函数式编程实现DDD

管理复杂度

聚焦领域

管理复杂度

管理复杂度: 分块

管理复杂度: 分段

管理复杂度: 分层

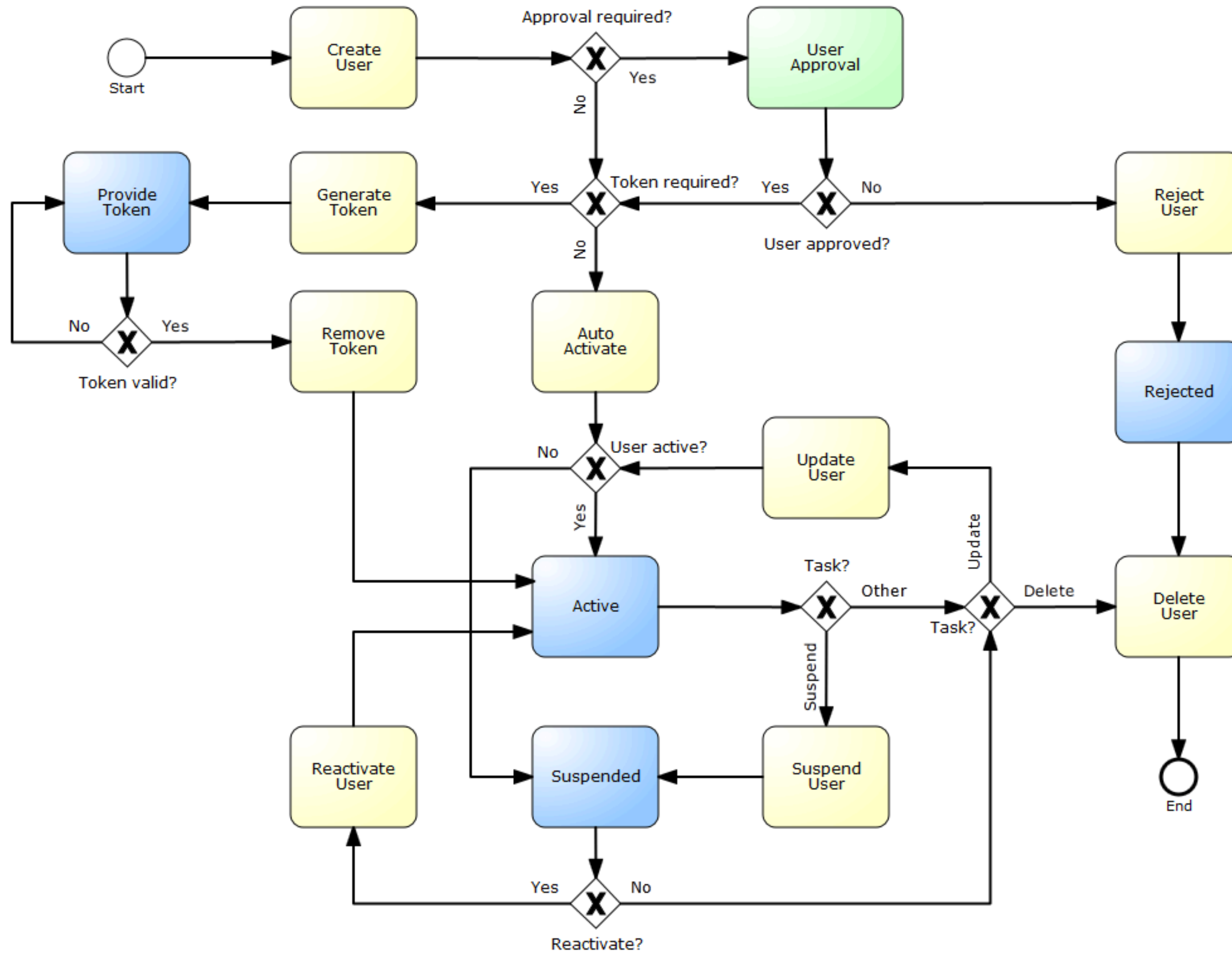
举个例子

注册用户

业务流程

提炼

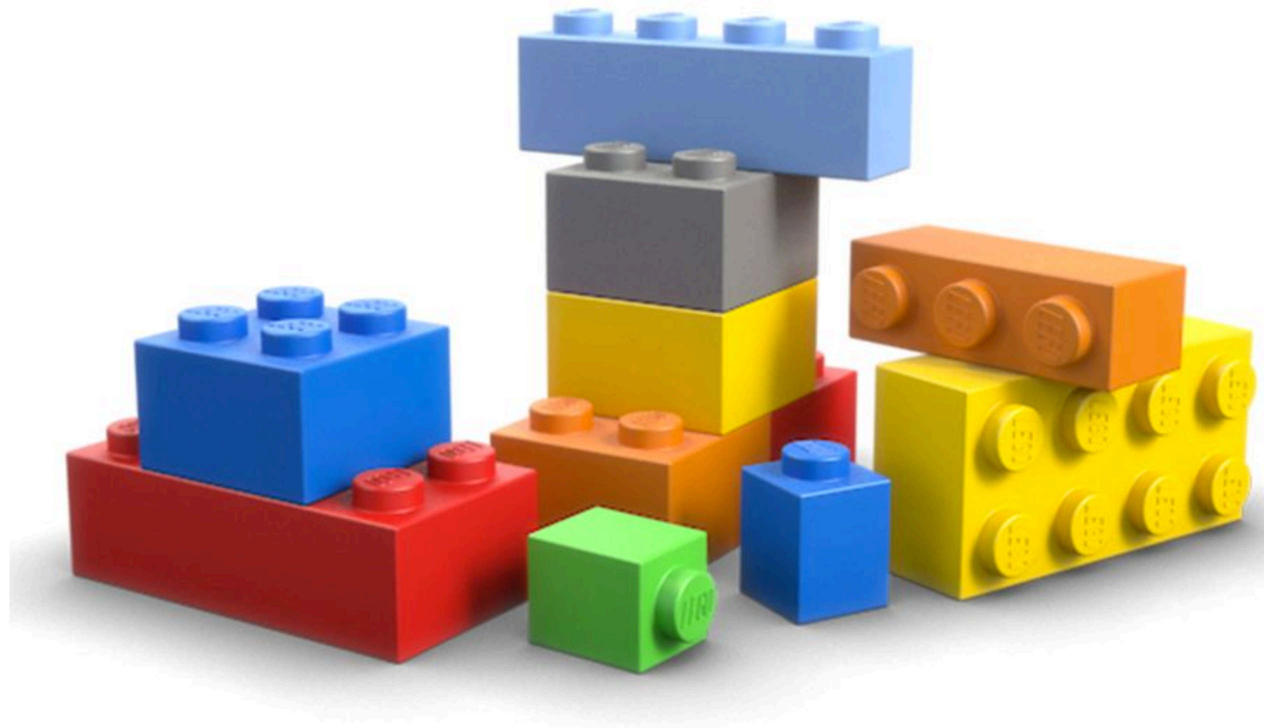
业务术语/对象



User

建模

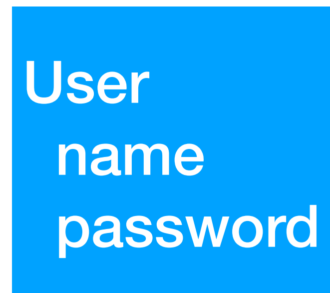
管理复杂度: 分块



用类型建模

领域建模

业务:



实现:

```
public class User {
    private String name;
    private String password;
}
```

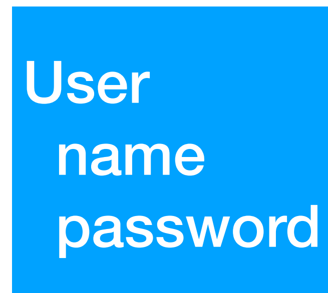
面向对象: 类

```
CREATE TABLE User (
    id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(30) NOT NULL,
    pwd VARCHAR(30) NOT NULL
    reg_date TIMESTAMP
)
```

数据库表

领域建模

业务:



实现:

```
public class User {
    private String name;
    private String password;
}
```

面向对象: 类

```
CREATE TABLE User (
    id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(30) NOT NULL
    pwd VARCHAR(30) NOT NULL
    reg_date TIMESTAMP
)
```

数据库表

掺入了技术实现细节，并且模糊了业务含义

领域建模

业务:

User
  name
  password

实现:

```
case class User(name: UserName, pwd: Password)
```

保留业务

```
type UserName = String
type Password = String
```

剥离技术实现

当业务变得复杂

用户名种类多
注册途径多

手机注册

邮箱注册

自定义用户名注册

邀请码注册

密码方式多
登录途径多

账号登录

第三方登录

生物特性登录

一键快捷登录

游客登录

领域建模

业务:

User
  name
  password

实现:

```
case class User(name: UserName, pwd: Password)
```

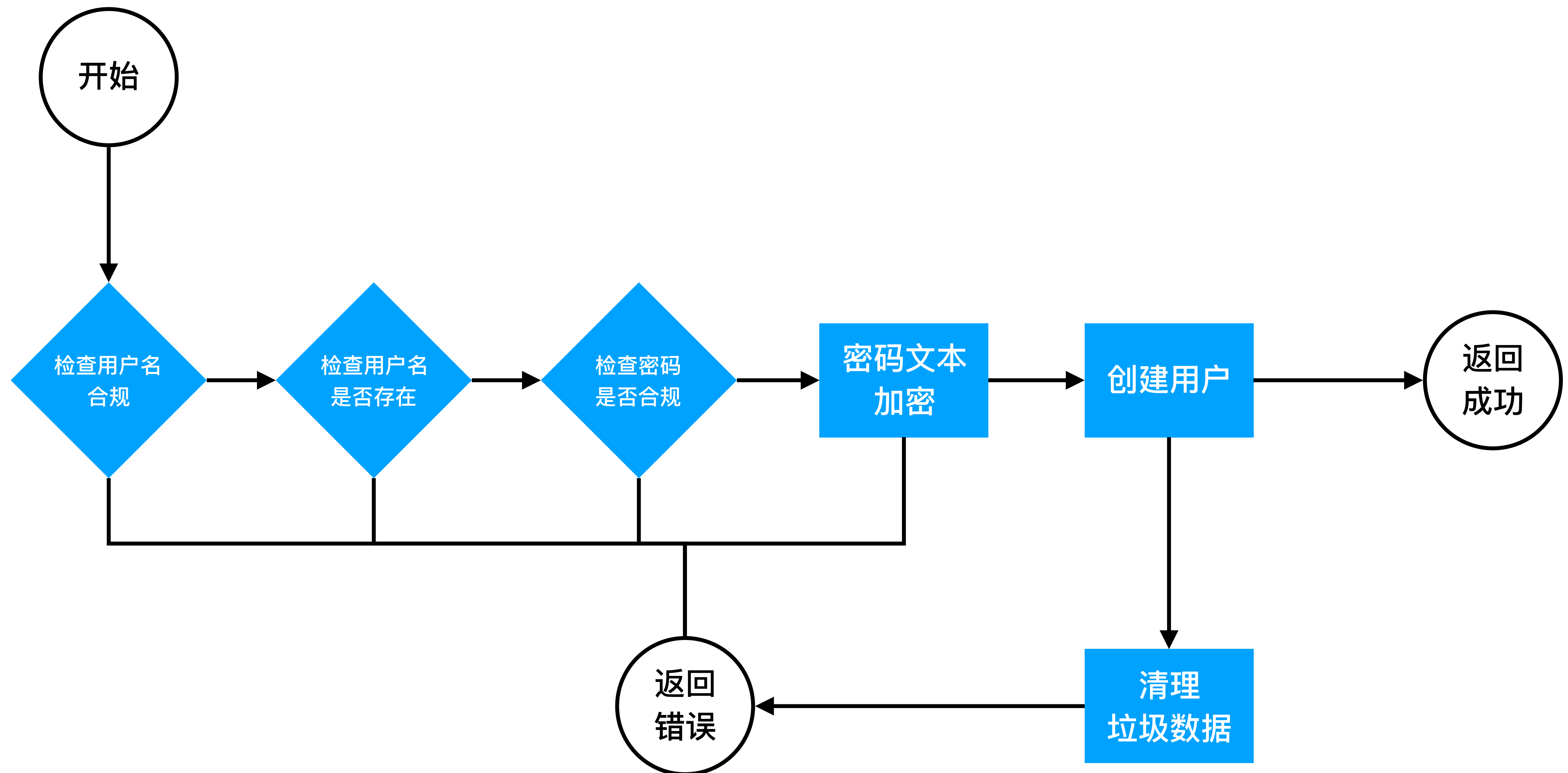
保持业务代码稳定

```
type UserName = String
type Password = String
```

修改实现，支持多种类用户名，验证方式

```
case class UserName(uid: Identification, typ: IdentificationType)

case class Password(auth: AuthenticationWord,authType: AuthenticationType)
```

注册用户



注册用户

非业务术语

```
public User register(String name, String password) throws Exception {
    if(!isUserNameValidate(name)) {
        throw new RegisterException("User name is invalidate"); // 缺乏具体的错误信息
    }
    if(!isUserNameExist(name)) {
        throw new RegisterException("User name is exist");
    }
    if(!isPasswordValidate(password)) {
        throw new RegisterException("Password is invalidate"); // 缺乏具体的错误信息
}
    String encodedPwd = encodePwd(password);
    User newUser = createUser(name, password);
    return newUser;
}
```

注册用户

改进

```
public User register(String name, String password) throws Exception {
    checkUserNameValidate(name)
    checkUserNameExist(name))
    checkPasswordValidate(password)
    String encodedPwd = encodePwd(password);
    User newUser = createUser(name, password);
    return newUser;
}
```

```
private int USERNAME_MIN_LENGTH = 3
private void checkUserNameValidate(String name) throws UserNameValidationException {
    if(StingUtils.isEmpty(name)) {
        throw new UserNameValidationException("User name can not be empty");
    }
    if(name.length < USERNAME_MIN_LENGTH) {
        throw new UserNameValidationException("length of User name must more then " + USERNAME_MIN_LENGTH);
    }
    if(isContainInvalidateCharacter(name)) {
        throw new UserNameValidationException("User name contains invalidate character");
    }
    ......
}
private void checkUserNameExist(String name) ......
```

业务流程 管道建模

管理复杂度: 分段

开始

检查用户名
合规

检查用户名
是否存在

检查密码
是否合规

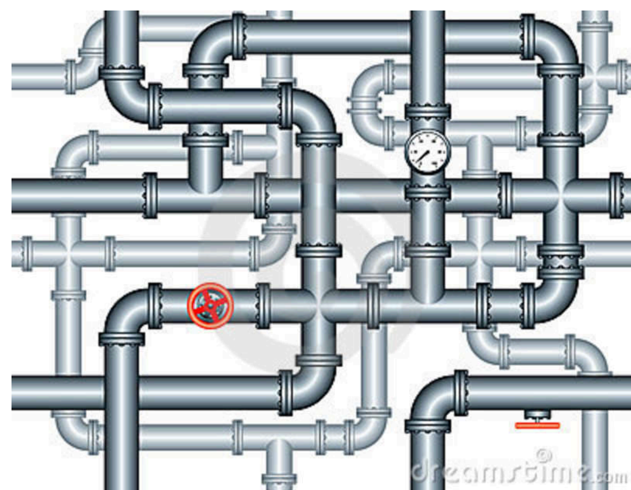
密码文本
加密

创建用户

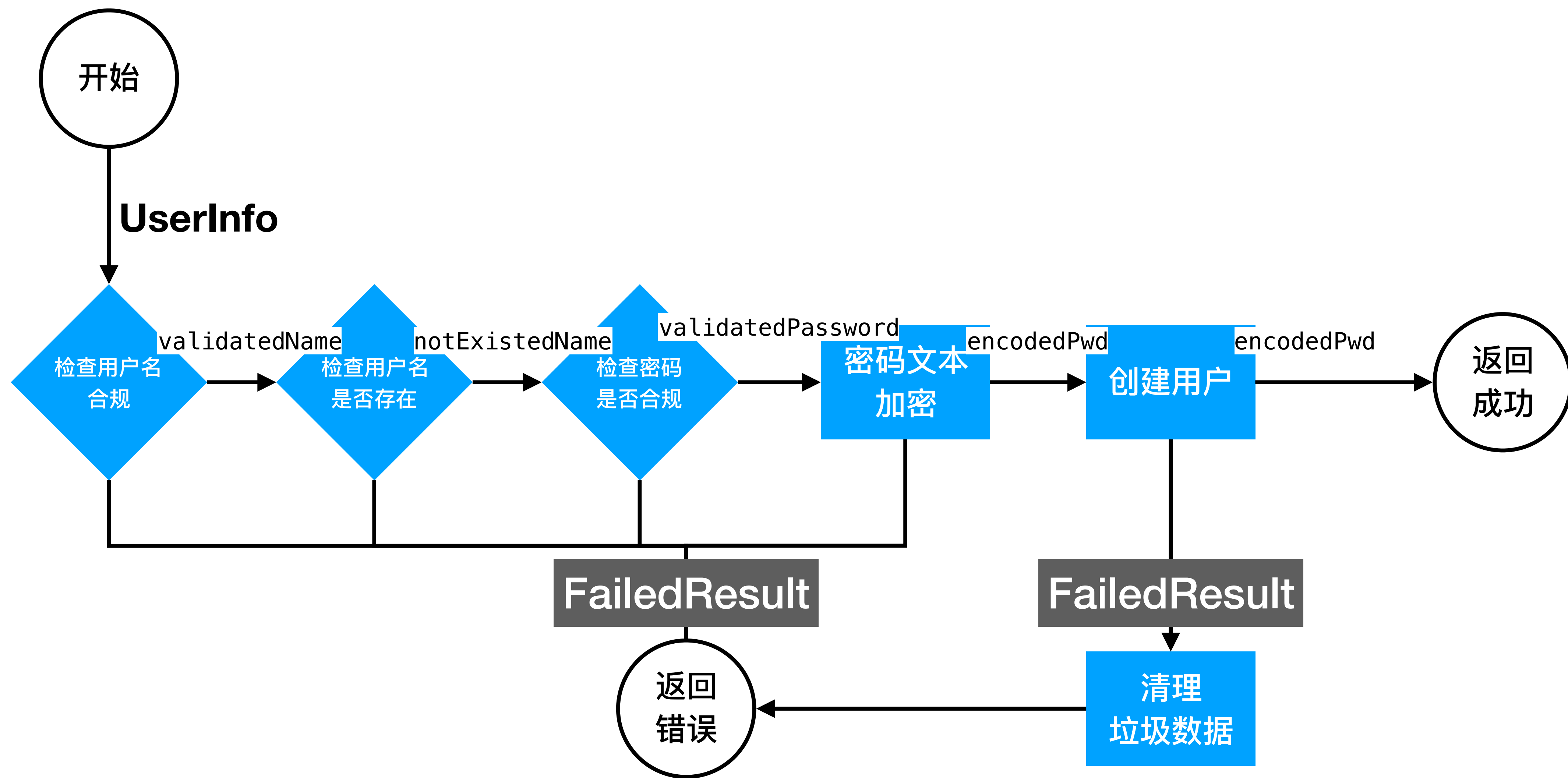
返回
成功

清理
垃圾数据

返回
错误



业务流程: 管道



管道

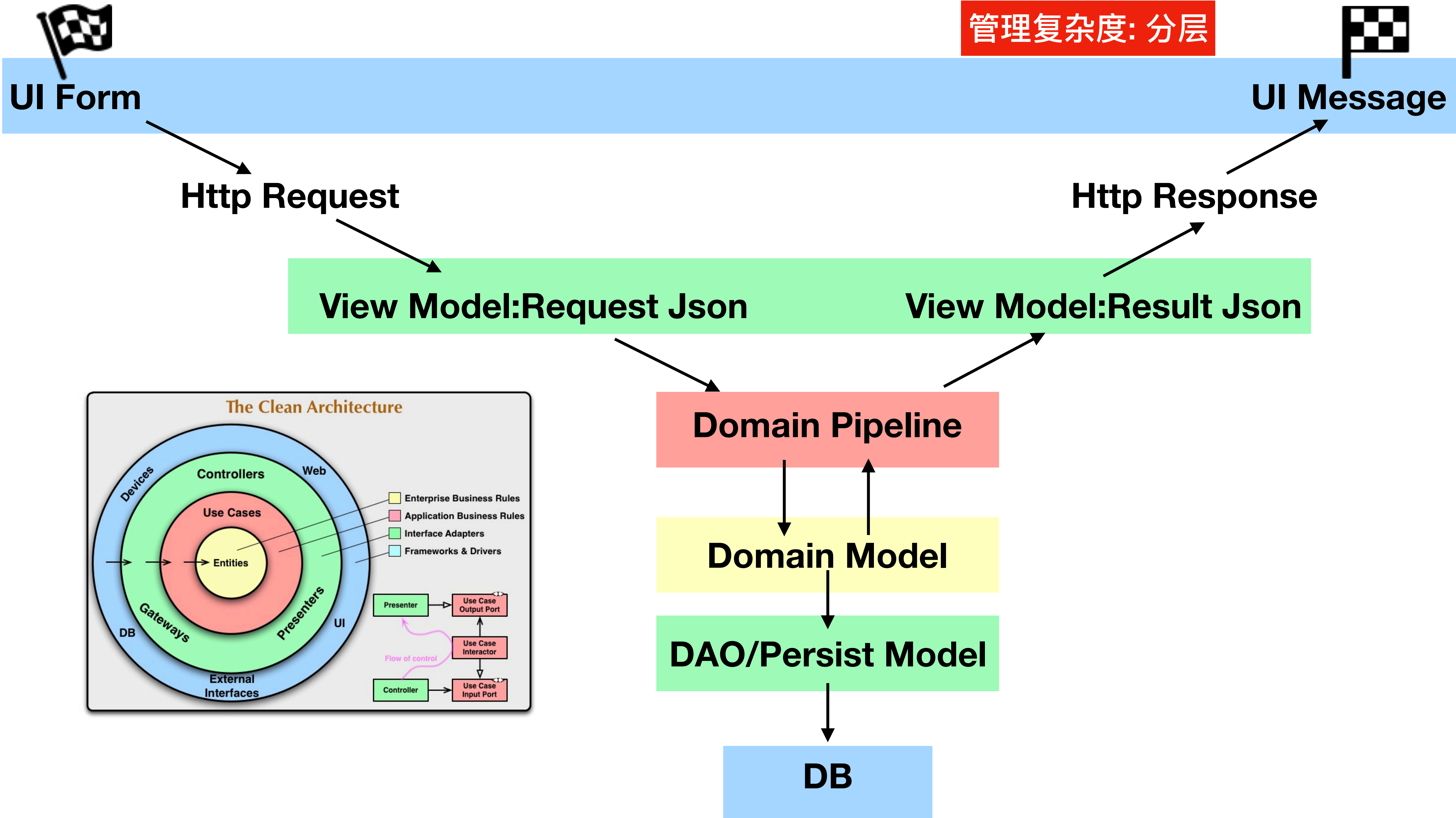
```
def registerUser(userInfo: UserInfo): BusinessResult[User] =
    for {
      validatedName <- checkUserNameValidate(userInfo.name)
      notExistedName <- checkUserNameExist(validatedName)
      validatedPassword <- checkPasswordValidate(userInfo.pwd)
      encodedPwd <- encodePwd(validatedPassword)
      newUser <- createUser(notExistedName, encodedPwd)
    } yield newUser
```

管道

实现: 非业务

```scala
sealed trait BusinessResult[+A] {

  def map[B](f: A => B): BusinessResult[B]

  def flatMap[B](f: A => BusinessResult[B]): BusinessResult[B]
}

case class OKResult[+A](result: A) extends BusinessResult[A] {
  def map[B](f: A => B): BusinessResult[B] = OKResult(f(result))

  def flatMap[B](f: A => BusinessResult[B]): BusinessResult[B] = f(result)
}

case class FailedResult(errorCode: ErrorCode, messages: String*) extends BusinessResult[Nothing] {

  def map[B](f: Nothing => B): BusinessResult[B] = this

  def flatMap[B](f: Nothing => BusinessResult[B]): BusinessResult[B] = this
}
```

业务流程 管道建模



DDD&FP
根本指导原则

关注点分离

DDD

构建业务和技术都懂的通用语言

去除业务中的行话
去除技术中的行话

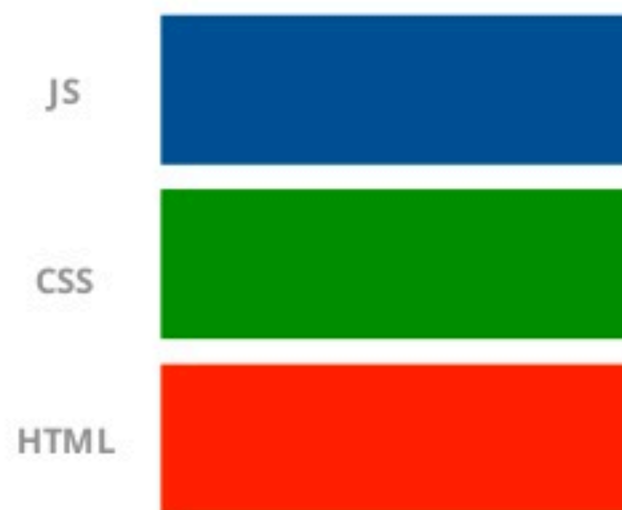
FP

一切都是不可变,无副作用

一个函数只关注输入和输出
函数和函数可以组合

管理复杂度

Separation of Concerns



Separation of Concerns

(only, from a different point of view)