A. 英文原文

Key Features of SQLAlchemy

Some of the key features at a glance:

No ORM Required

SQLAlchemy consists of two distinct components, known as the Core and the ORM. The Core is itself a fully featured SQL abstraction toolkit, providing a smooth layer of abstraction over a wide variety of DBAPI implementations and behaviors, as well as a SQL Expression Language which allows expression of the SQL language via generative Python expressions. A schema representation system that can both emit DDL statements as well as introspect existing schemas, and a type system that allows any mapping of Python types to database types, rounds out the system. The Object Relational Mapper is then an optional package which builds upon the Core. Many applications are built strictly on the Core, using the SQL expression system to provide succinct and exact control over database interactions.

Mature, High Performing Architecture

Over seven years of constant development, profiling, and refactoring has led to a toolkit that is high performing and accurate, well covered in tests, and deployed in thousands of environments. With virtually every major component in its second or third full iteration, SQLAlchemy 0.6 is roughly twice the speed of older 0.4 versions from just a few years ago, and versions 0.7 and 0.8 continue to improve. Its raw execution speed is competitive with comparable tools, and advanced ORM features like its unit of work, in-memory collections, eager loading of collections via joins or secondary subselects, and other optimizations allow SQLAlchemy's ORM to emit fewer and more efficient queries than in any previous version.

DBA Approved

Built to conform to what DBAs demand, including the ability to swap out generated SQL with hand-optimized statements, full usage of bind parameters for all literal values, fully transactionalized and batched database writes using the Unit of Work pattern. All object-relational patterns are designed around the usage of proper referential integrity, and foreign keys are an integral part of its usage.

Non-Opinionated

SQLAlchemy places the highest value on not getting in the way of database and application architecture. Unlike many tools, it never "generates" schemas (not to be confused with issuing user-defined DDL, in which it excels) or relies on naming conventions of any kind. SQLAlchemy supports the widest variety of database and architectural designs as is reasonably possible.

Unit Of Work

The Unit Of Work system, a central part of SQLAlchemy's Object Relational Mapper (ORM), organizes pending insert/update/delete operations into queues and flushes them all in

one batch. To accomplish this it performs a topological "dependency sort" of all modified items in the queue so as to honor inter-row dependencies, and groups redundant statements together where they can sometimes be batched even further. This produces the maximum efficiency and transaction safety, and minimizes chances of deadlocks. Modeled after Fowler's "Unit of Work" pattern as well as Hibernate, Java's leading object-relational mapper.

Function-based query construction

Function-based query construction allows SQL clauses to be built via Python functions and expressions. The full range of what's possible includes boolean expressions, operators, functions, table aliases, selectable subqueries, insert/update/delete statements, correlated updates, selects, and EXISTS clauses, UNION clauses, inner and outer joins, bind parameters, and free mixing of literal text within expressions. Constructed expressions are compilable specific to any number of vendor database implementations (such as PostgreSQL or Oracle), as determined by the combination of a "dialect" and "compiler" provided by the implementation.

Modular and Extensible

Different parts of SQLAlchemy can be used independently of the rest. Elements like connection pooling, SQL statement compilation and transactional services can be used independently of each other, and can also be extended through various plugin points. An integrated event system allows custom code to be injected at over fifty points of interaction, including within core statement execution, schema generation and introspection, connection pool operation, object relational configuration, persistence operations, attribute mutation events, and transactional stages. New SQL expression elements and custom database types can be built and integrated seamlessly.

Separate mapping and class design

The ORM standardizes on a "Declarative" configurational system that allows construction of user-defined classes inline with the table metadata they map to, in the same way most other object-relational tools provide. However this system is totally optional - at its core, the ORM considers the user-defined class, the associated table metadata, and the mapping of the two to be entirely separate. Through the use of the mapper() function, any arbitrary Python class can be mapped to a database table or view. Mapped classes also retain serializability (pickling) for usage in various caching systems.

Eager-loading and caching of related objects and collections

The ORM caches collections and references between objects once loaded, so that no SQL need be emitted on each access. The eager loading feature allows entire graphs of objects linked by collections and references to be loaded with few or just one query, configurable down to the exact statement count on a per-mapping or per-query basis, with no changes to existing queries needed. The "N+1" problem, whereby an ORM needs to emit individual statements for all objects in a collection, is a thing of the past with SQLAlchemy.

Composite (multiple-column) primary keys

In SQLAlchemy, primary and foreign keys are represented as sets of columns; truly

composite behavior is implemented from the ground up. The ORM has industrial strength support for meaningful (non-surrogate) primary keys, including mutability and compatibility with ON UPDATE CASCADE, as well as explicit support for other common composite PK patterns such as "association" objects (many-to-many relationships with extra meaning attached to each association).

Self-referential Object Mappings

Self-referential mappings are supported by the ORM. Adjacency list structures can be created, saved, and deleted with proper cascading, with no code overhead beyond that of non-self-referential structures. Loading of self-referential structures of any depth can be tuned to load collections recursively via a single statement with a series of joins (i.e. a joinedload), or via multiple statements where each loads the full set of records at a distinct level of depth (i.e. subqueryload). Persistence with tables that have mutually-dependent foreign key pairs (i.e. "many x"/"one particular x") are also supported natively using the "post update" feature.

Inheritance Mapping

Explicit support is available for single-table, concrete-table, and joined table inheritance. Polymorphic loading (that is, a query that returns objects of multiple descendant types) is supported for all three styles. The loading of each may be optimized such that only one round trip is used to fully load a polymorphic result set.

Raw SQL statement mapping

SQLA's object relational query facilities can accommodate raw SQL statements as well as plain result sets, and object instances can be generated from these results in the same manner as any other ORM operation. Any hyper-optimized query that you or your DBA can cook up, you can run in SQLAlchemy, and as long as it returns the expected columns within a rowset, you can get your objects from it. Statements which represent multiple kinds of objects can be used as well, with results received as named-tuples, or with dependent objects routed into collections on parent objects.

Pre- and post-processing of data

The type system allows pre- and post- processing of data, both at the bind parameter and the result set level. User-defined types can be freely mixed with built-in types. Generic types as well as SQL-specific types are available.

Supported Platforms

SQLAlchemy supports Python 2.5 through the latest 3.x versions. Other supported platforms include Jython and Pypy.

Supported Databases

SQLAlchemy includes dialects for SQLite, Postgresql, MySQL, Oracle, MS-SQL, Firebird, Sybase and others, most of which support multiple DBAPIs. Other dialects are published as external projects. The corresponding DB-API 2.0 implementation (or sometimes one of several available) is required to use each particular database. View Current DBAPI Support

B. 原文的翻译

SQLAlchemy 的主要特点

一些一目了然的主要特点：

无需 ORM

SQLAlchemy 的包括两个不同的组成部分，被称为核心和 ORM。核心本身是一个全功能的 SQL 抽象的工具包，这个核心在多种 DBAPI 实现和行为上，以及允许通过 Python 表示的 SQL 语言上提供了一个光滑的抽象层。一个模式的代表性系统，既可以发出 DDL 语句类似于反射现有模式和类型系统，又可以让 Python 类型映射数据库类型，完善了制度。对象关系映射器则是一个可选包，它是建立在核心。许多应用程序都严格建立在核心，使用 SQL 表达系统提供了数据库交互简洁和详细的控制。

成熟的，高性能的架构

经历了 70 的持续发展，剖析和重构催生了一个高性能和准确的，覆盖了测试的，部署在数以千计环境中的工具包。在其第二或第三个完整的迭代几乎每一个主要组成部分，在几年前 SQLAlchemy0.6 版本大致是从 0.4 版本两倍的速度，和版本 0.7 和 0.8 仍在不断改善。其原生的执行速度已经和类似的工具相媲美，先进的 ORM 的功能，比如其工作单位，在内存中的集合，通过连接或二级子选择集合预先装载的竞争力，以及其他优化允许的 SQLAlchemy 的 ORM 发出更少的都比以前的版本更加高效。

DBA 认证

内置符合什么数据库管理员的需求，包括使用手工优化报表的换出生成的 SQL 和为所有文字值绑定参数完整的用法，使用单元工作模式完成充分的事务化和批量的数据库写入。所有的对象关系方案的设计都是围绕适当的完整性引用，和外键作为其使用的一个组成部分。

非导向性

SQLAlchemy 没有将最高的价值放在数据库和应用程序架构应用上。不像许多工具，它从来没有“生成”模式（不与发行用户定义的 DDL 混淆），或依赖于任何形式的命名约定。 SQLAlchemy 的支持最广泛的数据库和合理可行的框架设计。

工作单元

工作单元的体系，是 SQLAlchemy 的的对象关系映射（ORM）的核心部分，组织挂起插入/更新/删除等操作到队列并把他们刷新放在一个批次。要做到这一点，他将队列中

修改过的项目进行了拓扑排序，以彰显行间依赖关系，以及冗余的语句组合在一起被进一步的批处理。这将产生最大的效率和交易的安全性，并最大限度地减少死锁的可能性。“工作单元”模式和 Java 领先的关系对象模型 Hibernate 很相似。

基于函数查询的构造

基于函数的查询构造允许 SQL 子句通过 Python 函数和表达式来构建。整个范围，包括布尔表达式，运算符，函数表的别名，可选的子查询，插入/更新/删除语句，级联更新，选择和 EXISTS 语句，UNION 语句，内部和外部连接，绑定参数，以及表达式中的文字文本的自由混合。构造的表达式可编译具体到任意数量的数据库供应商实现（如 PostgreSQL 的或 Oracle）组成，由一个“语言”和“编译”的实现提供的组合决定。

模块化和可扩展

SQLAlchemy 的不同部分可以被独立的应用于其他部分。就像连接池，SQL 语句编译和事务服务元素可以彼此独立地使用，并且还可以通过各种插件分扩展。一个集成的事件系统允许自定义代码在五十多个交互点进行注射，包括核心语句执行，模式生成和反射，连接池操作中，对象关系的配置，持久化操作，属性突变事件和事务阶段。新的 SQL 表达式元素和自定义数据库类型可以构建成并实现无缝集成。

独立的映射和类的设计

在 ORM 标准化了一个“声明”构型系统，它允许用户定义的构造类的内嵌它们映射的表元数据，就像对象关系工具提供的那样。然而，该系统是完全可选的 - 在其核心，在 ORM 考虑到了用户定义的类，内联的表元数据，并且两个的映射是完全分开的。通过使用映射器的功能，任意一个 Python 类可以被映射到数据库表或图。映射类在各种缓存系统仍保留串行化的用法（pickle 模块）。

支持的平台

通过 SQLAlchemy 的最新版本 3.x 的 Python 的支持 2.5。其他支持的平台包括 Jython 和 Pypy。

支持的数据库

SQLAlchemy 支持的语言包括 SQLite 的和 PostgreSQL，MySQL 和 Oracle，MS-SQL，Firebird，Sybase 和其他，其中大部分支持多 DBAPIs 语言。其他语言发型为外部项目。相应的 DB-API 2.0 实现（或有时几个可用的一种）是需要使用每个特定的数据库。查看当前 DBAPI 支持