Spring Framework

Introduction

The Spring Framework provides a comprehensive programming and configuration model for modern Java-based enterprise applications - on any kind of deployment platform. A key element of Spring is infrastructural support at the application level: Spring focuses on the "plumbing" of enterprise applications so that teams can focus on application-level business logic, without unnecessary ties to specific deployment environments.

Features

- Dependency Injection
- Aspect-Oriented Programming including Spring's declarative transaction management
- Spring MVC web application and RESTful web service framework
- Foundational support for JDBC, JPA, JMS
- Much more…

Minimum requirements

- JDK 6+ for Spring Framework 4.x
- JDK 5+ for Spring Framework 3.x

4.3.0
Maven
Gradle
The recommended way to get started using `spring-framework` in your project is with a dependency management system – the snippet below can be copied and pasted into your build. Need help? See our getting started guides on building with Maven and Gradle.

```
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>4.3.0.RELEASE</version>
    </dependency>
</dependencies>
```

Spring Framework includes a number of different modules. Here we are showing `spring-context` which provides core functionality. Refer to the getting started guides on the right for other options.

Once you've set up your build with the `spring-context` dependency, you'll be able to do the following:

hello/MessageService.java

```
package hello;

public interface MessageService {
    String getMessage();
}
```

hello/MessagePrinter.java

```
package hello;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component
public class MessagePrinter {

    final private MessageService service;

    @Autowired
    public MessagePrinter(MessageService service) {
        this.service = service;
    }

    public void printMessage() {
        System.out.println(this.service.getMessage());
    }
}
```

hello/Application.java

```
package hello;

import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.*;

@Configuration
@ComponentScan
public class Application {

    @Bean
```

```
    MessageService mockMessageService() {
        return new MessageService() {
            public String getMessage() {
              return "Hello World!";
            }
        };
    }

  public static void main(String[] args) {
      ApplicationContext context =
          new AnnotationConfigApplicationContext(Application.class);
      MessagePrinter printer = context.getBean(MessagePrinter.class);
      printer.printMessage();
  }
}
```

The example above shows the basic concept of dependency injection, the `MessagePrinter` is decoupled from the `MessageService` implementation, with Spring Framework wiring everything together.