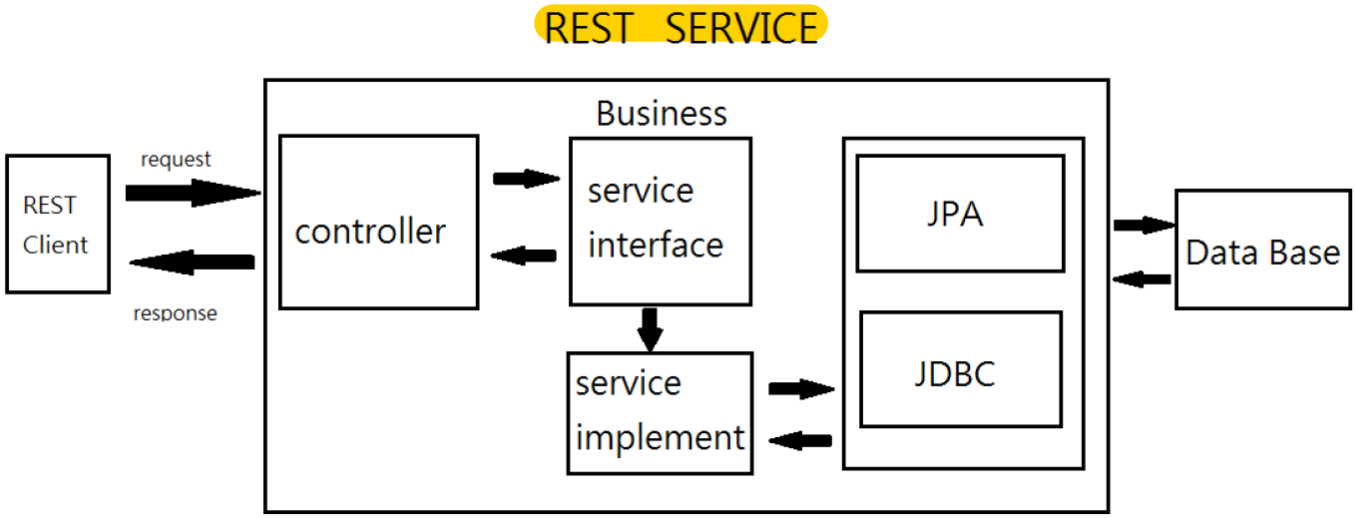
2. Spring Boot架構



整個 Spring Boot 架構基本上會像這張圖，左邊有個 REST Client，它發送請求到整個 Rest Service，負責處理接收 request 跟回送 response 會是第一層 Controller class。但整個業務邏輯的部分，並不會寫在這邊，**Controller 只負責處理請求跟回應而已**。

業務邏輯的部分則會切到下一層，也就是 Service 層，特別的地方是並不會單純的只建一個 Service class，而是會有一個 Service Interface 和 Service Implement class，原因在於物件導向 **SOLID** 原則的第五個 **Dependency Inversion Principle(DIP) 依賴反轉**，在寫物件導向程式語言時，會希望如果 class 間有互相依賴，那應該依賴在抽象介面上，而不是依賴於實體類別，好處在於抽換時會比較方便。

而 Service 層因為牽涉到業務邏輯，所以可能會用到 DB 內的資料，DB 資料的處理會用 JPA 的規範來進行處理，有個 class 來專門處理這件事情。

之後會細講每個 class 需要完成的事情，以及應該怎麼完成，這邊會大略帶過 Spring Boot 的架構，再來看幾個常見的問題，第一個是 Spring Boot 的專案架構，一定要建得像這樣嗎？

```
|--com.example.demospringboot
   |--DemospringbootApplication.java
|--com.example.demospringboot.controller
   |--TestController.java
```

前面介紹 Spring Boot initializr 時，有建一個 controller package，有人可能會把 `com.example.demospringboot.controller` 直接替換成 `controller` ，這麼做的話 Spring Boot 會出錯，

原因在於一開始 `main` 方法上面有個 `@SpringBootApplication` ，按 Ctrl 點進去看會下面的程式碼：

```
@Target({java.lang.annotation.ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Inherited
@SpringBootConfiguration
@EnableAutoConfiguration
@ComponentScan(excludeFilters={@org.springframework.context.annotation.ComponentScan.Filter(type
@ConfigurationPropertiesScan
public @interface SpringBootApplication
{
  @AliasFor(annotation=EnableAutoConfiguration.class)
  Class<?>[] exclude() default {};

  @AliasFor(annotation=EnableAutoConfiguration.class)
  String[] excludeName() default {};

  //以下省略
}
```

必須這麼做的原因是 `@ComponentScan` ，它會把當前這支 class 底下資料夾的所有 `@Component` 跟 `@Bean` 都掃起來，放到 Spring Boot 的 Bean pool 裡面，等要用時再使用 `@Autowired` 把物件拿出來。那什麼是 `@Component` 、 `@Bean` ？基本上 Spring Boot 多數 class 都可被視為 Bean 跟 Component，像前面用到的 `@Controller` ，點進去會發現有個 `@Component` ：

```
@Target({java.lang.annotation.ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Component
public @interface Controller
{
  @AliasFor(annotation=Component.class)
  String value() default "";
}
```

那 `@Bean` 跟 `@Component` 有不一樣嗎？現階段先當成一樣的東西就好，他們最終都會被放到 Spring Boot Bean pool 裡，只是兩者在用法上不太一樣。再來則是 `@Autowired` 的功用，放到 Spring Boot Bean pool 裡的東西，總有要拿來使用的時候，使用時就是使用 `@Autowired` ，它會幫忙把 Bean pool 裡的東西給抓出來。所以回過頭來看這個問題，如果專案架構不這樣建， `@ComponentScan` 就無法掃到 Bean 跟 Component，想要用 `@Autowired` 抓取 Bean pool 裡的東西就會出錯。

第二個問題是，把每個 class 功能切這麼細的原因是什麼，原因在於職責分工，個人認為有點像關注點分離，或物件導向 **SOLID** 原則的第一個 **Single responsibility principle(SRP) 單一職責原則**，每支 class 都應該只專職於一件事情，這在閱讀程式碼或找尋 bug 時有很大的幫助，我們當然可以嘗試把所

有的東西都寫在一個 class 裡面，但對於之後閱讀或維護程式碼來講，這樣的寫法其實不太友善，想像一下當錯誤出現在一支 class 中，而這支 class 因為所有的邏輯都寫在裡頭所以有上千行，多數人應該都不怎麼喜歡閱讀這樣的程式碼吧。

> 那 SOLID 其他幾個原則分別是什麼？

參考

https://skyyen999.gitbooks.io/-study-design-pattern-in-java/content/oodPrinciple.html
https://www.tutorialspoint.com/spring_boot/spring_boot_introduction.htm