50. 容器和依赖注入

学习要点：

1.依赖注入

2.容器

本节课我们来学习一下依赖注入的用法，以及容器的用法。

一. 依赖注入

1. 手册对依赖注入比较严谨的说明，具体如下：

依赖注入其实本质上是指对类的依赖通过构造器完成自动注入，例如在控制器架构方法和操作方法中一旦对参数进行对象类型约束则会自动触发依赖注入，由于访问控制器的参数都来自于 URL 请求，普通变量就是通过参数绑定自动获取，对象变量则是通过依赖注入生成。

2. 先看一个小例子，了解一下依赖注入的写法，创建一个模型；

```php
namespace app\model;
use think\Model;

class One extends Model
{
    public $name = 'Mr.Lee';
}
```

3. 创建一个控制器 Inject，通过依赖注入将模型 One 对象引入其内；

```php
namespace app\controller;
use app\model\One;

class Inject
{
    protected $one;
    public function __construct(One $one)
    {
        $this->one = $one;
    }
    public function index()
    {
        return $this->one->name;
    }
}
```

4. 依赖注入：即允许通过类的方法传递对象的能力，并且限制了对象的类型(约束)；
5. 而传递的对象背后的那个类被自动绑定并且实例化了，这就是依赖注入；

二. 容器

1. 依赖注入的类统一由容器管理的，大多数情况下是自动绑定和自动实例化的；
2. 如果想手动来完成绑定和实例化，可以使用 bind()和 app()助手函数来实现；

```
class Inject
{
    public function index()
    {
        bind('one', 'app\model\One');
        return app('one')->name;
    }
}
```

6. bind('one','...')绑定类库标识，这个标识具有唯一性，以便快速调用；
7. app('one')快速调用，并自动实例化对象，标识严格保持一致包括大小写；
8. 自动实例化对象的方式，是采用单例模式实现，如果想重新实例化一个对象，则：

```
//每次调用总是会重新实例化
$one = app('one', true);
return $one->name;
```

9. 当然，你也可以直接通过 app()绑定一个类到容器中并自动实例化；

```
return app('app\model\One')->name;
```

10. 使用 bind([])可以实现批量绑定，只不过系统有专门提供批量绑定的文件；

```
bind([
    'one'       =>      'app\model\One',
    'user'      =>      'app\model\User'
]);
return app('one')->name;

bind([
    'one'       =>      \app\model\One::class,
    'user'      =>      \app\model\User::class
]);
return app('user')->name;
```

11. ::class 模式，不需要单引号，而且需要在最前面加上\，之前的加不加都行；
12. 系统提供了 provider.php 文件，用于批量绑定类到容器中，这里不加不报错；

```
return [
    'one'       =>      app\model\One::class,
    'user'      =>      app\model\User::class
];
```

13．系统内置了很多常用的类库，以便开发者快速调用，具体如下：

系统类库	容器绑定标识
think\Build	build
think\Cache	cache
think\Config	config
think\Cookie	cookie
think\Debug	debug
think\Env	env
think\Hook	hook
think\Lang	lang
think\Log	log
think\Request	request
think\Response	response
think\Route	route
think\Session	session
think\Url	url
think\Validate	validate
think\View	view

```
return app('request')->param('name');
```

14．大家会发现，这个不是我们之前学过的 Request::param()么？对！

15．也就是说，实现同一个效果可以由容器的 bind()和 app()实现，也可以使用依赖注入实现，还有 Facade(下节课重点探讨)实现，以及助手函数实现；