处理。

### 配置文件

**1.关闭指定的自动配置**

通过上面的 @EnableAutoConfiguration 注解就能根据指定的依赖，自动进行配置。但如果你想关闭某一项自动配置，就需要使用 @SpringBootApplication 下的 exclude 参数来设置。比如，我想关闭 DataSource，代码具体如下：





**3.全局配置文件**

一般使用 application.properties 或者 application.yml 文件来当作全局配置文件。它能被添加在下面几个目录下，区别是加载的顺序是不同的。

