# 简介

依赖注入（Dependency Injection，DI）容器就是一个对象，它知道怎样初始化并配置对象及其依赖的所有对象。

[Martin 的文章](http://martinfowler.com/articles/injection.html) 已经解释了 DI 容器为什么很有用。 这里我们主要讲解 Yii 提供的 DI 容器的使用方法。

# 依赖注入（Dependency Injection）

Yii 通过 [yii\di\Container](https://www.yiichina.com/doc/api/2.0/yii-di-container) 类提供 DI 容器特性。 它支持如下几种类型的依赖注入：

* 构造方法注入;
* 方法注入;
* Setter 和属性注入;
* PHP 回调注入.

所谓依赖注入其实就是把当前对象需要的其它对象从外面通过各种方法传给当前对象。

如果不选择依赖注入，那么当前对象就需要在自己内部实例化所需对象，相对而言麻烦些，且不利于维护。

## 构造方法注入（Constructor Injection）

在参数类型提示的帮助下，DI 容器实现了构造方法注入。

当容器被用于创建一个新对象时， 类型提示会告诉它要依赖什么类或接口。 容器会尝试获取它所依赖的类或接口的实例， 然后通过构造器将其注入新的对象。例如：

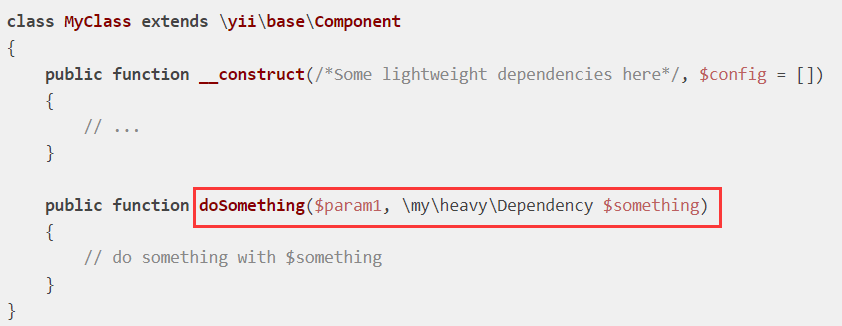


## 方法注入（Method Injection）

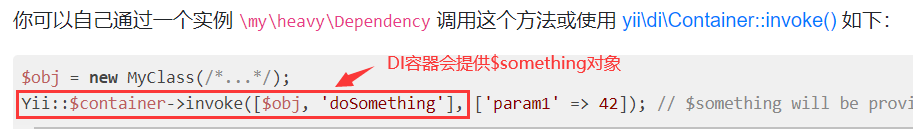
通常，类的依赖关系传递给构造函数，并且在整个生命周期中都可以在类内部使用。

通过方法注入，可以提供仅由类的单个方法需要的依赖关系， 并将其传递给构造函数可能不可行，或者可能会在大多数用例中导致太多开销。

类方法可以像下面例子中的 doSomething() 方法一样定义：



你可以自己通过一个实例 \my\heavy\Dependency 调用这个方法或使用 [yii\di\Container::invoke()](https://www.yiichina.com/doc/api/2.0/yii-di-container" \l "invoke()-detail) 如下：



## Setter 和属性注入（Setter and Property Injection）

Setter 和属性注入是通过[配置](https://www.yiichina.com/doc/guide/2.0/concept-configurations)提供支持的。

 当注册一个依赖或创建一个新对象时，你可以提供一个配置， 该配置会提供给容器用于通过相应的 Setter 或属性注入依赖。 例如：



其实就是属性注入。

## PHP 回调注入（PHP Callable Injection）

在这种情况下，容器将使用已注册的 PHP 回调来构建类的新实例。

每次调用 [yii\di\Container::get()](https://www.yiichina.com/doc/api/2.0/yii-di-container" \l "get()-detail) ，相应的回调将被调用。 调用方负责解析依赖项，并适当地将它们注入到新创建的对象中。 例如：



要省略构建新对象的复杂逻辑，可以使用静态类方法作为可调用的方法。例如：



这样做的话，想要配置 Foo 类的人不再需要知道它是如何构建的。

感觉回调注入没什么使用场景，太麻烦了。

# 注册依赖关系（Registering Dependencies）

可以用 [yii\di\Container::set()](https://www.yiichina.com/doc/api/2.0/yii-di-container" \l "set()-detail) 注册依赖关系。

注册会用到一个依赖关系名称和一个依赖关系的定义。

依赖关系名称可以是一个类名，一个接口名或一个别名。 依赖关系的定义可以是一个类名，一个配置数组，或者一个 PHP 回调。





****提示：****如果依赖关系名称和依赖关系的定义相同， 则不需要通过 DI 容器注册该依赖关系。

通过 set() 注册的依赖关系，在每次使用时都会产生一个新实例。 可以使用 [yii\di\Container::setSingleton()](https://www.yiichina.com/doc/api/2.0/yii-di-container" \l "setSingleton()-detail) 注册一个单例的依赖关系：



# 解决依赖关系（Resolving Dependencies）

注册依赖关系后，就可以使用 DI 容器创建新对象了。

容器会自动解决依赖关系， 将依赖实例化并注入新创建的对象。

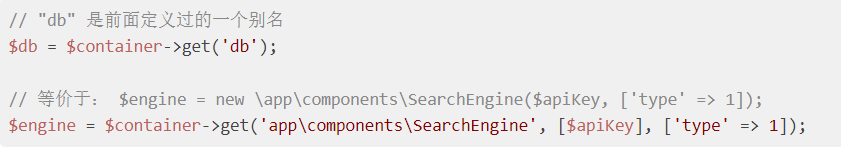
依赖关系的解决是递归的， 如果一个依赖关系中还有其他依赖关系， 则这些依赖关系都会被自动解决。

可以使用 [yii\di\Container::get()](https://www.yiichina.com/doc/api/2.0/yii-di-container" \l "get()-detail) 创建新的对象。

该方法接收一个依赖关系名称，它可以是一个类名， 一个接口名或一个别名。

依赖关系名或许是通过 set() 或 setSingleton() 注册的。 你可以随意地提供一个类的构造器参数列表和一个 [configuration](https://www.yiichina.com/doc/guide/2.0/concept-configurations) 用于配置新创建的对象。

例如：



代码背后，DI 容器做了比创建对象多的多的工作。 容器首先将检查类的构造方法，找出依赖的类或接口名， 然后自动递归解决这些依赖关系。

如下代码展示了一个更复杂的示例。UserLister 类依赖一个实现了 UserFinderInterface 接口的对象； UserFinder 类实现了这个接口，并依赖于一个 Connection 对象。 所有这些依赖关系都是通过类构造器参数的类型提示定义的。 通过属性依赖关系的注册，DI 容器可以自动解决这些依赖关系并能通过一个 简单的 get('userLister') 调用创建一个新的 UserLister 实例。





# 实践中的运用（Practical Usage）

当在应用程序的[入口脚本](https://www.yiichina.com/doc/guide/2.0/structure-entry-scripts)中引入 Yii.php 文件时， Yii 就创建了一个 DI 容器。这个 DI 容器可以通过 [Yii::$container](https://www.yiichina.com/doc/api/2.0/yii-baseyii" \l "$container-detail) 访问。

当调用 [Yii::createObject()](https://www.yiichina.com/doc/api/2.0/yii-baseyii" \l "createObject()-detail) 时，此方法实际上会调用这个容器的 [get()](https://www.yiichina.com/doc/api/2.0/yii-di-container" \l "get()-detail) 方法创建新对象。

如上所述，DI 容器会自动解决依赖关系（如果有）并将其注入新创建的对象中。 因为 Yii 在其多数核心代码中都使用了[Yii::createObject()](https://www.yiichina.com/doc/api/2.0/yii-baseyii" \l "createObject()-detail) 创建新对象， 所以你可以通过 [Yii::$container](https://www.yiichina.com/doc/api/2.0/yii-baseyii" \l "$container-detail) 全局性地自定义这些对象。

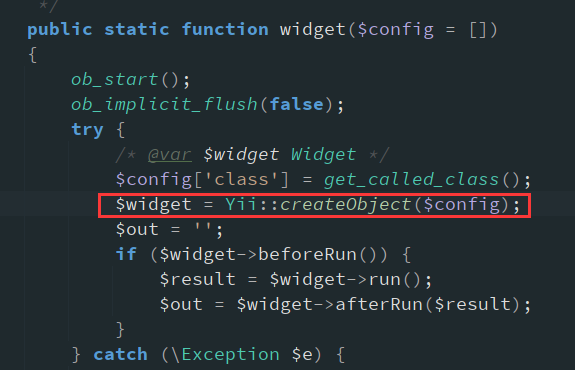
例如，你可以全局性自定义 [yii\widgets\LinkPager](https://www.yiichina.com/doc/api/2.0/yii-widgets-linkpager) 中分页按钮的默认数量:

\Yii::$container->set('yii\widgets\LinkPager', ['maxButtonCount' => 5]);

这样如果你通过如下代码在一个视图里使用这个挂件， 它的 maxButtonCount 属性就会被初始化为 5 而不是类中定义的默认值 10。

**echo** \yii\widgets\LinkPager::widget();

因为widget()方法中也调用了[Yii::createObject()](https://www.yiichina.com/doc/api/2.0/yii-baseyii" \l "createObject()-detail) 创建新对象。



另一个例子是借用 DI 容器中自动构造方法注入带来的好处。 假设你的控制器类依赖一些其他对象，例如一个旅馆预订服务。 你可以通过一个构造器参数声明依赖关系，然后让 DI 容器帮你自动解决这个依赖关系。



如果你从浏览器中访问这个控制器，你将看到一个报错信息，提醒你 BookingInterface 无法被实例化。 这是因为你需要告诉 DI 容器怎样处理这个依赖关系。

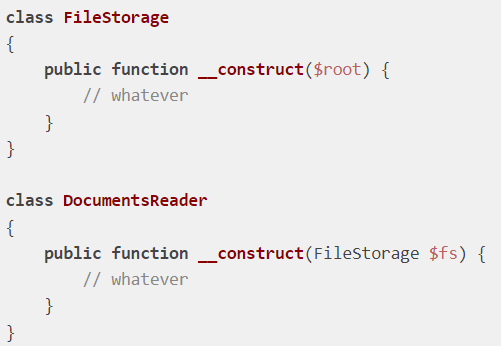


现在如果你再次访问这个控制器，一个 app\components\BookingService 的实例就会被创建并被作为第三个参数注入到控制器的构造器中。

# 高级实用性（Advanced Practical Usage）

比如说我们在 API 应用方面工作：

* app\components\Request 类继承了 yii\web\Request 并提供了额外的功能
* app\components\Response 类继承了 yii\web\Response 并且在创建时应该将 format 属性设置为 json
* app\storage\FileStorage 和 app\storage\DocumentsReader 用于处理位于某些文件存储中的文档的某些逻辑：



可以一次配置多个定义，将配置数组传递给 [setDefinitions()](https://www.yiichina.com/doc/api/2.0/yii-di-container" \l "setDefinitions()-detail) 或 [setSingletons()](https://www.yiichina.com/doc/api/2.0/yii-di-container" \l "setSingletons()-detail) 方法。 遍历配置数组，将分别为每个对象分别调用 [set()](https://www.yiichina.com/doc/api/2.0/yii-di-container" \l "set()-detail) 或 [setSingleton()](https://www.yiichina.com/doc/api/2.0/yii-di-container" \l "setSingleton()-detail) 方法。

配置数组格式为：

* key：类名称，接口名称或别名。 该 key 将作为第一个参数 $class 传递给 [set()](https://www.yiichina.com/doc/api/2.0/yii-di-container" \l "set()-detail) 方法。
* value：与 $class 关联的定义。$definition 参数的值可能在 [set()](https://www.yiichina.com/doc/api/2.0/yii-di-container" \l "set()-detail) 文档中描述。$definition 将作为第二个参数传递给 set() 方法。

例如，让我们配置我们的容器以遵循上述要求：



一切正常，但如果我们需要创建 Document Writer 类， 我们将复制粘贴创建 FileStorage 对象的行，这显然不是最聪明的方式。

如 [解决依赖关系](https://www.yiichina.com/doc/guide/2.0/concept-di-container" \l "resolving-dependencies) 子节中所述，[set()](https://www.yiichina.com/doc/api/2.0/yii-di-container" \l "set()-detail) 和 [setSingleton()](https://www.yiichina.com/doc/api/2.0/yii-di-container" \l "setSingleton()-detail) 可以选择将依赖项的构造函数参数作为第三个参数。 要设置构造函数参数，可以使用以下配置数组格式：

* key：类名称，接口名称或别名。该 key 将作为第一个参数 $class 传递给 [set()](https://www.yiichina.com/doc/api/2.0/yii-di-container" \l "set()-detail) 方法。
* value：两个元素的数组。第一个元素将传递给 [set()](https://www.yiichina.com/doc/api/2.0/yii-di-container" \l "set()-detail) 方法 作为第二个参数 $definition，第二个元素为 $params。

让我们来修改我们的例子：



你可能会注意到 Instance::of('tempFileStorage') 符号。这意味着，[Container](https://www.yiichina.com/doc/api/2.0/yii-di-container) 将隐含地提供一个用 tempFileStorage 名称注册的依赖项， 并将其作为 app\storage\DocumentsWriter 构造函数的第一个参数传递。

配置优化的另一个步骤是将某些依赖项注册为单例。 通过 [set()](https://www.yiichina.com/doc/api/2.0/yii-di-container" \l "set()-detail) 注册的依赖项将在每次需要时实例化。 某些类在运行时不会更改状态， 因此它们可能会被注册为单例以提高应用程序的性能。

一个很好的例子可以是 app\storage\FileStorage 类，它用一个简单的 API （例如 $fs->read()，$fs->write()）在文件系统上执行一些操作。 这些操作不会更改内部类的状态，因此我们可以创建一次实例并多次使用它。



# 什么时候注册依赖关系（When to Register Dependencies）

由于依赖关系在创建新对象时需要解决，因此它们的注册应该尽早完成。 如下是推荐的实践：

* 如果你是一个应用程序的开发者， 你可以在应用程序的[入口脚本](https://www.yiichina.com/doc/guide/2.0/structure-entry-scripts) 或者被入口脚本引入的脚本中注册依赖关系。
* 如果你是一个可再分发[扩展](https://www.yiichina.com/doc/guide/2.0/structure-extensions)的开发者， 你可以将依赖关系注册到扩展的引导类中。

# 总结（Summary）

依赖注入和[服务定位器](https://www.yiichina.com/doc/guide/2.0/concept-service-locator)都是流行的设计模式， 它们使你可以用充分解耦且更利于测试的风格构建软件。

强烈推荐你阅读 [Martin 的文章](http://martinfowler.com/articles/injection.html) ， 对依赖注入和服务定位器有个更深入的理解。

Yii 在依赖住入（DI）容器之上实现了它的[服务定位器](https://www.yiichina.com/doc/guide/2.0/concept-service-locator)。 当一个服务定位器尝试创建一个新的对象实例时，它会把调用转发到 DI 容器。 后者将会像前文所述那样自动解决依赖关系。