# Spring Boot基础知识（核心内容）

## 0.Spring boot 简介（百度百科）

### 1）发展介绍

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。该框架使用了特定的方式来进行配置，从而使开发人员不再需要定义样板化的配置。通过这种方式，Spring Boot致力于在蓬勃发展的快速应用开发领域(rapid application development)成为领导者。

Spring框架最初是没有打算构建一个自己的WebMVC框架，其开发人员在开发过程中认为现有的StrutsWeb框架的呈现层和请求处理层之间以及请求处理层和模型之间的分离不够，于是创建了SpringMVC。

SpringBoot是由Pivotal团队在2013年开始研发、2014年4月发布第一个版本的全新开源的轻量级框架。它基于Spring4.0设计，不仅继承了Spring框架原有的优秀特性，而且还通过简化配置来进一步简化了Spring应用的整个搭建和开发过程。另外SpringBoot通过集成大量的框架使得依赖包的版本冲突，以及引用的不稳定性等问题得到了很好的解决。

SpringBoot框架中还有两个非常重要的策略：开箱即用和约定优于配置。

SpringBoot应用系统开发模板的基本架构设计从前端到后台进行说明：前端常使用模板引擎，主要有FreeMarker和Thymeleaf，它们都是用Java语言编写的，渲染模板并输出相应文本，使得界面的设计与应用的逻辑分离，同时前端开发还会使用到Bootstrap、AngularJS、JQuery等；在浏览器的数据传输格式上采用Json，非xml，同时提供RESTfulAPI；SpringMVC框架用于数据到达服务器后处理请求；到数据访问层主要有Hibernate、MyBatis、JPA等持久层框架；数据库常用[MySQL](https://baike.baidu.com/item/MySQL/471251)；开发工具推荐IntelliJIDEA。

从最根本上来讲，Spring Boot就是一些库的集合，它能够被任意项目的构建系统所使用。

### 2）SpringBoot所具备的特征有：

（1）可以创建独立的[Spring](https://baike.baidu.com/item/Spring/85061)应用程序，并且基于其Maven或Gradle插件，

可以创建可执行的JARs和WARs；

（2）内嵌Tomcat或Jetty等Servlet容器；

（3）提供自动配置的“starter”项目对象模型（POMS）以简化[Maven](https://baike.baidu.com/item/Maven/6094909)配置；

（4）尽可能自动配置Spring容器；

（5）提供准备好的特性，如指标、健康检查和外部化配置；

（6）绝对没有代码生成，不需要XML配置。

### 3）Spring boot与Spring cloud 是什么关系？

（参考1.微服务的图）

Spring Boot是build anything。Anything包含很多，其中就包含右侧的Spring Cloud和再右侧的Spring Cloud Data Flow。

Spring Cloud是Coordinate Anything。下面写的：Built directly on Spring Boot's innovative approach to enterprise Java。

Spring boot 是 Spring 的一套快速配置脚手架，可以基于spring boot 快速开发单个微服务，Spring Cloud是一个基于Spring Boot实现的云应用开发工具；Spring boot专注于快速、方便集成的单个个体，Spring Cloud是关注全局的服务治理框架；spring boot使用了默认大于配置的理念，很多集成方案已经帮你选择好了，能不配置就不配置，Spring Cloud很大的一部分是基于Spring boot来实现。

Spring boot可以离开Spring Cloud独立使用开发项目，但是Spring Cloud离不开Spring boot，属于依赖的关系。

## 1.微服务

<https://martinfowler.com/microservices/> 微服务定义介绍

Microservices Guide

In short, the microservice architectural style is an approach to developing a single application as a **suite of small services, each running in its own process** and communicating with lightweight mechanisms, often an HTTP resource API. These services are **built around business capabilities** and **independently deployable** by fully automated deployment machinery. There is a **bare minimum of centralized management** of these services, which may be written in different programming languages and use different data storage technologies.

**-- James Lewis and Martin Fowler (2014)**

微服务：

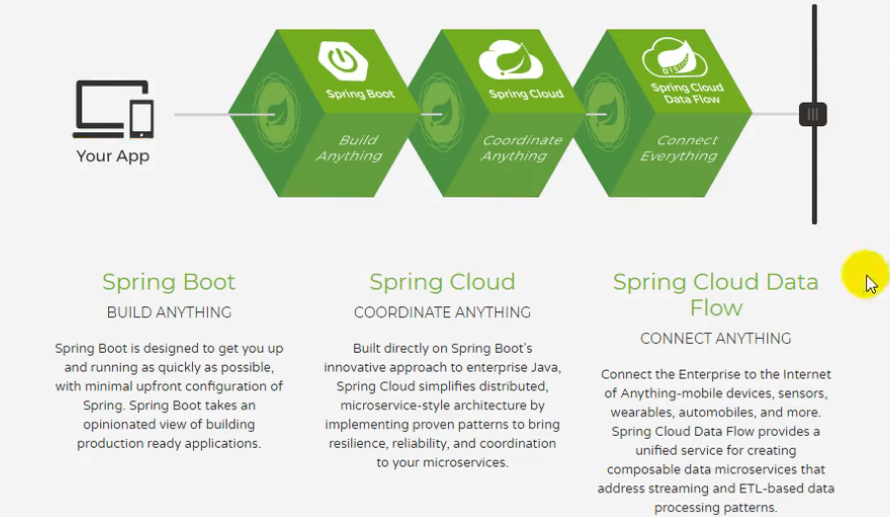
（单体服务：目前传统开发方式，all in one）

一个应用应该是一组小型服务，可以通过HTTP的方式进行互通。

每个功能元素最终都是一个可独立替换和独立升级的软件单元 。

（一个大型的分布式应用）

这么复杂的系统怎么构建? Spring的一系列框架都帮我们想好了



## 2.学习SpringBoot的前提

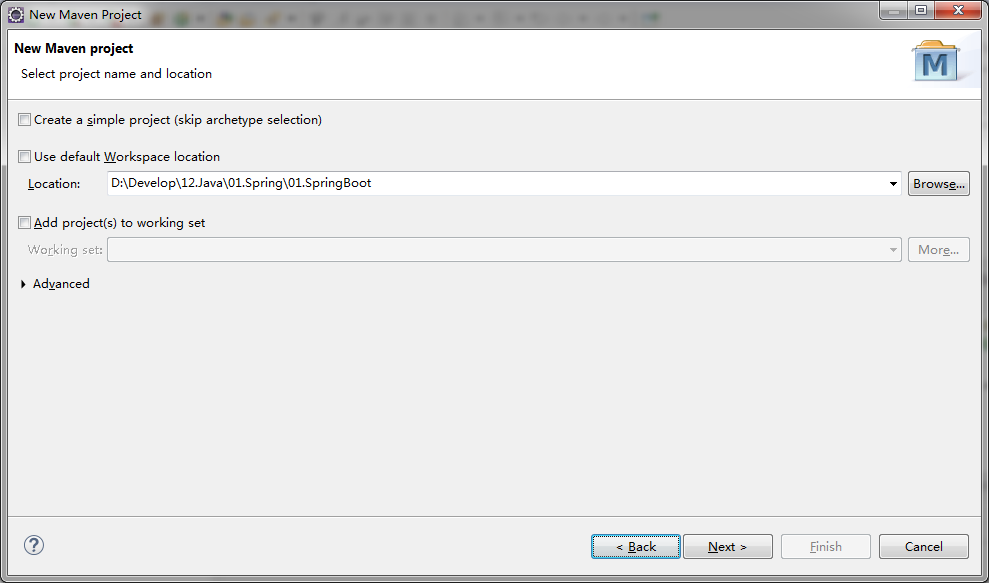
-有Spring框架的使用经验

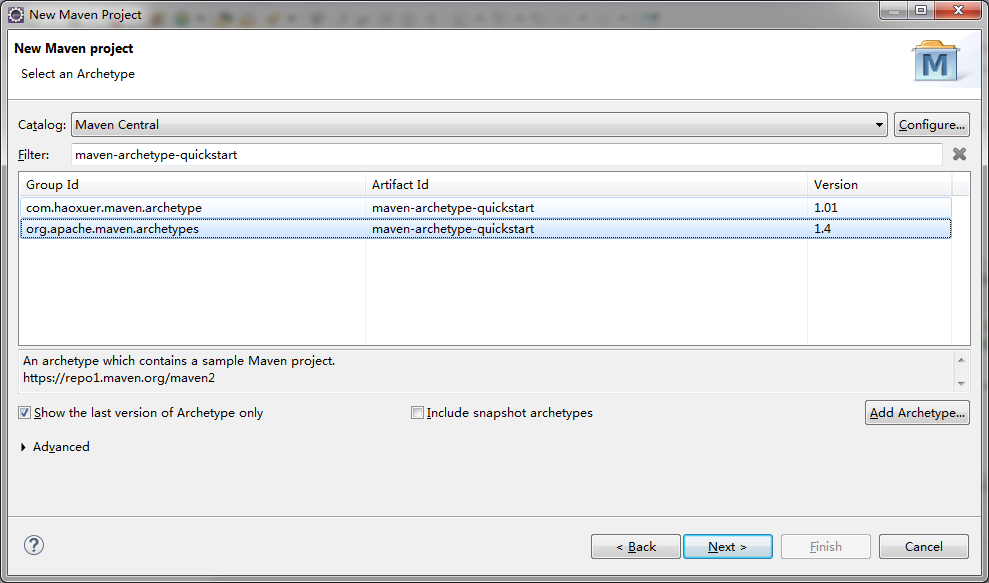
-熟练使用maven进行项目的构建和依赖管理

-熟练使用Eclipse或IDEA

## 3.Spring Boot hello world

### 1）创建maven工程



maven-archetype-quickstart 

**知识扩展：**

archetype的意思就是模板原型的意思，原型是一个Maven项目模板工具包。一个原型被定义为从其中相同类型的所有其它事情是由一个原始图案或模型。名称配合，因为我们正在努力提供一种系统，该系统提供了一种生成Maven项目的一致的手段。原型将帮助作者为用户创建Maven项目模板，并为用户提供了手段，产生的这些项目模板参数化的版本。

建立Maven项目时，网上建议的分别是：

1、cocoon-22-archetype-webapp

2、maven-archetype-quickstart

3、maven-archetype-webapp

### 2）导入springboot的相关依赖

POM.xml:

<parent>

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

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

<version>1.5.9.RELEASE</version>

<relativePath/>

</parent>

<dependencies>

<dependency>

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

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

</dependency>

</dependencies>

### 3）编写一个主程序：启动springboot应用

/\*\*

\***@SpringBootApplication** 标注一个springboot主程序

\*/

@SpringBootApplication

**public** **class** HelloWorldMainApplication {

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

// Spring 应用启动起来

SpringApplication.*run*(HelloWorldMainApplication.**class**, args);

}

}

### 4）编写controller，services

@Controller

**public** **class** HelloController {

@ResponseBody

@RequestMapping("/hello")

**public** String hello() {

**return** "Hello World!";

}

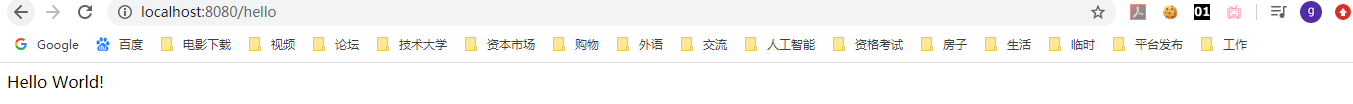
}

### 5）eclipse启动main方法 用浏览器访问

2020-09-09 23:54:51.255 INFO 13468 --- [ main] s.b.c.e.t.TomcatEmbeddedServletContainer : Tomcat started on port(s): 8080 (http)

2020-09-09 23:54:51.318 INFO 13468 --- [ main] c.d.s.HelloWorldMainApplication : Started HelloWorldMainApplication in 17.711 seconds (JVM running for 20.31)

浏览器访问



### 6）使用插件打包为可执行的jar包 cmd启动应用

插件加入pom.xml

<!-- 这个maven插件可以将应用打包成可执行jar -->

<build>

<plugins>

<plugin>

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

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

</plugin>

</plugins>

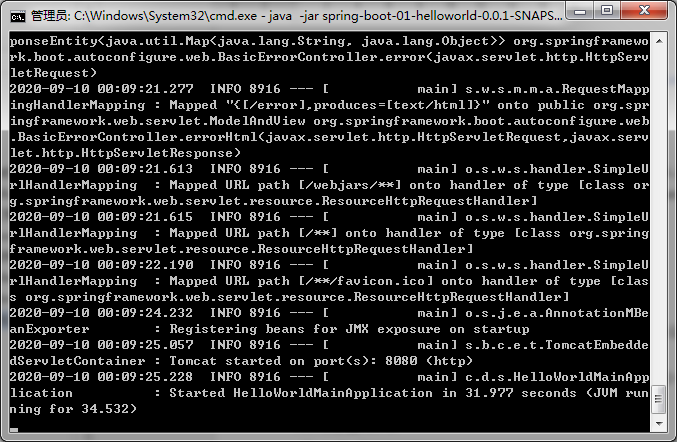
</build>

运行maven install

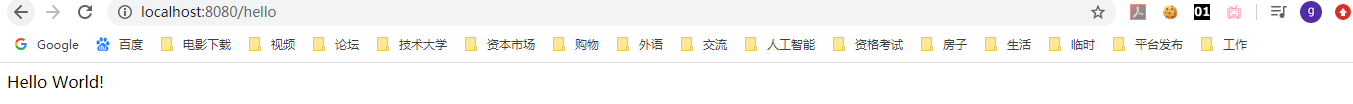
打包后在target目录下：spring-boot-01-helloworld-0.0.1-SNAPSHOT.jar

进入cmd

使用java –jar spring-boot-01-helloworld-0.0.1-SNAPSHOT.jar 也可以启动



用浏览器访问：



## 4.helloworld的研究

### 1）POM文件

#### ①父亲项目

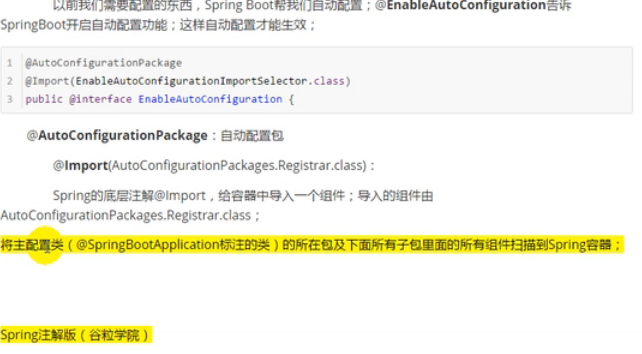


#### ②导入的依赖



### 2）主程序类：主入口类





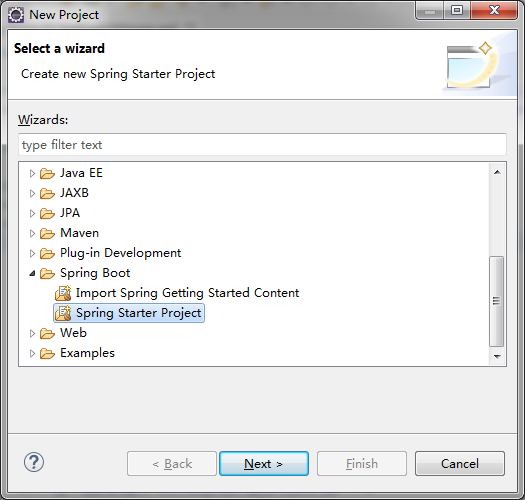


## 5.快速创建springboot项目

(springboot使用的事嵌入式的tomcat 默认是不支持jsp页面的)

使用IDEA的spring Intializer可以快速创建springboot项目

使用Eclipse的话要先安装springtool插件,然后使用spring starterproject也可以快速创建springboot项目,效果和spring initializer一样



## 6.springboot的配置文件

springboot使用一个全局配置文件,文件名是固定的,

/src/main/resources/application.properties

/src/main/resources/application.yaml (或者/src/main/resources/application.yml)

这个配置文件的作用是修改springboot的默认配置值,

(springboot在底层其实已经给我们配置好了)

配置文件的作用：修改SpringBoot自动配置的默认值；SpringBoot在底层都给我们自动配置好；

YAML（YAML Ain't Markup Language）

YAML A Markup Language：是一个标记语言

YAML isn't Markup Language：不是一个标记语言；

标记语言：

以前的配置文件；大多都使用的是 xxxx.xml文件；

YAML：以数据为中心，比json、xml等更适合做配置文件；

### YAML：配置例子

server:

port: 8081

XML：

<server>

<port>8081</port>

</server>

### 1、基本语法

k:(空格)v：表示一对键值对（空格必须有）；

以空格的缩进来控制层级关系；只要是左对齐的一列数据，都是同一个层级的

属性和值也是大小写敏感；

## 7.profile

## ※8.SpringBoot自动配置原理

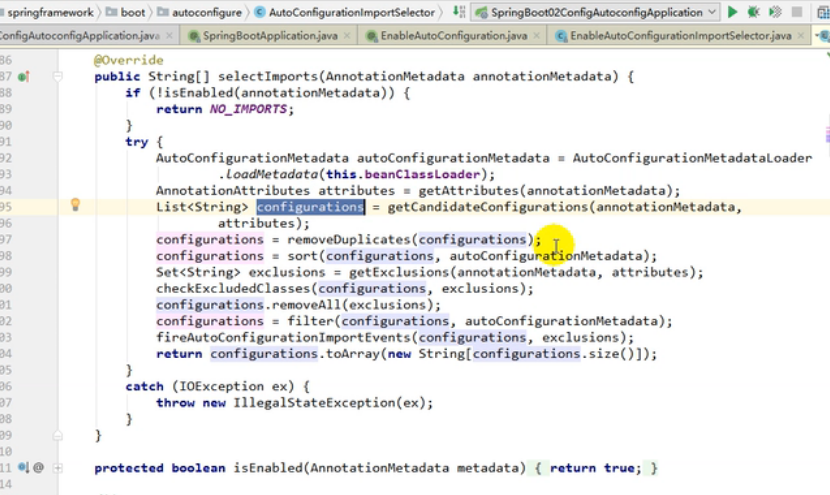
### 8-1.自动配置原理

1）、SpringBoot启动的时候加载主配置类，开启了自动配置功能@EnableAutoConfiguration

**2）、@EnableAutoConfiguration 作用：**

利用EnableAutoConfigurationImportSelector给容器中导入一些组件？

可以查看selectImports()方法的内容；



List configurations = getCandidateConfigurations(annotationMetadata, attributes);获取候选的配置

SpringFactoriesLoader.loadFactoryNames()

扫描所有jar包类路径下 META‐INF/spring.factories

把扫描到的这些文件的内容包装成properties对象

从properties中获取到EnableAutoConfiguration.class类（类名）对应的值，然后把他们添加在容器中

**将类路径下META-INF/spring.factories 里面配置的所有EnableAutoConfiguration的值加入到了容器中；**

# Auto Configure

org.springframework.boot.autoconfigure.EnableAutoConfiguration=\

org.springframework.boot.autoconfigure.admin.SpringApplicationAdminJmxAutoConfiguration,\

org.springframework.boot.autoconfigure.aop.AopAutoConfiguration,\

org.springframework.boot.autoconfigure.amqp.RabbitAutoConfiguration,\

org.springframework.boot.autoconfigure.batch.BatchAutoConfiguration,\

org.springframework.boot.autoconfigure.cache.CacheAutoConfiguration,\

org.springframework.boot.autoconfigure.cassandra.CassandraAutoConfiguration,\

org.springframework.boot.autoconfigure.cloud.CloudAutoConfiguration,\

org.springframework.boot.autoconfigure.context.ConfigurationPropertiesAutoConfiguration,\

org.springframework.boot.autoconfigure.context.MessageSourceAutoConfiguration,\

org.springframework.boot.autoconfigure.context.PropertyPlaceholderAutoConfiguration,\

org.springframework.boot.autoconfigure.couchbase.CouchbaseAutoConfiguration,\

org.springframework.boot.autoconfigure.dao.PersistenceExceptionTranslationAutoConfiguration

省略。。。

org.springframework.boot.autoconfigure.web.WebClientAutoConfiguration,\

org.springframework.boot.autoconfigure.web.WebMvcAutoConfiguration,\

org.springframework.boot.autoconfigure.websocket.WebSocketAutoConfiguration,\

org.springframework.boot.autoconfigure.websocket.WebSocketMessagingAutoConfiguration,\

org.springframework.boot.autoconfigure.webservices.WebServicesAutoConfiguration

每一个这样的 xxxAutoConfiguration类都是容器中的一个组件，都加入到容器中；用他们来做自动配置；

3）、每一个自动配置类进行自动配置功能；

4）、**以HttpEncodingAutoConfiguration（Http编码自动配置）为例解释自动配置原理；**

@Configuration //表示这是一个配置类，以前编写的配置文件一样，也可以给容器中添加组件

@EnableConfigurationProperties(HttpEncodingProperties.class) //启动指定类的

ConfigurationProperties功能；将配置文件中对应的值和HttpEncodingProperties绑定起来；并把

HttpEncodingProperties加入到ioc容器中

@ConditionalOnWebApplication //Spring底层@Conditional注解（Spring注解版），根据不同的条件，如果

满足指定的条件，整个配置类里面的配置就会生效； 判断当前应用是否是web应用，如果是，当前配置类生效

@ConditionalOnClass(CharacterEncodingFilter.class) //判断当前项目有没有这个类

CharacterEncodingFilter；SpringMVC中进行乱码解决的过滤器；

@ConditionalOnProperty(prefix = "spring.http.encoding", value = "enabled", matchIfMissing =

true) //判断配置文件中是否存在某个配置 spring.http.encoding.enabled；如果不存在，判断也是成立的

//即使我们配置文件中不配置pring.http.encoding.enabled=true，也是默认生效的；

public class HttpEncodingAutoConfiguration {

//他已经和SpringBoot的配置文件映射了

private final HttpEncodingProperties properties;

//只有一个有参构造器的情况下，参数的值就会从容器中拿

public HttpEncodingAutoConfiguration(HttpEncodingProperties properties) {

this.properties = properties;

}

@Bean //给容器中添加一个组件，这个组件的某些值需要从properties中获取

@ConditionalOnMissingBean(CharacterEncodingFilter.class) //判断容器没有这个组件？

public CharacterEncodingFilter characterEncodingFilter() {

CharacterEncodingFilter filter = new OrderedCharacterEncodingFilter();

filter.setEncoding(this.properties.getCharset().name());

filter.setForceRequestEncoding(this.properties.shouldForce(Type.REQUEST));

filter.setForceResponseEncoding(this.properties.shouldForce(Type.RESPONSE));

return filter;

}

}

根据当前不同的条件判断，决定这个配置类是否生效？

一但这个配置类生效；这个配置类就会给容器中添加各种组件；这些组件的属性是从对应的properties类中获取的，这些类里面的每一个属性又是和配置文件绑定的；

5）、所有在配置文件中能配置的属性都是在xxxxProperties类中封装者‘；配置文件能配置什么就可以参照某个功能对应的这个属性类

@ConfigurationProperties(prefix = "spring.http.encoding") //从配置文件中获取指定的值和bean的属性进行绑定

public class HttpEncodingProperties {

public static final Charset DEFAULT\_CHARSET = Charset.forName("UTF‐8");

#### 精髓：

**1）、SpringBoot启动会加载大量的自动配置类**

**2）、我们看我们需要的功能有没有SpringBoot默认写好的自动配置类；**

**3）、我们再来看这个自动配置类中到底配置了哪些组件；（只要我们要用的组件有，我们就不需要再来配置了）**

**4）、给容器中自动配置类添加组件的时候，会从properties类中获取某些属性。我们就可以在配置文件中指定这些属性的值；**

xxxxAutoConfigurartion：自动配置类；

给容器中添加组件

xxxxProperties:封装配置文件中相关属性；

### 8-2、细节

1、@Conditional派生注解（Spring注解版原生的@Conditional作用）

作用：必须是@Conditional指定的条件成立，才给容器中添加组件，配置配里面的所有内容才生效；

@Conditional扩展注解 作用（判断是否满足当前指定条件）

@ConditionalOnJava 系统的java版本是否符合要求

@ConditionalOnBean 容器中存在指定Bean；

@ConditionalOnMissingBean 容器中不存在指定Bean；

@ConditionalOnExpression 满足SpEL表达式指定

@ConditionalOnClass 系统中有指定的类

@ConditionalOnMissingClass 系统中没有指定的类

@ConditionalOnSingleCandidate 容器中只有一个指定的Bean，或者这个Bean是首选Bean

@ConditionalOnProperty 系统中指定的属性是否有指定的值

@ConditionalOnResource 类路径下是否存在指定资源文件

@ConditionalOnWebApplication 当前是web环境

@ConditionalOnNotWebApplication 当前不是web环境

@ConditionalOnJndi JNDI 存在指定项自动配置类必须在一定的条件下才能生效；

我们怎么知道哪些自动配置类生效；

**我们可以通过启用 debug=true属性；来让控制台打印自动配置报告**，这样我们就可以很方便的知道哪些自动配置

类生效；

## 三、日志

### 1、日志框架

市面上的日志框架；

JUL、JCL、Jboss-logging、logback、log4j、log4j2、slf4j....

SpringBoot：底层是Spring框架，Spring框架默认是用JCL；‘

SpringBoot选用 SLF4j和logback；

### 2、SLF4j使用

1、如何在系统中使用SLF4j https://www.slf4j.org

以后开发的时候，日志记录方法的调用，不应该来直接调用日志的实现类，而是调用日志抽象层里面的方法；

给系统里面导入slf4j的jar和 logback的实现jar

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class HelloWorld {

public static void main(String[] args) {

Logger logger = LoggerFactory.getLogger(HelloWorld.class);

logger.info("Hello World");

}

}

图示；

## 四、Web开发

### 1、简介

使用SpringBoot；

1）、创建SpringBoot应用，选中我们需要的模块；

2）、SpringBoot已经默认将这些场景配置好了，只需要在配置文件中指定少量配置就可以运行起来

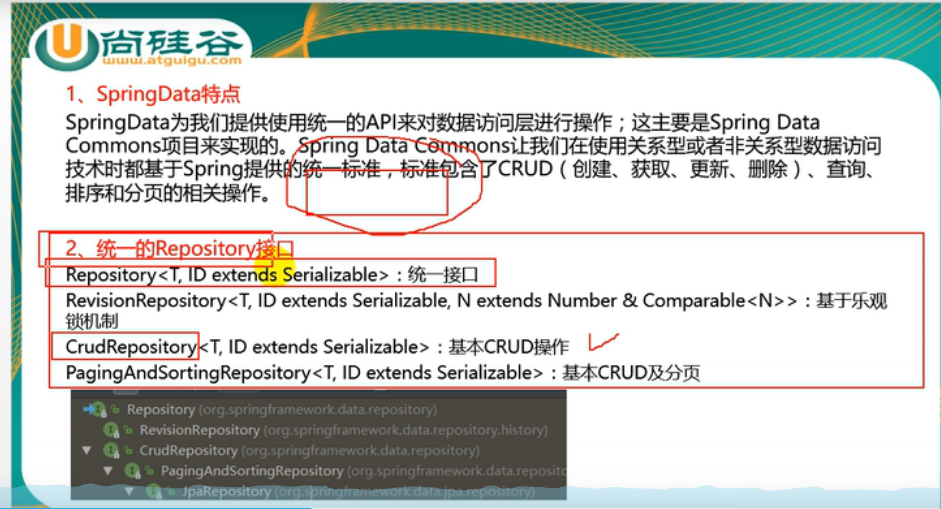
3）、自己编写业务代码；

#### 自动配置原理？

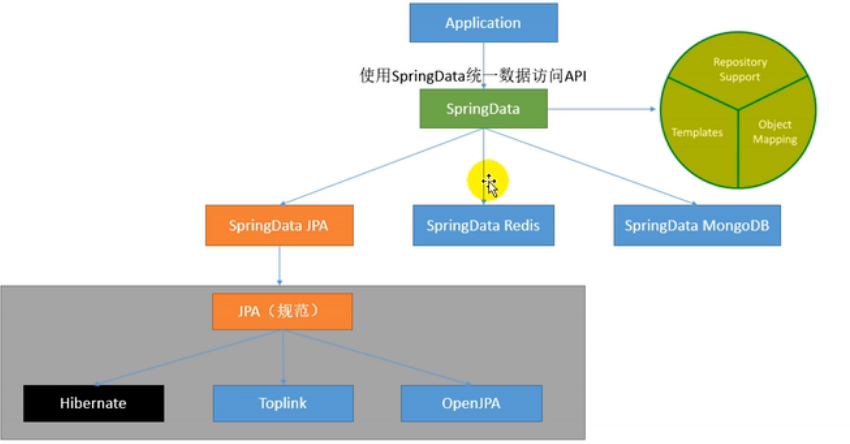
这个场景SpringBoot帮我们配置了什么？能不能修改？能修改哪些配置？能不能扩展？xxx

## 六、SpringBoot与数据访问

### Spring data



### 整合SpringData JPA



## 七、启动配置原理

几个重要的事件回调机制

配置在META-INF/spring.factories

ApplicationContextInitializer

SpringApplicationRunListener

只需要放在ioc容器中

ApplicationRunner

CommandLineRunner

启动流程：

1、创建SpringApplication对象

2、运行run方法

3、事件监听机制

## 八、自定义starter

starter：

1、这个场景需要使用到的依赖是什么？

2、如何编写自动配置

# SpringBoot高级应用