Jeffrey Palermo 8:14 am em 30 de julho de 2008 Tags: onion architecture (4)

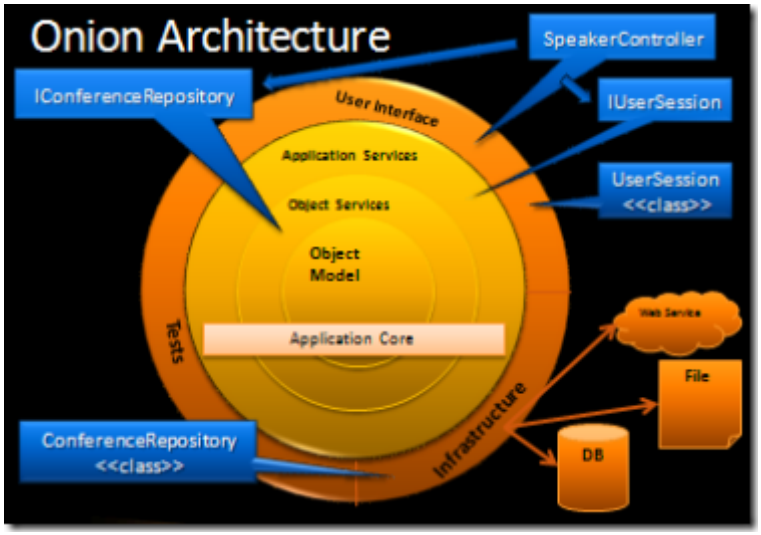
A arquitetura da cebola: parte 2

parte 1 . Esta é a parte 2 . parte 3 . parte 4 . Meu feed (rss) .

Na parte 1 , apresentei um padrão de arquitetura que chamei de “Onion Architecture”. Os conceitos de design orientado a objetos não são novos, mas estou reunindo muitas técnicas e convenções em um único padrão e dando um nome a ele. Minha esperança é que a indústria possa usar esse nome para comunicar a abordagem arquitetônica quando apropriado.

Parte 2: Exemplo prático:

CodeCampServer usa a arquitetura Onion. Se você estiver procurando por um aplicativo completo e funcional como exemplo, dê uma olhada. O exemplo prático que coloquei antes de você foi tirado diretamente do CodeCampServer. É uma fatia estreita e vertical de um exemplo. Estou mantendo o escopo pequeno para ser digerível. Vou começar com um diagrama para que você possa entender onde todo o código reside nas camadas da cebola.



CodeCampServer usa o ASP.NET MVC Framework, então SpeakerController faz parte da interface do usuário. Esse controlador é acoplado ao ASP.NET MVC Framework e não há como contornar isso. SpeakerController depende de IConferenceRepository e IUserSession (e IClock, mas vamos omitir isso). O controlador depende apenas de interfaces, que são definidas no núcleo do aplicativo. Lembre-se que **todas as dependências estão voltadas para o centro** .

Volte sua atenção para as classes ConferenceRepository e UserSession. Observe que eles estão em camadas fora do núcleo do aplicativo e também dependem das interfaces, para que possam implementá-los. Essas duas classes implementam uma interface mais próxima do centro do que ela mesma. Em tempo de execução, nosso contêiner Inversion of Control examinará seu registro e construirá as classes apropriadas para satisfazer as dependências do construtor de SpeakerController, que é o seguinte:

```
public SpeakerController(IConferenceRepository conferenceRepository,
                         IUserSession userSession, relógio IClock)
    : base(userSession)
{
    _conferenceRepository = conferenceRepository;
    _relógio = relógio;
    _userSession = userSession;
}
```

Em tempo de execução, o contêiner IoC resolverá as classes que implementam interfaces e as passará para o construtor SpeakerController. Neste momento, o SpeakerController pode fazer seu trabalho.

Com base nas regras da Onion Architecture, o SpeakerController *poderia* usar UserSession diretamente, pois está na mesma camada, mas não pode usar ConferenceRepository diretamente. Ele deve depender de algo externo passando em uma instância de IConferenceRepository. Esse padrão é usado por toda parte, e o contêiner IoC torna esse processo perfeito.

No final desta série, pretendo publicar um sistema funcional completo que siga o padrão Onion Architecture. Os sistemas que construímos para clientes usam essa abordagem, mas não tenho liberdade para discutir esse código, portanto, criarei um aplicativo de referência para aqueles que preferem uma solução concreta do Visual Studio para digerir.