novática
Revista de la Asociación de Técnicos de Informática

Novática, revista fundada en 1975 y decana de la prensa informática española, es el órgano oficial de expresión y formación continua de **ATI** (Asociación de Técnicos de Informática), organización que edita también la revista **REICIS** (Revista Española de Innovación, Calidad e Ingeniería del Software).

<http://www.ati.es/novatica/>
<http://www.ati.es/reicis/>

ATI es miembro fundador de **CEPIS** (*Council of European Professional Informatics Societies*) y es representante de España en **IFIP** (*International Federation for Information Processing*); tiene un acuerdo de colaboración con **ACM** (*Association for Computing Machinery*), así como acuerdos de vinculación o colaboración con **AdaSpain**, **AI2**, **ASTIC**, **RITSI** e **Hispalinux**, junto a la que participa en **ProInnova**.

Coordinación Editorial, Redacción Central y Redacción ATI Madrid
Padilla 66, 3º, dcha., 28006 Madrid
Tlfn.914029391; fax.913093685 <novatica@ati.es>
Composición, Edición y Redacción ATI Valencia
Av. del Reino de Valencia 23, 46005 Valencia
Tlfn. 963740173 <novatica_prod@ati.es>
Administración y Redacción ATI Cataluña
Via Laietana 46, ppal. 1ª, 08003 Barcelona
Tlfn.934125235; fax 934127713 <secregen@ati.es>
Redacción ATI Aragón
Lagasca 9, 3-B, 50006 Zaragoza.
Tlfn./fax 976235181 <secreara@ati.es>
Redacción ATI Andalucía <secreand@ati.es>
Redacción ATI Galicia<secregal@ati.es>
Suscripción y Ventas <novatica.subscripciones@atinet.es>
Publicidad Padilla 66, 3º, dcha, 28006 Madrid.
Tlnf.914029391; fax.913093685 <novatica@ati.es>
Imprenta: Derra S.A., Juan de Austria 66, 08005 Barcelona.
Depósito legal: B 15.154-1975 -- ISSN: 0211-2124; CODEN NOVAEC
Portada: Escalera de color - Concha Arias Pérez / © ATI
Diseño: Fernando Agresta / © ATI 2003

Nº 221, enero-febrero 2013, año XXXIX

sumario

Tema del próximo número: "Lenguajes de programación"

Cormoran: Un *framework* de persistencia para Python

Jaime Gil de Sagredo Luna
Desarrollador de servicios REST y aplicaciones de gestión en Taric, S.A., ganador del VI Concurso Universitario de Software Libre en la categoría "Mejor proyecto de Innovación"

<jaimegildesagredo@gmail.com>

Resumen: *Cormoran es un* framework *de persistencia y ORM (*Object-Relational Mapping*) para el lenguaje de programación Python. Su principal objetivo es proveer de un API moderna y potente para el acceso a datos de aplicaciones con independencia de la fuente de datos. Actualmente se ha liberado una primera versión de la librería y se está trabajando en mejorarla, además de incorporar nuevas fuentes de datos, como APIs REST.*

Palabras clave: *Aplicaciones, Cormoran, datos, desarrollo, ORM, persistencia, Python.*

Autor

Jaime Gil de Sagredo Luna es estudiante de Ingeniería Informática en la Universidad Nacional de Educación a Distancia y desarrollador de servicios REST y aplicaciones de gestión en Taric, S.A. Empezó en el desarrollo de software mucho antes de entrar en la universidad, principalmente desarrollando aplicaciones Open Source para entornos Linux. Conoció Python al poco tiempo y desde entonces ha desarrollado principalmente en este lenguaje. Actualmente mantiene varias librerías *Open Source*, además de contribuir en varios proyectos abiertos. Es un apasionado del buen código y de las metodologías ágiles (TDD, BDD, *Continuous Delivery*, etc.). Su blog personal se encuentra en <http://jaimegildesagredo.github.com>.

1. Introducción

Cormoran es un *framework* de persistencia y ORM[1] para Python. Su objetivo es acelerar el desarrollo de aplicaciones con acceso a datos proporcionando un API[2] sencilla y potente, y mejorar la velocidad de consumición de datos de estas aplicaciones en producción.

El proyecto surgió como alternativa a *frameworks* como SQLAlchemy, Django ORM o MongoEngine. Comenzó su desarrollo para la V edición del Concurso Universitario de Software Libre, y siguió durante la edición siguiente [1][2][3].

2. ¿Por qué desarrollar otro ORM?

Cormoran nace por la necesidad de una capa de abstracción potente sobre distintos motores de persistencia de datos, como bases de datos SQL y noSQL, sistemas de ficheros, etc. Aunque como hemos dicho existen otras alternativas, éstas suelen enfocarse a un tipo de persistencia en concreto. Por ejemplo, SQLAlchemy o el ORM de Django son mapeadores objecto-relacional, es decir, trabajan exclusivamente con bases de datos relacionales.

Por otro lado, tenemos persistencia sobre el sistema de ficheros en la librería estándar de Python (Pickle), pero por lo general no es lo suficientemente potente para datos y estructuras complejas. Por último, con la llegada de las bases de datos no relacionales (noSQL) han surgido librerías que crean una abstracción entre dichas bases de datos y objetos Python pero, como pasa en el caso de los ORM, solo soportan este tipo de base de datos.

Como desarrolladores disponemos de una gran variedad de soluciones para persistir nuestros datos, pero ninguna termina de ser todo lo flexible que nos gustaría. Cuando comenzamos un proyecto en el que debemos guardar información en una base de datos, lo normal es empezar con una base de datos relacional, como MySQL o PostgreSQL, o incluso SQLite durante el desarrollo. Esto nos da muchas ventajas a la hora de desarrollar e incluso de poner en producción nuestra aplicación, pero ¿qué pasa si nuestra aplicación empieza a tener más usuarios, el tráfico aumenta y los datos almacenados cada vez son mayores?

Es muy posible que nuestra base de datos relacional empiece a darnos problemas y lleguemos a la conclusión de que debemos migrar a una solución más escalable, como por ejemplo MongoDB.

En este punto, y con las soluciones actuales, deberíamos no solo migrar nuestros datos de una base de datos a otra, sino que además deberíamos reescribir buena parte de nuestro código. Todo aquel código que interactúe con la base de datos, aunque sea a través de un ORM. Además de que reescribir todo este código es realmente costoso y conlleva un gran trabajo, pueden surgir otros problemas derivados de reimplementar parte de nuestro código, trabajar con un nuevo *framework*, etc. ¿Os imagináis poder desarrollar una aplicación completamente agnóstica del sistema de persistencia que vaya a utilizar? Incluso si éste cambia a lo largo de la vida de nuestra aplicación.

Además de la homogeneización de sistemas de persistencia, Cormoran pretende proveer a los desarrolladores de un API clara, potente y consistente. Al final a nosotros lo único que nos debería importar es que estamos guardando, leyendo o borrando algún dato de la aplicación, no si es una base de datos relacional, si es PostgreSQL o una base de datos documental (ver **figura 1**). Aun así, es evidente que no todas las aplicaciones tienen los mismos requisitos ni problemas, por lo que es necesario cierto control y flexibilidad.

En definitiva, Cormoran debería darnos la posibilidad de escribir aplicaciones sencillas de forma rápida pero sin perder de vista la flexibilidad para construir aplicaciones grandes y complejas.

3. Objetivos para el desarrollo

Desde un primer momento intentamos marcar una serie de objetivos para desarrollar a lo largo del concurso. Realmente había muchas cosas que hacer y éramos conscientes de que no era un trabajo fácil. Había que saber en cada momento de la dirección del proyecto y de los puntos a tratar en cada iteración.

Para ello, en la primera versión se fijaron una serie de funcionalidades, como dar soporte para definir los modelos de datos de forma declarativa en Python, como ya hacen otras librerías. Por otra parte, era básico dar soporte CRUD[3], es decir, que pudiéramos insertar, modificar, leer y borrar información de la base de datos. Y por último dar soporte al menos a una base de datos relacional, que en este caso sería SQLite por su sencillez y por estar integrada en la librería estándar de Python.

Para futuras iteraciones se pensó en dar soporte para cachear respuestas de la base de datos, así como dar soporte a varios *backends* de caché. Añadir soporte para trabajar con datos relacionales, tanto la definición de sus modelos como el proceso de manipulación de los mismos. Y poco a poco ir añadiendo más *backends* de persistencia, además de crear un pequeño framework para la definición de los mismos.

> “Además de la homogeneización de sistemas de persistencia, Cormoran pretende proveer a los desarrolladores de un API clara, potente y consistente”

A lo largo del desarrollo fuimos priorizando muchos de estos objetivos según iban apareciendo más o menos necesidades.

4. El proceso de desarrollo

Para el proceso de desarrollo se tomaron una serie de decisiones que afectarían directamente a la consecución de los objetivos de la mejor manera posible. Era necesario llevar a cabo un desarrollo ágil y poder liberar versiones en cuanto se cumpliese la lista de objetivos. Además, al tratarse de una librería para desarrolladores era muy importante la estabilidad y robustez de la misma.

De esta forma se consideró necesaria una buena batería de pruebas unitarias y de integración, además de sistema de integración continua con pruebas en distintas plataformas para automatizar la compilación, la ejecución de las pruebas y la generación de paquetes.

El desarrollo se llevó a cabo por medio de *Test Driven Development*[4], lo que aceleró el desarrollo además de dotar el resultado de bastante calidad. Así mismo se utilizó el servicio de integración continua[5] Travis junto con Github para mantener todas estas pruebas siempre en verde en distintas versiones de Python.

Actualmente se ha liberado una primera versión del *framework* con todas las funcionalidades que se habían planeado para esa iteración. Aun así, se trata de una versión beta que sirve como punto de inflexión en el desarrollo del proyecto, pero que en ningún caso lo convierte en un software realmente estable y para uso en sistemas en producción. Además se ha seguido trabajando en esta primera versión solucionando *bugs* y añadiendo un backend para bases de datos MySQL.

Durante el desarrollo siempre se ha intentado añadir valor a la comunidad publicando documentación y tutoriales, artículos sobre el desarrollo del proyecto y contribuciones a proyectos *Open Source* con los que trabajamos para desarrollar Cormoran.

Además, hemos conseguido bastante repercusión entre la comunidad Python, a pesar de ser un proyecto bastante joven, y se ha puesto en producción en algunas aplicaciones como capa de acceso a datos.

5. Estado del proyecto

Debido a lo rápido que empezaba a crecer el proyecto, actualmente se ha dividido en varias partes.

Por un lado se ha extraído una librería para el modelado y validación de los datos. Este proyecto, llamado Booby [4], se ha liberado con licencia Apache2.

Por otro lado se ha creado otra librería que comparte el API de Cormoran, pero que se dedica a la consumición de datos a través de APIs HTTP, en concreto APIs REST. Este último proyecto es Finch [5] y también ha sido liberado con licencia Apache2.

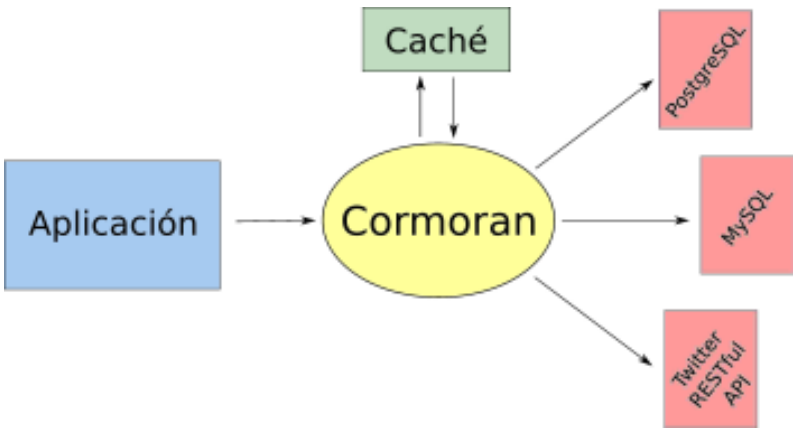

Figura 1. Utilización de Cormoran independiente del sistema de persistencia final.

Ahora la idea es evolucionar estas dos librerías y seguir mejorando la capa de acceso a bases de datos relacionales, así como añadir soporte a bases de datos no relacionales y demás sistemas de persistencia.

Referencias

[1] Blog del proyecto, <http://cormoran-project.blogspot.com.es>.
[2] Documentación, <http://cormoran.nhopkg.org/docs>.
[3] Fuentes de Cormoran, <https://github.com/jaimegildesagredo/cormoran>.
[4] Fuentes de Booby, <https://github.com/jaimegildesagredo/booby>.
[5] Fuentes de Finch, <https://github.com/jaimegildesagredo/finch>.

Notas

[1] El mapeo objeto-relacional (más conocido por su nombre en inglés, *Object-Relational mapping*, o sus siglas O/RM, ORM, y O/R *mapping*) es una técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y el utilizado en una base de datos relacional, utilizando un motor de persistencia. <http://es.wikipedia.org/wiki/Mapeo_objeto-relacional>.

[2] Interfaz de programación de aplicaciones (IPA) o API (del inglés *Application Programming Interface*) es el conjunto de funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción. Son usadas generalmente en las bibliotecas. <http://es.wikipedia.org/wiki/Interfaz_de_programaci%C3%B3n_de_aplicaciones>.

[3] En computación CRUD es el acrónimo de Crear, Obtener, Actualizar y Borrar (del original en inglés: *Create*, *Read*, *Update* and *Delete*). Es usado para referirse a las funciones básicas en bases de datos o la capa de persistencia en un software.

[4] Desarrollo guiado por pruebas de software, o *Test-driven development* (TDD) es una práctica de programación que involucra otras dos prácticas: Escribir las pruebas primero (*Test First Development*) y Refactorización (*Refactoring*). <http://es.wikipedia.org/wiki/Desarrollo_guiado_ por_pruebas>.

[5] La integración continua (*continuous integration* en inglés) es un modelo informático propuesto inicialmente por Martin Fowler que consiste en hacer integraciones automáticas de un proyecto lo más a menudo posible para así poder detectar fallos cuanto antes. Entendemos por integración la compilación y ejecución de tests de todo un proyecto. <http://es.wikipedia.org/wiki/Integraci%C3%B3n_continua>.