

## 14 | 实战二（下）：如何利用面向对象设计和编程开发接口鉴权功能？

王争 · 设计模式之美



在上一节课中，针对接口鉴权功能的开发，我们讲了如何进行面向对象分析（OOA），也就是需求分析。实际上，需求定义清楚之后，这个问题就已经解决了一大半，这也是为什么我花了那么多篇幅来讲解需求分析。今天，我们再来看一下，针对面向对象分析产出的需求，如何进行面向对象设计（OOD）和面向对象编程（OOP）。

### 如何进行面向对象设计？

我们知道，面向对象分析的产出是详细的需求描述，那面向对象设计的产出就是类。在面向对象设计环节，我们将需求描述转化为具体的类的设计。我们把这一设计环节拆解细化一下，主要包含以下几个部分：

- 划分职责进而识别出有哪些类；

- 定义类及其属性和方法；

- 定义类与类之间的交互关系；

将类组装起来并提供执行入口。

实话讲，不管是面向对象分析还是面向对象设计，理论的东西都不多，所以我们还是结合鉴权这个例子，在实战中体会如何做面向对象设计。

## 1. 划分职责进而识别出有哪些类

在面向对象有关书籍中经常讲到，类是现实世界中事物的一个建模。但是，并不是每个需求都能映射到现实世界，也并不是每个类都与现实世界中的事物一一对应。对于一些抽象的概念，我们是无法通过映射现实世界中的事物的方式来定义类的。

所以，大多数讲面向对象的书籍中，还会讲到另外一种识别类的方法，那就是把需求描述中的名词罗列出来，作为可能的候选类，然后再进行筛选。对于没有经验的初学者来说，这个方法比较简单、明确，可以直接照着做。

不过，我个人更喜欢另外一种方法，那就是根据需求描述，把其中涉及的功能点，一个一个罗列出来，然后再去看哪些功能点职责相近，操作同样的属性，是否应该归为同一个类。我们来看一下，针对鉴权这个例子，具体该如何来做。

在上一节课中，我们已经给出了详细的需求描述，为了方便你查看，我把它重新贴在了下面。

调用方进行接口请求的时候，将 URL、AppID、密码、时间戳拼接在一起，通过加密算法生成 token，并且将 token、AppID、时间戳拼接在 URL 中，一并发送到微服务端。

微服务端在接收到调用方的接口请求之后，从请求中拆解出 token、AppID、时间戳。

微服务端首先检查传递过来的时间戳跟当前时间，是否在 token 失效时间窗口内。如果已经超过失效时间，那就算接口调用鉴权失败，拒绝接口调用请求。

如果 token 验证没有过期失效，微服务端再从自己的存储中，取出 AppID 对应的密码，通过同样的 token 生成算法，生成另外一个 token，与调用方传递过来的 token 进行匹配。如果一致，则鉴权成功，允许接口调用；否则就拒绝接口调用。

首先，我们要做的是逐句阅读上面的需求描述，拆解成小的功能点，一条一条罗列下来。注意，拆解出来的每个功能点要尽可能的小。每个功能点只负责做一件很小的事情（专业叫法

是“单一职责”，后面章节中我们会讲到）。下面是我逐句拆解上述需求描述之后，得到的功能点列表：

1. 把 URL、AppID、密码、时间戳拼接为一个字符串；
2. 对字符串通过加密算法加密生成 token；
3. 将 token、AppID、时间戳拼接到 URL 中，形成新的 URL；
4. 解析 URL，得到 token、AppID、时间戳等信息；
5. 从存储中取出 AppID 和对应的密码；
6. 根据时间戳判断 token 是否过期失效；
7. 验证两个 token 是否匹配；

从上面的功能列表中，我们发现，1、2、6、7 都是跟 token 有关，负责 token 的生成、验证；3、4 都是在处理 URL，负责 URL 的拼接、解析；5 是操作 AppID 和密码，负责从存储中读取 AppID 和密码。所以，我们可以粗略地得到三个核心的类：AuthToken、Url、CredentialStorage。AuthToken 负责实现 1、2、6、7 这四个操作；Url 负责 3、4 两个操作；CredentialStorage 负责 5 这个操作。

当然，这是一个初步的类的划分，其他一些不重要的、边边角角的类，我们可能暂时没法一下子想全，但这也没关系，面向对象分析、设计、编程本来就是一个循环迭代、不断优化的过程。根据需求，我们先给出一个粗糙版本的设计方案，然后基于这样一个基础，再去迭代优化，会更加容易一些，思路也会更加清晰一些。

不过，我还要再强调一点，接口调用鉴权这个开发需求比较简单，所以，需求对应的面向对象设计并不复杂，识别出来的类也并不多。但如果我们面对的是更加大型的软件开发、更加复杂的需求开发，涉及的功能点可能会很多，对应的类也会比较多，像刚刚那样根据需求逐句罗列功能点的方法，最后会得到一个长长的列表，就会有点凌乱、没有规律。针对这种复杂的需求开发，我们首先要做的是进行模块划分，将需求先简单划分成几个小的、独立的功能模块，然后再在模块内部，应用我们刚刚讲的方法，进行面向对象设计。而模块的划分和识别，跟类的划分和识别，是类似的套路。

## 2. 定义类及其属性和方法

刚刚我们通过分析需求描述，识别出了三个核心的类，它们分别是 AuthToken、Url 和 CredentialStorage。现在我们来看下，每个类都有哪些属性和方法。我们还是从功能点列表中挖掘。

**AuthToken 类相关的功能点有四个：**

把 URL、AppID、密码、时间戳拼接为一个字符串；

对字符串通过加密算法加密生成 token；

根据时间戳判断 token 是否过期失效；

验证两个 token 是否匹配。

对于方法的识别，很多面向对象相关的书籍，一般都是这么讲的，识别出需求描述中的动词，作为候选的方法，再进一步过滤筛选。类比一下方法的识别，我们可以把功能点中涉及的名词，作为候选属性，然后同样进行过滤筛选。

我们可以借用这个思路，根据功能点描述，识别出来 AuthToken 类的属性和方法，如下所示：

| AuthToken类 |  |
|------------|--|
| 属性         | <pre>private static final long DEFAULT_EXPIRED_TIME_INTERVAL = 1*60*1000; private String token; private long createTime; private long expiredTimeInterval = DEFAULT_EXPIRED_TIME_INTERVAL;</pre>               |
| 构造函数       | <pre>public AuthToken(String token, long createTime); public AuthToken(String token, long createTime, long expiredTimeInterval);</pre>   |
| 函数         | <pre>public static AuthToken create(String baseUrl, long createTime, Map&lt;String, String&gt; params); public String getToken(); public boolean isExpired(); public boolean match(AuthToken authToken);</pre> |

从上面的类图中，我们可以发现这样三个小细节。

第一个细节：并不是所有出现的名词都被定义为类的属性，比如 URL、AppID、密码、时间戳这几个名词，我们把它作为了方法的参数。

第二个细节：我们还需要挖掘一些没有出现在功能点描述中属性，比如 createTime, expireTimeInterval，它们用在 isExpired() 函数中，用来判定 token 是否过期。

第三个细节：我们还给 AuthToken 类添加了一个功能点描述中没有提到的方法 getToken()。

第一个细节告诉我们，从业务模型上来说，不应该属于这个类的属性和方法，不应该被放到这个类里。比如 URL、AppID 这些信息，从业务模型上来说，不应该属于 AuthToken，所以我们不应该放到这个类中。

第二、第三个细节告诉我们，在设计类具有哪些属性和方法的时候，不能单纯地依赖当下的需求，还要分析这个类从业务模型上来讲，理应具有哪些属性和方法。这样可以一方面保证类定义的完整性，另一方面不仅为当下的需求还为未来的需求做些准备。

## Uri 类相关的功能点有两个：

将 token、AppID、时间戳拼接到 URL 中，形成新的 URL；

解析 URL，得到 token、AppID、时间戳等信息。

虽然需求描述中，我们都是以 URL 来代指接口请求，但是，接口请求并不一定是以 URL 的形式来表达，还有可能是 Dubbo、RPC 等其他形式。为了让这个类更加通用，命名更加贴切，我们接下来把它命名为 ApiRequest。下面是我根据功能点描述设计的 ApiRequest 类。

| ApiRequest类   |
|---|
| 属性  |
| <pre>private String baseUrl;<br/>private String token;<br/>private String appId;<br/>private long timestamp;</pre>  |
| 构造函数  |
| <pre>public ApiRequest(String baseUrl, String token,<br/>String appId, long timestamp);</pre>   |
| 函数  |
| <pre>public static ApiRequest createFromFullUrl(String url);<br/><br/>public String getBaseUrl();<br/>public String getToken();<br/>public String getAppId();<br/>public long getTimestamp();</pre> |

CredentialStorage 类相关的功能点有一个：

从存储中取出 AppID 和对应的密码。

CredentialStorage 类非常简单，类图如下所示。为了做到抽象封装具体的存储方式，我们将 CredentialStorage 设计成了接口，基于接口而非具体的实现编程。

## CredentialStorage接口

### 接口函数

```
String getPasswordByAppId(String appId);
```

### 3. 定义类与类之间的交互关系

类与类之间都有哪些交互关系呢？UML 统一建模语言中定义了六种类之间的关系。它们分别是：泛化、实现、关联、聚合、组合、依赖。关系比较多，而且有些还比较相近，比如聚合和组合，接下来我就逐一讲解一下。

**泛化**（Generalization）可以简单理解为继承关系。具体到 Java 代码就是下面这样：

```
1 public class A { ... }  
2 public class B extends A { ... }
```

[复制代码](#)

**实现**（Realization）一般是指接口和实现类之间的关系。具体到 Java 代码就是下面这样：


```
1 public interface A {...}  
2 public class B implements A { ... }
```

[复制代码](#)

**聚合**（Aggregation）是一种包含关系，A 类对象包含 B 类对象，B 类对象的生命周期可以不依赖 A 类对象的生命周期，也就是说可以单独销毁 A 类对象而不影响 B 对象，比如课程与学




生之间的关系。具体到 Java 代码就是下面这样：

 复制代码


```
1 public class A {
2     private B b;
3     public A(B b) {
4         this.b = b;
5     }
6 }
```

**组合** (Composition) 也是一种包含关系。A 类对象包含 B 类对象，B 类对象的生命周期依赖 A 类对象的生命周期，B 类对象不可单独存在，比如鸟与翅膀之间的关系。具体到 Java 代码就是下面这样：

 复制代码

```
1 public class A {
2     private B b;
3     public A() {
4         this.b = new B();
5     }
6 }
```


**关联** (Association) 是一种非常弱的关系，包含聚合、组合两种关系。具体到代码层面，如果 B 类对象是 A 类的成员变量，那 B 类和 A 类就是关联关系。具体到 Java 代码就是下面这样：

 复制代码

```
1 public class A {
2     private B b;
3     public A(B b) {
4         this.b = b;
5     }
6 }
7 或者
8 public class A {
9     private B b;
10    public A() {
```

```
11     this.b = new B();
12 }
13 }
```

**依赖**（Dependency）是一种比关联关系更加弱的关系，包含关联关系。不管是 B 类对象是 A 类对象的成员变量，还是 A 类的方法使用 B 类对象作为参数或者返回值、局部变量，只要 B 类对象和 A 类对象有任何使用关系，我们都称它们有依赖关系。具体到 Java 代码就是下面这样：

 复制代码

```
1 public class A {
2     private B b;
3     public A(B b) {
4         this.b = b;
5     }
6 }
7 或者
8 public class A {
9     private B b;
10    public A() {
11        this.b = new B();
12    }
13 }
14 或者
15 public class A {
16     public void func(B b) { ... }
17 }
```

看完了 UML 六种类关系的详细介绍，不知道你有何感受？我个人觉得这样拆分有点太细，增加了学习成本，对于指导编程开发没有太大意义。所以，我从更加贴近编程的角度，对类与类之间的关系做了调整，只保留了四个关系：泛化、实现、组合、依赖，这样你掌握起来会更加容易。

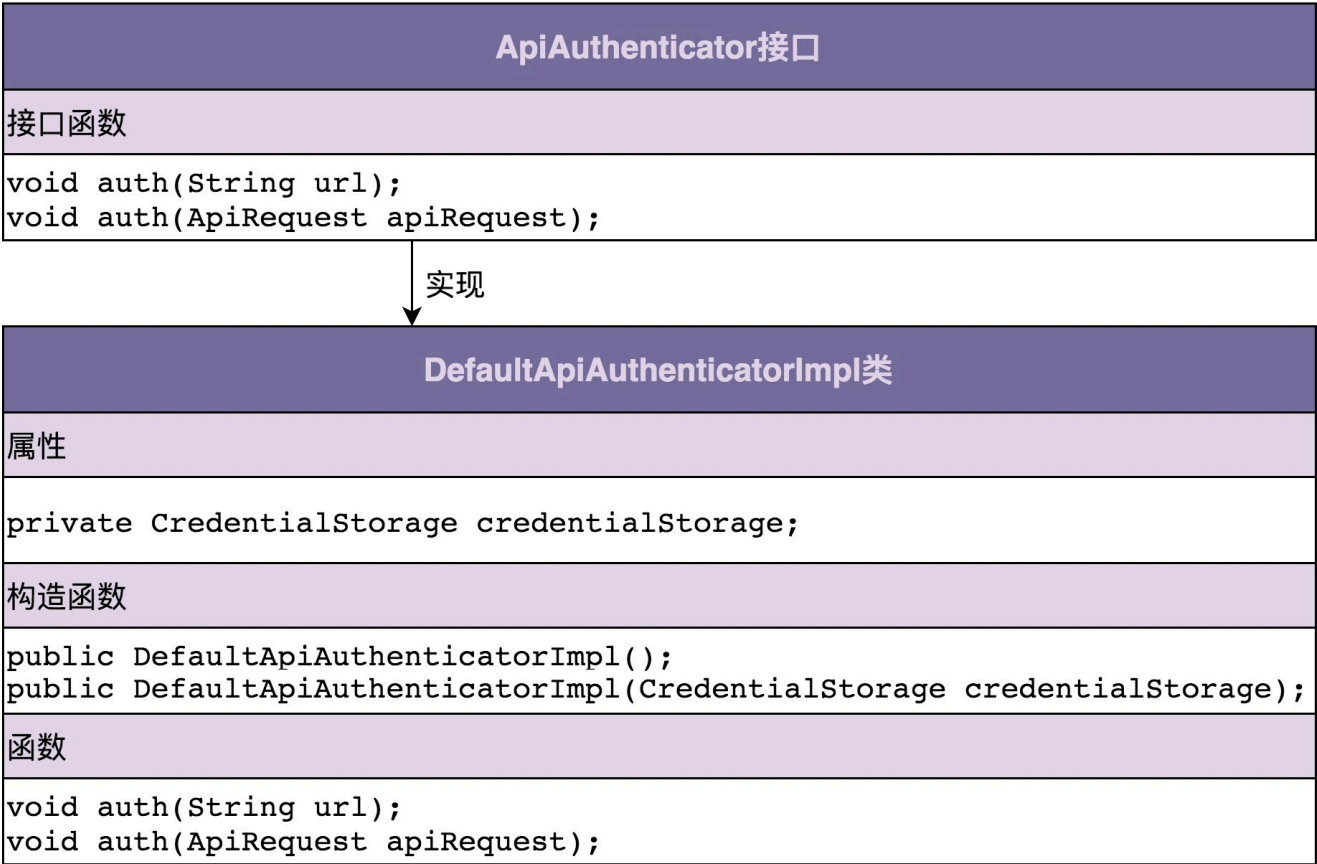
其中，泛化、实现、依赖的定义不变，组合关系替代 UML 中组合、聚合、关联三个概念，也就相当于重新命名关联关系为组合关系，并且不再区分 UML 中的组合和聚合两个概念。之所以这样重新命名，是为了跟我们前面讲的“多用组合少用继承”设计原则中的“组合”统一含义。只要 B 类对象是 A 类对象的成员变量，那我们就称，A 类跟 B 类是组合关系。

理论的东西讲完了，让我们来看一下，刚刚我们定义的类之间都有哪些关系呢？因为目前只有三个核心的类，所以只用到了实现关系，也即 `CredentialStorage` 和 `MysqlCredentialStorage` 之间的关系。接下来讲到组装类的时候，我们还会用到依赖关系、组合关系，但是泛化关系暂时没有用到。

#### 4. 将类组装起来并提供执行入口

类定义好了，类之间必要的交互关系也设计好了，接下来我们要将所有的类组装在一起，提供一个执行入口。这个入口可能是一个 `main()` 函数，也可能是一组给外部用的 API 接口。通过这个入口，我们能触发整个代码跑起来。


接口鉴权并不是一个独立运行的系统，而是一个集成在系统上运行的组件，所以，我们封装所有的实现细节，设计了一个最顶层的 `ApiAuthenticator` 接口类，暴露一组给外部调用者使用的 API 接口，作为触发执行鉴权逻辑的入口。具体的类的设计如下所示：



## 如何进行面向对象编程？

面向对象设计完成之后，我们已经定义清晰了类、属性、方法、类之间的交互，并且将所有的类组装起来，提供了统一的执行入口。接下来，面向对象编程的工作，就是将这些设计思路翻译成代码实现。有了前面的类图，这部分工作相对来说就比较简单了。所以，这里我只给出比较复杂的 ApiAuthenticator 的实现。

对于 AuthToken、ApiRequest、CredentialStorage 这三个类，在这里我就不给出具体的代码实现了。给你留一个课后作业，你可以试着把整个鉴权框架自己去实现一遍。

 复制代码

```
1 public interface ApiAuthenticator {
2     void auth(String url);
3     void auth(ApiRequest apiRequest);
4 }
5
6 public class DefaultApiAuthenticatorImpl implements ApiAuthenticator {
7     private CredentialStorage credentialStorage;
8
9     public DefaultApiAuthenticatorImpl() {
10         this.credentialStorage = new MysqlCredentialStorage();
11     }
12
13     public DefaultApiAuthenticatorImpl(CredentialStorage credentialStorage) {
14         this.credentialStorage = credentialStorage;
15     }
16
17     @Override
18     public void auth(String url) {
19         ApiRequest apiRequest = ApiRequest.buildFromUrl(url);
20         auth(apiRequest);
21     }
22
23     @Override
24     public void auth(ApiRequest apiRequest) {
25         String appId = apiRequest.getAppId();
26         String token = apiRequest.getToken();
27         long timestamp = apiRequest.getTimestamp();
28         String originalUrl = apiRequest.getOriginalUrl();
29
30         AuthToken clientAuthToken = new AuthToken(token, timestamp);
31         if (clientAuthToken.isExpired()) {
32             throw new RuntimeException("Token is expired.");
33         }
34     }
35 }
```

```
34
35     String password = credentialStorage.getPasswordByAppId(appId);
36     AuthToken serverAuthToken = AuthToken.generate(originalUrl, appId, password,
37     if (!serverAuthToken.match(clientAuthToken)) {
38         throw new RuntimeException("Token verification failed.");
39     }
40 }
41 }
```

## 辩证思考与灵活应用

在之前的讲解中，面向对象分析、设计、实现，每个环节的界限划分都比较清楚。而且，设计和实现基本上是按照功能点的描述，逐句照着翻译过来的。这样做的好处是先做什么、后做什么，非常清晰、明确，有章可循，即便是没有太多设计经验的初级工程师，都可以按部就班地参照着这个流程来做分析、设计和实现。

不过，在平时的工作中，大部分程序员往往都是在脑子里或者草纸上完成面向对象分析和设计，然后就开始写代码了，边写边思考边重构，并不会严格地按照刚刚的流程来执行。而且，说实话，即便我们在写代码之前，花很多时间做分析和设计，绘制出完美的类图、UML 图，也不可能把每个细节、交互都想得很清楚。在落实到代码的时候，我们还是要反复迭代、重构、打破重写。

毕竟，整个软件开发本来就是一个迭代、修修补补、遇到问题解决问题的过程，是一个不断重构的过程。我们没法严格地按照顺序执行各个步骤。这就类似你去学驾照，驾校教的都是比较正规的流程，先做什么，后做什么，你只要照着做就能顺利倒车入库，但实际上，等你开熟练了，倒车入库很多时候靠的都是经验和感觉。

## 重点回顾

今天的内容到此就讲完了。我们来一块总结回顾一下，你需要掌握的重点内容。

面向对象分析的产出是详细的需求描述。面向对象设计的产出是类。在面向对象设计这一环节中，我们将需求描述转化为具体的类的设计。这个环节的工作可以拆分为下面四个部分。

### 1. 划分职责进而识别出有哪些类

根据需求描述，我们把其中涉及的功能点，一个一个罗列出来，然后再去看哪些功能点职责相近，操作同样的属性，可否归为同一个类。

## 2. 定义类及其属性和方法

我们识别出需求描述中的动词，作为候选的方法，再进一步过滤筛选出真正的方法，把功能点中涉及的名词，作为候选属性，然后同样再进行过滤筛选。

## 3. 定义类与类之间的交互关系

UML 统一建模语言中定义了六种类之间的关系。它们分别是：泛化、实现、关联、聚合、组合、依赖。我们从更加贴近编程的角度，对类与类之间的关系做了调整，保留四个关系：泛化、实现、组合、依赖。

## 4. 将类组装起来并提供执行入口

我们要将所有的类组装在一起，提供一个执行入口。这个入口可能是一个 `main()` 函数，也可能是一组给外部用的 API 接口。通过这个入口，我们能触发整个代码跑起来。

## 课堂讨论

软件设计的自由度很大，这也是软件的复杂之处。不同的人对类的划分、定义、类之间交互的设计，可能都不大一样。那除了我今天给出的设计思路，你还有没有其他设计思路呢？

欢迎在留言区写下你的答案，和同学们一起交流和分享。如果有收获，也欢迎你把这篇文章分享给你的朋友。

### AI智能总结

本文介绍了如何利用面向对象设计和编程开发接口鉴权功能。首先，文章讲解了面向对象设计的步骤，包括划分职责、定义类及其属性和方法等。通过针对鉴权功能的实例，详细阐述了如何根据需求描述识别出相关类，并设计其属性和方法。文章强调了在设计类时需要考虑业务模型的完整性和未来需求的准备。此外，还介绍了面向对象设计在复杂需求开发中的应用方法，以及模块划分和识别类的套路。总的来说，本文通过实例详细解释了面向对象设计的思路和方法，对于开发人员理解面向对象设计和编程具有一定的指导意义。文章还提到了UML统一建模语言中定义的类之间的六种关系，以及如何将类组装起来并提供执行入口。最后，文章强调了在实际工作中，面向对象设计和编程是一个迭代、修修补补、不断重构的过程，需要灵活应用。

## 全部留言 (178)

最新 精选



斜杠青年 置顶

2019-12-09

生成Token的算法可以用不可逆的Hash算法吗？例如MD5或是SHA？

作者回复: 可以的，我就是用的不可逆的

共 5 条评论 >

👍 8



大大。

2019-12-04

争哥：设计了一个最顶层的 ApiAuthencator 接口类，为什么要设计出一个顶层的接口，虽然是面向接口编程，但是写成接口，意味着还会有其他的不同实现吗，如果有其他不同的实现，那么接口中的第二个参数的ApiRequest也不一定会适用不同的那个实现的传参，不知道说明白没有

作者回复: 你说得很好，多谢指出啊，确实没必要搞个接口，ApiRequest的设计也依赖了具体的url实现，所以也不是很通用。

共 5 条评论 >

👍 66



ThinkingQuest

2020-08-29

ApiAuthenticator接口的两个方法，都声明为返回void，用Exception来控制token过期或不合法，这样好吗？

token过期或不合法，在业务上应该是一种正常的，需要程序逻辑分支来处理的情形吧，用throw runtime exception的方式来处理，感觉上等于是把exception用于控制执行流程了。

关于这个，不知道应该怎么考虑。

作者回复: 嗯嗯，你说的也对，那应该用boolean值作为返回值吗？那携带的信息就很少呀，只有true、false，不知道失败的原因。我觉得选择exception就是一种设计上的权衡。



👍 8



瑞泉

2019-12-16

王争老师，代码是在这个地址吗？<https://github.com/wangzheng0822/codedesign>，是空的

作者回复: 是的 还没空整理 稍等一阵子吧

共 6 条评论 >

👍 5



grey927

2020-07-25

王老师，你好，我有个问题，针对：AuthToken中，isExpired()方法，这里返回的是布尔类型，那么，如果我传的是

- 错误的token
- 正确的token但是过期

这两类都会判断成false，那么我怎么知道我的token是格式问题还是过期问题呢？

作者回复: isExpired()只用来判断是否过期，match()用来判断是否错误



👍 3



unreal

2019-12-07

有个问题，ApiRequest叫法虽然更加通用，但实际创建接口命名还是依赖实现了吧，都是基于url的，还是我理解问题？

作者回复: 多谢建议，你说的没错，ApiRequest确实还是依赖了url

共 3 条评论 >

👍 2



航哥很帅

2020-11-13



面型对象设计主要分为4个步骤：

- 1.根据需求设计需要的类；
- 2.确定类中的属性和方法，确定属性一般根据需求分析中的名称，确定方法一般根据需求分析中的动词；
- 3.确定类和类之间的关系，这些关系一般包括：泛化（继承）、实现、组合、依赖；
- 4.将类封装起来并提供执行的入口，这个入口就是这段代码运行起来的入口。

作者回复: 总结的到位



兔嘟嘟

2020-06-22

看了几遍还是没懂，AuthToken类为什么构造函数要传入一个token，token不应该是类内生成的吗？以及如果已经通过构造函数获得了一个token，那么create函数又有什么作用？

作者回复: 既支持外界生成好token，传递进来，创建AuthToken，比如从客户端传来的token，我们在后端包裹成AuthToken继续处理。也支持直接根据基础数据生成AuthToken,也就是create函数的作用



tt

2019-12-04

本讲中的面向对象分析和面向对象设计在阅读源码的时候可以使用么？就是先感知源码的整体设计，让后自顶向下的剖析源码

作者回复: 先了解整体结构，再了解细节，思路是对的。



WolvesLeader

2020-11-29

大佬，我想问一下，怎么改成提供token的服务呢

作者回复: 没怎么看懂

共 2 条评论>

