依赖倒置原则
+
控制反转+依赖注入

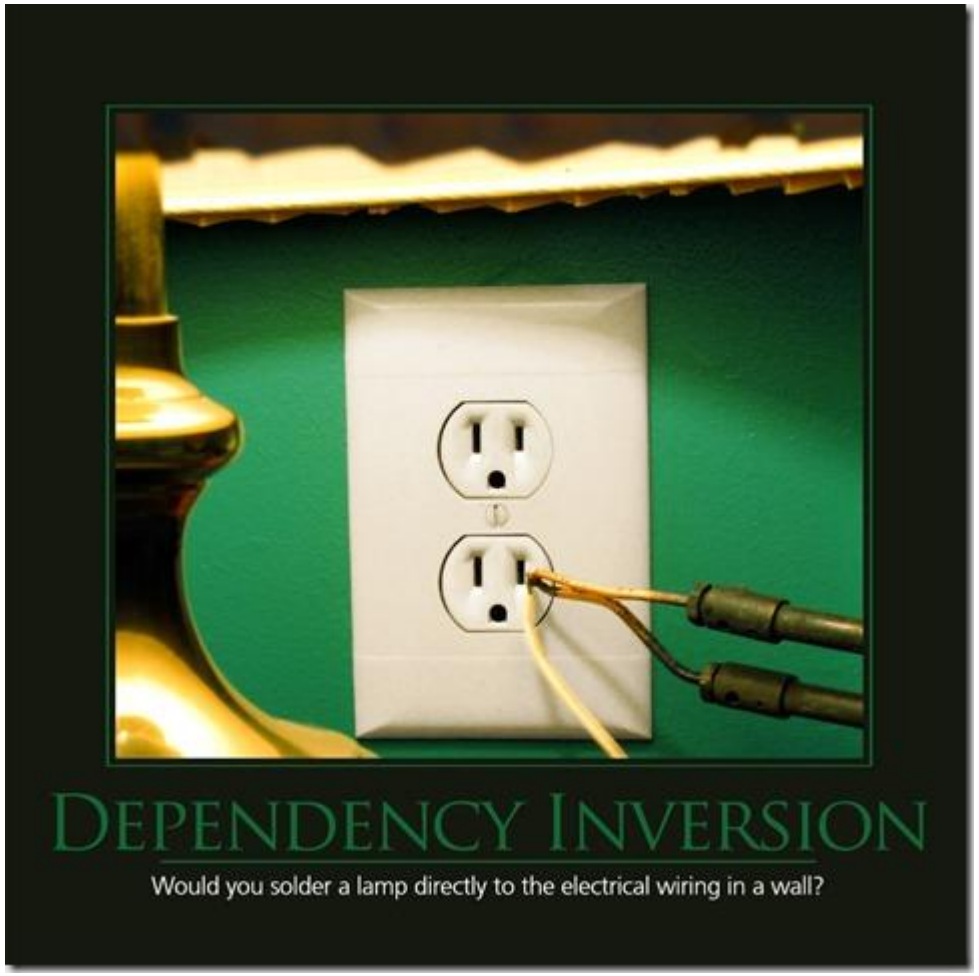
波波老师~研发总监/资深架构师

波波微课
spring2go.com

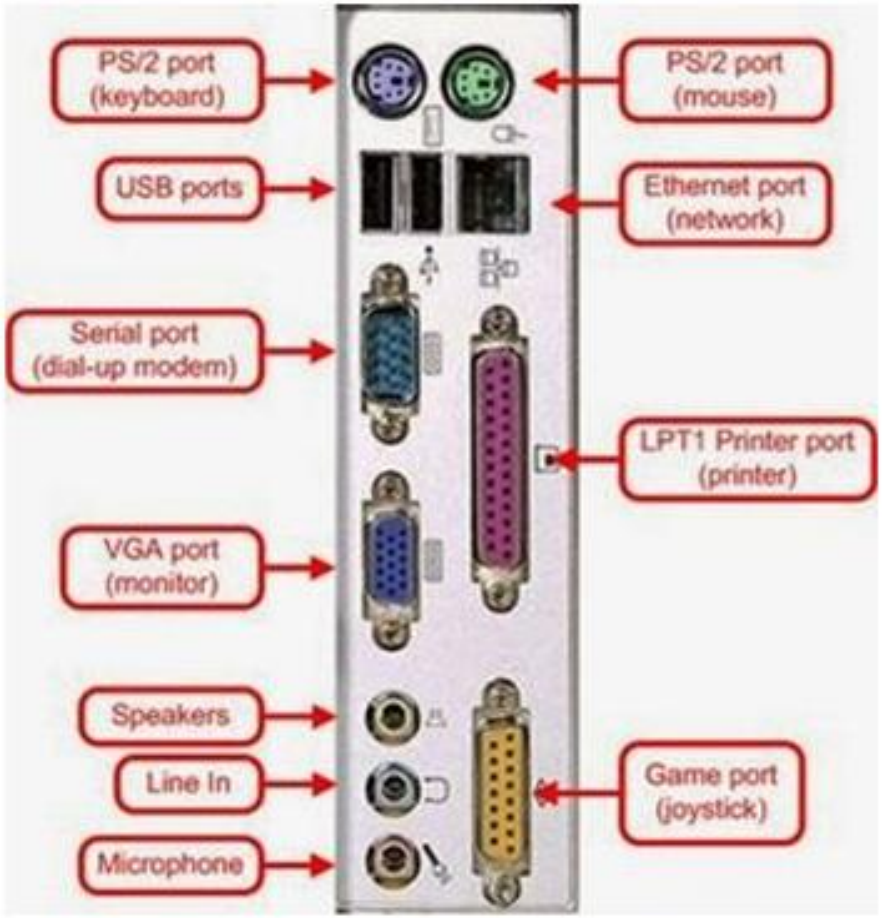
依赖倒置原则 (Dependency Inversion Principle)

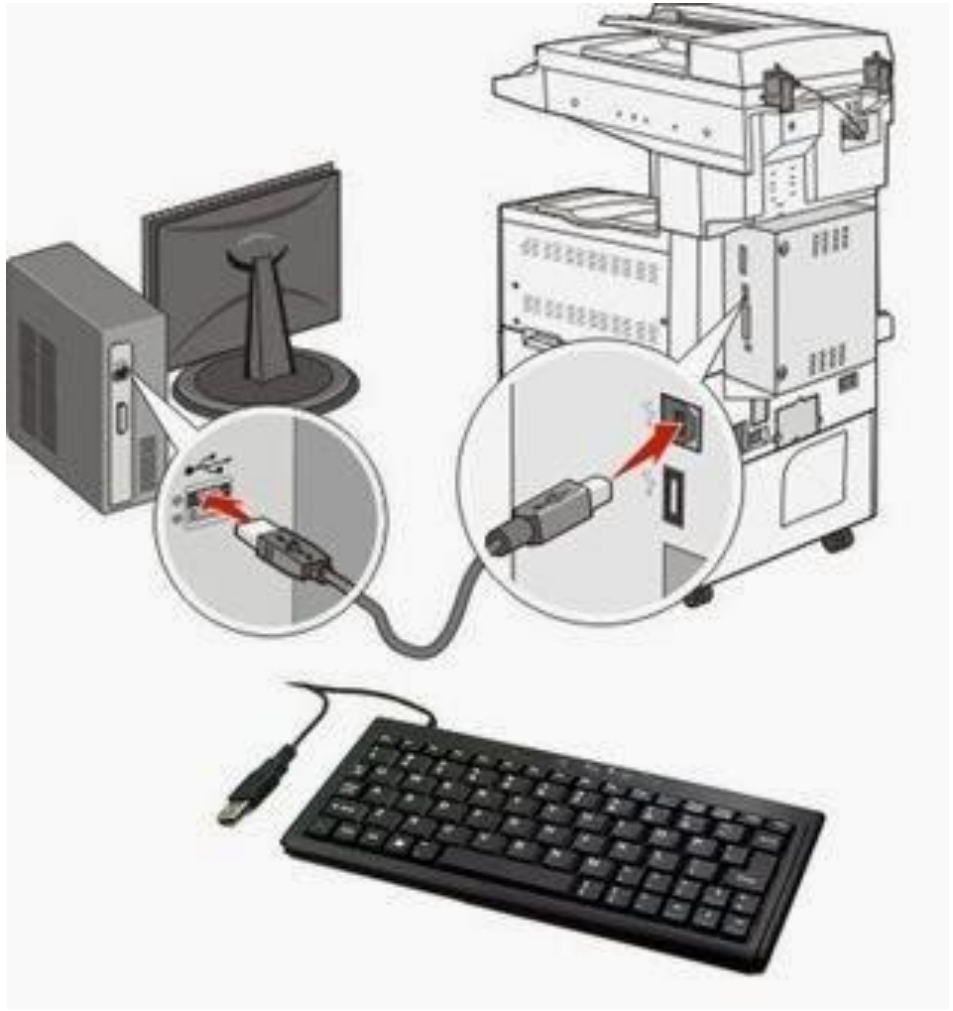
- SOLID面向对象原理之一
  - 高层模块不应该依赖底层模块。两者都应该依赖于抽象。
  - 抽象不应该依赖于细节。细节应该依赖于抽象。
- 面向接口编程

问题


DEPENDENCY INVERSION
Would you solder a lamp directly to the electrical wiring in a wall?

现实案例

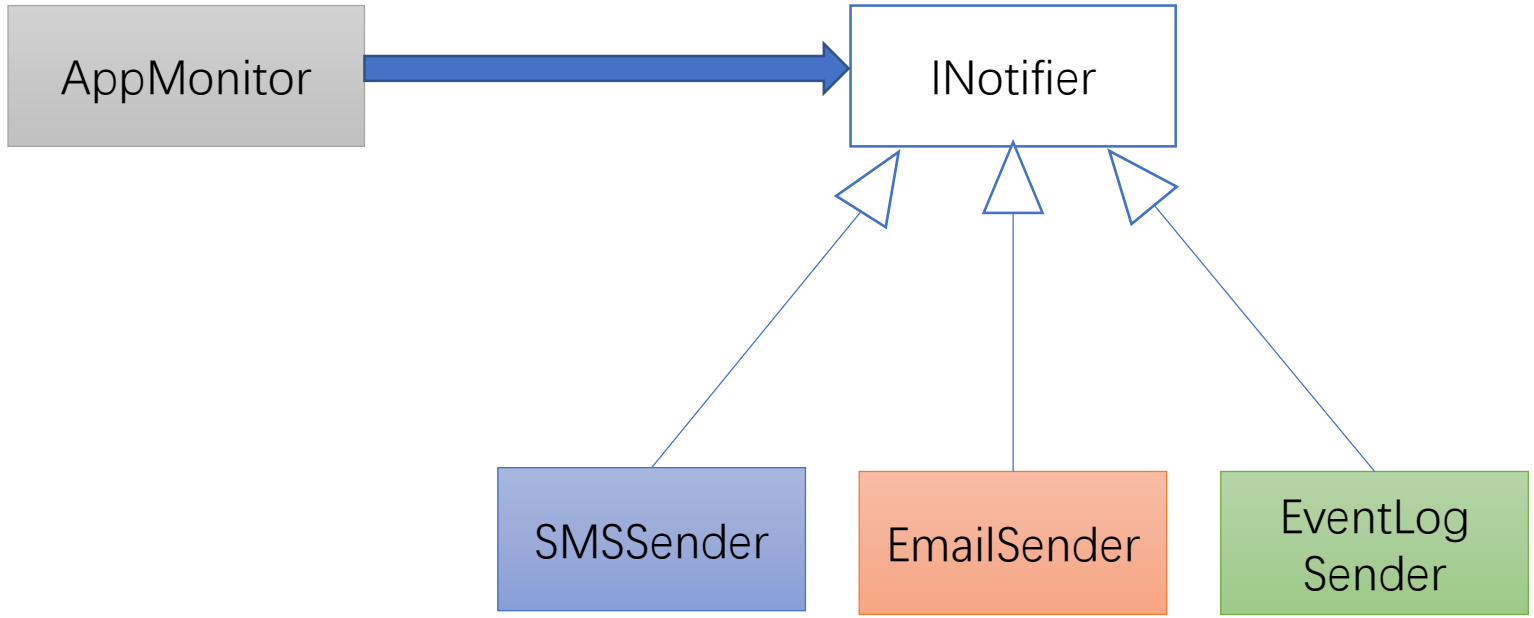

PS/2 port (keyboard)
PS/2 port (mouse)
USB ports
Ethernet port (network)
Serial port (dial-up modem)
LPT1 Printer port (printer)
VGA port (monitor)
Speakers
Line In
Game port (joystick)
Microphone



问题代码

```java
package io.spring2go.corespring.nodip;

public class AppMonitorNoDIP {

    // 负责将事件日志写到日志系统
    private EventLogWriter writer = null;

    // 应用有问题时该方法将被调用
    public void notify(String message) {
        if (writer == null) {
            writer = new EventLogWriter();
        }
        writer.write(message);
    }

    public static void main(String[] args) {
        AppMonitorNoDIP appMonitor = new AppMonitorNoDIP();
        appMonitor.notify("App has a problem ...");
    }
}

class EventLogWriter {

    public void write(String message) {
        // 写到事件日志
        System.out.println("Write to event log, message : " + message);
    }
}
```

关系图



- 高层模块不应该依赖底层模块。两者都应该依赖于抽象。
- 抽象不应该依赖于细节。细节应该依赖于抽象。

DIP实现~接口和实现类

```java
package io.spring2go.corespring.dip;

// 事件通知器接口
public interface INotifier {
    public void notify(String message);
}
```

```java
package io.spring2go.corespring.dip;

public class EmailSender implements INotifier {

    public void notify(String message) {
        // 发送Email
        System.out.println("Send email, message : " + message);
    }

}
```

```java
package io.spring2go.corespring.dip;

public class EventLogWriter implements INotifier {

    public void notify(String message) {
        // 写事件日志
        System.out.println("Write to event log, message : " + message);
    }

}
```

```java
package io.spring2go.corespring.dip;

public class SMSSender implements INotifier {

    public void notify(String message) {
        // 发送短消息
        System.out.println("Send SMS, message : " + message);
    }
}
```

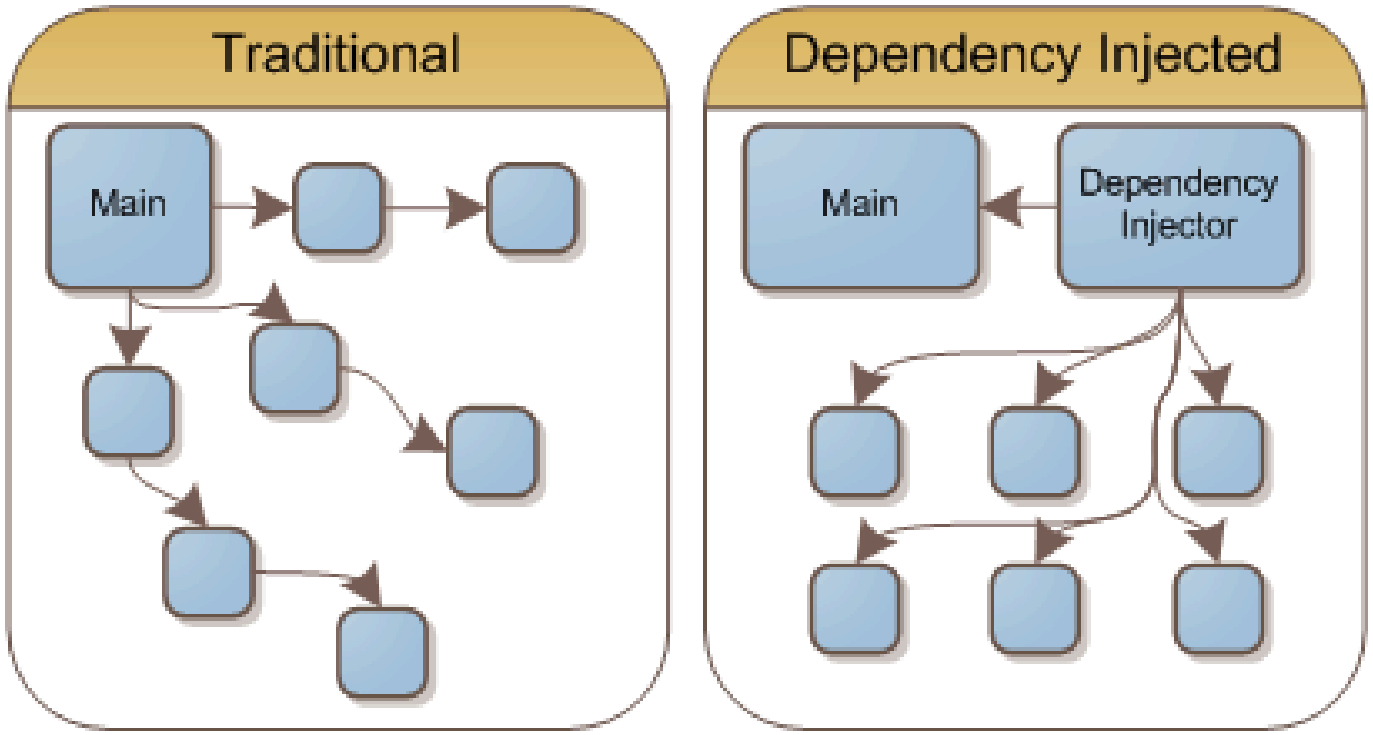
DIP实现~客户类

```java
package io.spring2go.corespring.dip;

public class AppMonitorDIP {
    // 事件通知器
    private INotifier notifier = null;

    // 应用有问题时调用该方法
    public void notify(String message) {
        if (notifier == null) {
            // 将抽象接口映射为具体类
            notifier = new EventLogWriter();
        }
        notifier.notify(message);
    }

    public static void main(String[] args) {
        AppMonitorDIP appMonitor = new AppMonitorDIP();
        appMonitor.notify("App has a problem ...");
    }
}
```

还有问题？

```java
package io.spring2go.corespring.dip;

public class AppMonitorDIP {
    // 事件通知器
    private INotifier notifier = null;

    // 应用有问题时调用该方法
    public void notify(String message) {
        if (notifier == null) {
            // 将抽象接口映射为具体类
            notifier = new EventLogWriter();
        }
        notifier.notify(message);
    }

    public static void main(String[] args) {
        AppMonitorDIP appMonitor = new AppMonitorDIP();
        appMonitor.notify("App has a problem ...");
    }
}
```

控制反转（Inversion of Control）



依赖注入（Dependency Injection）

- 构造函数注入
- Setter注入
- 接口注入

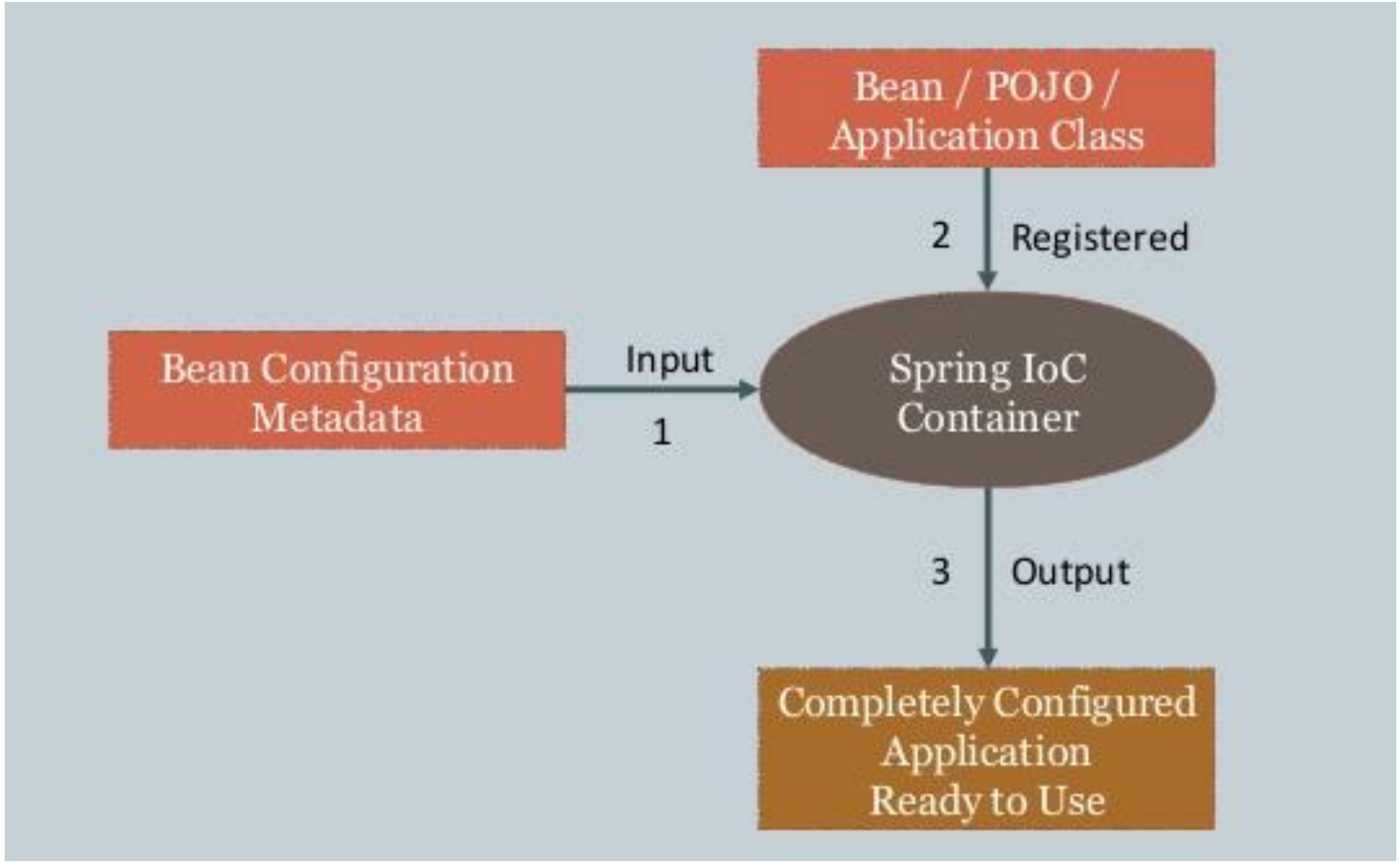
构造函数注入

```java
package io.spring2go.corespring.injection;

import io.spring2go.corespring.ioc.EventLogWriter;
import io.spring2go.corespring.ioc.INotifier;

public class AppMonitorConstructorInjection {
    // 事件通知器
    private INotifier notifier = null;

    public AppMonitorConstructorInjection(INotifier notifier) {
        this.notifier = notifier;
    }

    // 应用有问题时该方法被调用
    public void notify(String message) {
        notifier.notify(message);
    }

    public static void main(String[] args) {
        EventLogWriter writer = new EventLogWriter();
        AppMonitorConstructorInjection monitor =
                new AppMonitorConstructorInjection(writer);
        monitor.notify("App has a problem ...");
    }
```

Setter注入

```java
package io.spring2go.corespring.injection;

import io.spring2go.corespring.ioc.EventLogWriter;
import io.spring2go.corespring.ioc.INotifier;

public class AppMonitorSetterInjection {
    // 事件通知器
    private INotifier notifier = null;

    public void SetNotifier(INotifier notifier) {
        this.notifier = notifier;
    }

    // 应用有问题时该方法被调用
    public void notify(String message) {
        notifier.notify(message);
    }

    public static void main(String[] args) {
        EventLogWriter writer = new EventLogWriter();
        AppMonitorSetterInjection monitor =
                new AppMonitorSetterInjection();
        // 可以在其它类中设置
        monitor.SetNotifier(writer);
        // 可以在其它类中调用
        monitor.notify("App has a problem ...");
    }
}
```

Spring IoC

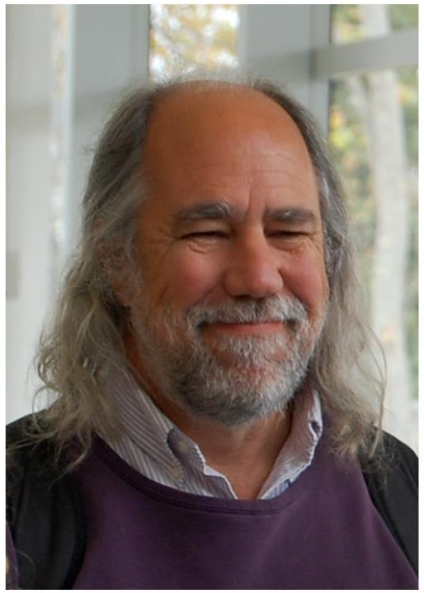

Bean / POJO / Application Class
2
Registered
Bean Configuration Metadata
Input
1
Spring IoC Container
3
Output
Completely Configured Application Ready to Use

好处

- 依赖解耦
- 模块化
- 易于测试
- 易于变化和扩展

什么是架构

- Architecture represents the significant design decisions that shape a system, where significant is measured by cost of change.
- 架构表示对一个系统的成型起关键作用的设计决策，架构定系统基本就成型了，这里的关键性可以由**变化的成本**来决定
- +**质量反馈的速度**



Grady Booch，UML创始人

课后学习

- Spring IoC和依赖注入

参考

- An Absolute Beginner’ s tutorial on Dependency Inversion Principle, Inversion of Control and Dependency Injection（by Rahul Rajat Singh）
  - https://www.codeproject.com/Articles/615139/An-Absolute-Beginners-Tutorial-on-Dependency-Inver

代码

- https://github.com/spring2go/core-spring-patterns

波波微课

- 关于波波微课
  - 十多年研发经验老司机波波老师主导
  - 致力于使用新媒体技术提升学习成效
  - 主题面向Java，Spring，面向对象开发和微服务等
  - 关注工程师的成长
- 理念
  - 交互式的课程体验
  - 贴近一线企业实践
- 方法
  - 短视频，平均10分钟，最长不超过15分钟
  - 一个视频专注讲清楚一个主题
  - 50%原理+50%案例代码
  - 所有代码和ppt在github上可免费获得