

Capítulo 9. API externa: integración con otros sistemas

El servidor Odoo proporciona una API externa, que es utilizada por su cliente web y está disponible para otras aplicaciones cliente. En este capítulo, aprenderemos cómo usar la API externa de Odoo desde nuestros propios programas cliente, usando la API externa de Odoo.

Para evitar la introducción de idiomas adicionales con los que el lector podría no estar familiarizado, aquí nos centraremos en clientes basados en Python, aunque las técnicas para manejar las llamadas RPC también se aplican a otros lenguajes de programación.

Describiremos cómo usar las llamadas Odoo RPC, y luego usar ese conocimiento para construir una aplicación simple de línea de comandos de la Biblioteca usando Python.

Los siguientes temas se tratarán en este capítulo:

- Configurar Python en la máquina del cliente
- Conexión a Odoo utilizando XML-RPC
- Ejecución de métodos de servidor utilizando XML-RPC
- Los métodos de búsqueda y lectura de API
- La interfaz XML-RPC del cliente de biblioteca
- La interfaz de usuario del cliente de la Biblioteca
- Usando la `OdooRPC` biblioteca
- Sobre el `ERPpeek` cliente

◀ Sección anterior Sección (/book/business/9781789532470/8/ch08lvl1sec91/further-reading)

siguiente ➤ (/book/business/9781789532470/9/ch09lvl1sec92/technical-requirements)



Requerimientos técnicos

El código de este capítulo se basa en el código creado en el Capítulo 3 (/book/business/9781789532470/3) , **Su primera aplicación Odoo** . El código necesario se puede encontrar en el `ch03/` directorio del repositorio de Git en <https://github.com/PacktPublishing/Odoo-12-Development-Essentials-Fourth-Edition> (<https://github.com/PacktPublishing/Odoo-12-Development-Essentials-Fourth-Edition>) .

Debe tenerlo en su ruta de complementos e instalar el `library_app` módulo. Los ejemplos de código supondrán que la base de datos de Odoo para trabajar es `12-library` , para ser coherente con las instrucciones de instalación que se dan en el Capítulo 2 (/book/business/9781789532470/2) , **Preparación del entorno de desarrollo** .

El código de este capítulo se puede encontrar en el mismo repositorio, en el `ch09/` directorio.

[◀ Sección anterior Sección \(/book/business/9781789532470/9\)](#)

[siguiente ▶ \(/book/business/9781789532470/9/ch09lvl1sec93/learning-project-a-client-to-catalogue-books\)](#)



Proyecto de aprendizaje: un cliente para catalogar libros

En este capítulo, trabajaremos en una aplicación cliente simple para administrar el catálogo de libros de la biblioteca. Es la aplicación de **interfaz de línea de comandos (CLI)**, que utiliza Odoocomo el backend Las características de aplicación serán muy limitados, por lo que nos podemos centrar en la tecnología utilizada para interactuar con el servidor Odoo, en lugar de la detalles de implementación de nuestra aplicación particular.

Nuestra sencilla aplicación debería poder hacer lo siguiente:

- Buscar por título y listar libros
- Agregar nuevos títulos al catálogo
- Corregir el título de un libro.
- Eliminar un libro del catálogo

La aplicación será un script de Python que espera que se ejecuten los comandos. Una sesión de uso se verá así:

Dupdo

```
$ python3 library.py add "Moby-Dick"
$ python3 library.py list "moby"
3 Moby-Dick
$ python3 library.py set-title 3 "Moby Dick"
$ python3 library.py del 3
```

◀ Sección anterior Sección (/book/business/9781789532470/9/ch09lvl1sec92/technical-requirements)

siguiente ▶ (/book/business/9781789532470/9/ch09lvl1sec94/setting-up-python-on-the-client-machine)



Configurar Python en la máquina del cliente

Se puede acceder a la API de Odoo externamente usando dos protocolos diferentes: XML-RPC y JSON-RPC. Cualquier programa externo capaz de implementar un cliente para uno de estos protocolos podrá interactuar con un servidor Odoo. Para evitar la introducción de lenguajes de programación adicionales, seguiremos usando Python para explorar la API externa.

Hasta ahora, hemos estado ejecutando código Python solo en el servidor. Esta vez, utilizaremos Python en el lado del cliente, por lo que es posible que deba realizar alguna configuración adicional en su estación de trabajo.

Para seguir los ejemplos de este capítulo, deberá poder ejecutar archivos Python en su computadora de trabajo. Usaremos Python 3 para esto. En este punto, debe asegurarse de tener Python 3 instalado en su estación de trabajo. Esto se puede confirmar ejecutando el `python3 --version` comando en una Terminal. Si no, puede remitir la página web oficial del paquete de instalación de su plataforma, en <https://www.python.org/downloads/> (<https://www.python.org/downloads/>) .

Para Ubuntu, en este punto, probablemente ya tenga instalado Python 3. Si no, puede instalarlo ejecutando lo siguiente:

Dupdo

```
$ sudo apt-get install python3 python3-pip
```

Si es un usuario de Windows y tiene Odoo instalado en su máquina, es posible que se pregunte por qué aún no tiene un intérprete de Python y por qué se necesita una instalación adicional. La respuesta breve es que la instalación de Odoo tiene un intérprete de Python incorporado que no se usa fácilmente en el exterior.



Conexión a la API de Odoo utilizando XML-RPC

El método más simple para accederEl servidor está utilizando XML-RPC. Podemos usarla `xmlrpc.lib` biblioteca de la biblioteca estándar de Python para esto. Recuerde que estamos programando un cliente para conectarse a un servidor, por lo que necesitamos que se ejecute una instancia del servidor Odoo para conectarse. En nuestros ejemplos, asumiremos que una instancia del servidor Odoo se está ejecutando en la misma máquina (`localhost`), pero puede usar cualquier dirección IP o nombre de servidor accesible, si el servidor se está ejecutando en una máquina diferente.

Hagamos el primer contacto con la API externa de Odoo. Inicie una consola Python 3 y escriba lo siguiente:

Dupdo

```
>>> from xmlrpc import client
>>> srv = 'http://localhost:8069'
>>> common = client.ServerProxy('%s/xmlrpc/2/common' % srv)
>>> common.version()
{'server_version': '12.0', 'server_version_info': [12, 0, 0, 'final', 0, ''], 'server_serie': '12.0', 'protocol_version': 1}
```

Aquí, importamos la `xmlrpc.client` biblioteca y luego configuramos una variable con la información para la dirección del servidor y el puerto de escucha. Siéntase libre de adaptarlos a su configuración específica.

A continuación, configuramos el acceso a los servicios públicos del servidor (que no requieren un inicio de sesión), expuestos en el `/xmlrpc/2/common` punto final. Uno de los métodos disponibles es el `version()` que inspecciona la versión del servidor. Lo usamos para confirmar que podemos comunicarnos con el servidor.

Otro método público es `authenticate()` . De hecho, esto no crea una sesión, como se te puede hacer creer. Este método solo confirma que el nombre de usuario y la contraseña son aceptados y devuelve la identificación de usuario que debe usarse en las solicitudes en lugar del nombre de usuario, como se muestra aquí:

Dupdo

```
>>> db = '12-library'
>>> user, pwd = 'admin', 'admin'
>>> uid = common.authenticate(db, user, pwd, {})
>>> print(uid)
2
```

Comenzamos configurando, en la `db` variable, el nombre de la base de datos que usaremos. En nuestro ejemplo, es `12-library` , pero puede cambiarlo si está trabajando con una base de datos con un nombre diferente.

Si las credenciales de inicio de sesión no son correctas, `False` se devuelve un valor, en lugar de una ID de usuario.

El `authenticate()` ultimo argumento es el entorno del agente de usuario, que se utiliza para proporcionarEl servidor con algunos metadatos sobre el cliente. Es obligatorio, pero puede ser un diccionario vacío.



Ejecución de métodos de servidor utilizando XML-RPC

Con XML-RPC, no se mantiene ninguna sesión y las credenciales de autenticación se envían con cada solicitud. Esto agrega algo de sobrecarga al protocolo, pero lo hace más fácil de usar.

A continuación, configuramos el acceso a los métodos del servidor que necesitan un inicio de sesión para acceder. Estos están expuestos en el `/xmlrpc/2/object` punto final, como se muestra a continuación:

Dupdo

```
>>> api = client.ServerProxy('%s/xmlrpc/2/object' % srv)
>>> api.execute_kw(db, uid, pwd, 'res.partner', 'search_count', [[]])
42
```

Aquí, estamos accediendo a la API del servidor por primera vez, realizando un recuento en los registros de **socios**. Los métodos se llaman usando el `execute_kw()` método que toma los siguientes argumentos:

- El nombre de la base de datos para conectarse
- La ID de usuario de conexión
- La contraseña de usuario
- El nombre del identificador del modelo de destino
- El método para llamar
- Una lista de argumentos posicionales.
- Un diccionario opcional con argumentos de palabras clave (no utilizado en el ejemplo)

El ejemplo anterior llama al `search_count` método del `res.partner` modelo con un argumento posicional `[]` y sin argumentos de palabras clave. El argumento posicional es un dominio de búsqueda; Como estamos proporcionando una lista vacía, cuenta todos los Socios.

Las acciones frecuentes son `search` y `read`. Cuando se llama desde el RPC, el `search` método devuelve una lista de ID que coinciden con un dominio. El `browse` método no está disponible desde el RPC y `read` debe usarse en su lugar para obtener una lista de ID de registro y recuperar sus datos, como se muestra en el siguiente código:

Dupdo

```
>>> domain = [('is_company', '=', True)]
>>> api.execute_kw(db, uid, pwd, 'res.partner', 'search', [domain])[14, 10, 11, 15, 12, 13, 9, 1]
>>> api.execute_kw(db, uid, pwd, 'res.partner', 'read', [[14]], {'fields': ['id', 'name', 'country_id']})
[{'id': 14, 'name': 'Azure Interior', 'country_id': [233, 'United States']}
```

Note that, for the `read` method, we're using one positional argument for the list of IDs, `[14]`, and one keyword argument, `fields`. We can also notice that many-to-one relational fields, such as `country_id`, are retrieved as a pair, with the related record's ID and display name. That's something to keep in mind when processing the data in your code.

The `search` and `read` combination is so frequent that a `search_read` method is provided to perform both operations in a single step. The same result from the previous two steps can be obtained with the following instruction:

Copy

```
>>> api.execute_kw(db, uid, pwd, 'res.partner', 'search_read', [domain], {'fields': ['id', 'name', 'country_id']})
```

El `search_read` método se comporta como `read` , pero espera un dominio como primer argumento posicional en lugar de una lista de ID. Vale la pena mencionar que el `fields` argumento sobre `read` y `search_read` no es obligatorio. Si no se proporciona, se recuperarán todos los campos. Esto puede causar cálculos caros de la función campos y una gran cantidad de datos que se recuperarán, pero probablemente nunca se utilizarán, por lo que generalmente se recomienda proporcionar una lista explícita de campos.

◀

Sección anterior Sección (/book/business/9781789532470/9/ch09lvl1sec95/connecting-to-odoo-api-using-xml-rpc)

siguiente ▶

(/book/business/9781789532470/9/ch09lvl1sec97/search-and-read-api-methods)



Buscar y leer métodos de API

Hemos visto, en el Capítulo 7 (/book/business/9781789532470/7) , **Conjuntos de registros: trabajar con datos** del modelo, los métodos de modelo más importantes utilizados para generar conjuntos de registros y cómo para escribirles Pero hay algunos más métodos modelo disponibles para acciones más específicas, como se muestra aquí:

- `read([fields])` es similar al `browse` método, pero, en cambio de un conjunto de registros, devuelve una lista de filas de datos con los campos dados como argumento. Cada fila es un diccionario. Proporciona una representación serializada de los datos que pueden enviarse a través de protocolos RPC y está destinado a ser utilizado por los programas del cliente y no en la lógica del servidor.
- `search_read([domain], [fields], offset=0, limit=None, order=None)` realiza una operación de búsqueda seguida de una lectura en la lista de registros resultante. Está destinado a ser utilizado por los clientes de RPC y les ahorra el viaje de ida y vuelta adicional que se necesita al hacerlo, `search` seguido de `read` los resultados.


Todos los demás métodos de modelo se exponen a través de RPC, excepto los prefijados con un guión bajo, que se consideran privados. Esto significa que podemos usar `create` , `write` y `unlink` modificar los datos en el servidor de la siguiente manera:

Dupdo

```
>>> x = api.execute_kw(db, uid, pwd, 'res.partner', 'create',
[{'name': 'Packt Pub'}])
>>> print(x)
51
>>> api.execute_kw(db, uid, pwd, 'res.partner', 'write',
[[x], {'name': 'Packt Publishing'}])
True
>>> api.execute_kw(db, uid, pwd, 'res.partner', 'read',
[[x], ['id', 'name']])
[{'id': 51, 'name': 'Packt Publishing'}]
>>> api.execute_kw(db, uid, pwd, 'res.partner', 'unlink', [[x]])
True
>>> api.execute_kw(db, uid, pwd, 'res.partner', 'read', [[x]])
[]
```

Una limitación del protocolo XML-RPC es que no admite `None` valores. Hay una extensión XML-RPC que admite `None` valores, pero si está disponible dependerá de la biblioteca XML-RPC particular que se utilice en su cliente. Los métodos que no devuelven nada pueden no ser utilizables a través de XML-RPC, ya que están regresando implícitamente `None` . Es por eso que los métodos siempre deben terminar con al menos una `return True` declaración. Otra alternativa es utilizar JSON-RPC en su lugar, también compatible. La biblioteca OdooRPC lo permite, y la usaremos más adelante en este capítulo, en la sección **Uso de la biblioteca OdooRPC** .

Vale la pena repetir que la mayoría de los lenguajes de programación pueden utilizar la API externa de Odoo. En la documentación oficial, podemos encontrar ejemplos prácticos para Ruby, PHP y Java. Está disponible en <https://www.odoo.com/documentation/12.0/webservices/odoo.html> (<https://www.odoo.com/documentation/12.0/webservices/odoo.html>) .



Nota

Los métodos del modelo con un guión bajo se consideran privados y no se exponen a través de XML-RPC.



[◀ Sección anterior](#) Sección (/book/business/9781789532470/9/ch09lvl1sec96/running-server-methods-using-xml-rpc) siguiente [▶ \(/book/business/9781789532470/9/ch09lvl1sec98/the-library-client-xml-rpc-interface\)](#)



La interfaz XML-RPC del cliente de biblioteca

Comencemos con la implementación de la aplicación cliente de la Biblioteca. Lo dividiremos en dos archivos: uno que se ocupa de la interfaz con el servidor del servidor `library_api.py` y otro que se ocupa de la interfaz de usuario de la aplicación `library.py`. Entonces vamos a ofrecer implementación alternativa para la interfaz de gestión, el uso de la biblioteca existente, `OdooRPC`.

Crearemos una clase para configurar la conexión con un servidor Odoo, y leer / escribir datos del Libro de la Biblioteca. Debería exponer los métodos básicos de CRUD:

- `search_read()` para recuperar datos del libro
- `create()` para crear libros
- `write()` para actualizar libros
- `unlink()` eliminar un libro

Elija un directorio para alojar los archivos de la aplicación y cree el `library_api.py` archivo. Comenzamos agregando el constructor de clase, de la siguiente manera:

Dupdo

```
from xmlrpc import client

class LibraryAPI():
    def __init__(self, srv, port, db, user, pwd):
        common = client.ServerProxy(
            'http://%s:%d/xmlrpc/2/common' % (srv, port))
        self.api = client.ServerProxy(
            'http://%s:%d/xmlrpc/2/object' % (srv, port))
        self.uid = common.authenticate(db, user, pwd, {})
        self.pwd = pwd
        self.db = db
        self.model = 'library.book'
```

Aquí, almacenamos toda la información necesaria en el objeto creado para ejecutar llamadas en un modelo: la referencia de API `uid`, la contraseña, el nombre de la base de datos y el modelo a utilizar.

A continuación, definiremos un método auxiliar para ejecutar las llamadas. Aprovecha los datos almacenados del objeto para proporcionar una firma de función más pequeña, como se muestra a continuación:

Dupdo

```
def execute(self, method, arg_list, kwarg_dict=None):
    return self.api.execute_kw(
        self.db, self.uid, self.pwd, self.model,
        method, arg_list, kwarg_dict or {})
```

Ahora podemos usarlo para implementar los métodos de nivel superior.

El `search_read()` método aceptará una lista opcional de ID para recuperar. Si no aparece ninguno, se devolverán todos los registros:

Dupdo

```
def search_read(self, text=None):
    domain = [('name','ilike', text)] if text else []
    fields = ['id', 'name']
    return self.execute('search_read', [domain, fields])
```

El `create()` método creará un nuevo libro con el título dado y devuelve la ID del registro creado:

Dupdo

```
def create(self, title):
    vals = {'name': title}
    return self.execute('create', [vals])
```

El `write()` método tendrá el nuevo título y la ID del libro como argumentos y realizará una operación de escritura en ese libro:

Dupdo

```
def write(self, title, id):
    vals = {'name': title}
    return self.execute('write', [[id], vals])
```

Y luego tenemos la `unlink()` implementación del método, que es bastante simple:

Dupdo

```
def unlink(self, id):
    return self.execute('unlink', [[id]])
```

Terminamos el archivo con un pequeño fragmento de código de prueba que se ejecutará si ejecutamos el archivo Python:

Dupdo

```
if __name__ == '__main__':
    # Sample test configurations
    srv, db, port = 'localhost' , '12-library' , 8069
    user, pwd = 'admin', 'admin'
    api = LibraryAPI(srv, port, db, user, pwd)
    from pprint import pprint
    pprint(api.search_read())
```

Si ejecutamos el script Python, deberíamos ver el contenido de nuestros libros de la biblioteca impresos:

Dupdo

```
$ python3 library_api.py
[{'id': 2, 'name': 'Odoo 11 Development Cookbook'},
 {'id': 1, 'name': 'Odoo Development Essentials 11'}]
```

Ahora que tenemos un contenedor simple alrededor de nuestro backend de Odoo, tratemos con la interfaz de usuario de línea de comandos.

◀ Sección anterior Sección (/book/business/9781789532470/9/ch09lvl1sec97/search-and-read-api-methods)

siguiente ▶ (/book/business/9781789532470/9/ch09lvl1sec99/the-library-client-user-interface)



La interfaz de usuario del cliente de la Biblioteca

Nuestro objetivo aquí era aprender a escribir la interfaz entre una aplicación externa y el servidor Odoo, y esto se hizo en la sección anterior. Pero sería una pena no dar un paso más y realmente ponerlo a disposición del usuario final.

Para mantener la configuración tan simple como sea posible, utilizaremos las funciones integradas de Python para implementar la aplicación de línea de comandos. Como es parte de la biblioteca estándar, no requiere ninguna instalación adicional.

Ahora, junto con el `library_api.py` archivo, cree un nuevo `library.py` archivo. Primero importará el analizador de argumentos de línea de comandos de Python y luego la `LibraryAPI` clase, como se muestra en el siguiente código:

Dupdo

```
from argparse import ArgumentParser
from library_api import LibraryAPI
```

A continuación, describimos los comandos que el analizador de argumentos esperará; hay cuatro comandos:

- Buscar y enumerar libros
- Agregar un libro
- Establecer (cambiar) el título de un libro
- Eliminar un libro

Este es el código para agregarlos al analizador de línea de comandos:

Dupdo

```
parser = ArgumentParser()
parser.add_argument(
    'command',
    choices=['list', 'add', 'set', 'del'])
parser.add_argument('params', nargs='*') # optional args
args = parser.parse_args()
```

En este punto, `args` es un objeto que contiene los argumentos dados a la secuencia de comandos, `args.command` es el comando proporcionado, y `args.params` opcionalmente tendrá parámetros adicionales para usar para el comando. Si no se dan comandos incorrectos o no, el analizador de argumentos lo manejará por nosotros y le mostrará al usuario qué entrada se espera. Para una referencia completa sobre `argparse`, puede referirse a la documentación oficial en <https://docs.python.org/3/library/argparse.html> (<https://docs.python.org/3/library/argparse.html>).

El siguiente paso es realizar lo previsto comportamiento. Primero prepararemos la conexión con el servidor Odoo:

Dupdo

```
srv, port, db = 'localhost', 8069, '12-library'
user, pwd = 'admin', 'admin'
api = LibraryAPI(srv, port, db, user, pwd)
```

La primera línea establece algunos parámetros fijos para la instancia del servidor y la base de datos para conectarse. En este ejemplo, nos estamos conectando a un servidor Odoo en la misma máquina (`localhost`) que escucha en el `8069` puerto predeterminado, con una `12-library` base de datos disponible. Si desea conectarse a un servidor y una base de datos diferentes, debe adaptar estos parámetros en consecuencia.

Tiene una dirección de servidor codificada y una contraseña de texto sin formato, por lo que está lejos de ser la mejor implementación. Deberíamos tener un paso de configuración para recopilar estas configuraciones del usuario y posiblemente almacenarlas de forma segura. Pero debemos tener en cuenta que nuestro objetivo aquí es aprender a trabajar con el RPC de Odoo, así que considere esto como un código de prueba de concepto y no como un producto terminado.

Ahora escribiremos el código para manejar cada uno de los comandos compatibles, que también harán uso del `api` objeto.

Podemos comenzar con el `list` comando, proporcionando una lista de los libros:

Dupdo

```
if args.command == 'list':
    books = api.search_read(args.text)
    for book in books:
        print('%(id)d %(name)s' % book)
```

Aquí, utilizamos el `LibraryAPI.search_read()` método para recuperar la lista de registros del Libro del servidor. Luego iteramos a través de cada elemento de la lista y lo imprimimos. Utilizamos el formato de cadena de Python para presentar cada `book` registro, un diccionario de valores clave, al usuario.

A continuación, tenemos el comando agregar. Esto hará uso de los parámetros adicionales, para los títulos de los libros:

Dupdo

```
if args.command == 'add':
    for title in args.params:
        new_id = api.create(title)
        print('Book added with ID %d.' % new_id)
```

Dado que el trabajo duro ya se realizó en el `LibraryAPI` objeto, aquí solo necesitamos llamar al `write()` método y mostrar el resultado al usuario final.

El `set` comando permite cambiarEl título de un libro existente. Debe tener dos parámetros, el nuevo título y la ID del libro:

Dupdo

```
if args.command == 'set-title':
    if len(params) != 2:
        print("set command requires a Title and ID.")
    return
    book_id title = int(args.params[0]), args.params[1]
    api.write(title, book_id)
    print('Title set for Book ID %d.' % book_id)
```

Finalmente, tenemos la implementación del `del` comando, que debería eliminar un registro del Libro. En este punto, esto no debería ser un desafío para nosotros:

Dupdo

```
if args.command == 'del':
    for param in params:
        api.unlink(int(param))
        print('Book with ID %s deleted.' % param)
```

En este punto, la API CLI básica está terminada, y el lector puede probar algunos comandos para verificar cómo está funcionando. Por ejemplo, ahora deberíamos poder ejecutar los comandos de ejemplo que se muestran al comienzo de este capítulo, en la sección **Proyecto de aprendizaje: un cliente para catalogar libros** . Accediendo a los datos en la aplicación Biblioteca usando el cliente web normal también sería útil para confirmar que la aplicación CLI funciona como se esperaba.

Esta es una aplicación bastante básica, y probablemente podría pensar en algunas maneras de mejorarla mientras revisa el código. Pero recuerde que el punto aquí es hacer un ejemplo de formas interesantes de aprovechar la API de Odoo RPC.

◀

Sección anterior Sección (/book/business/9781789532470/9/ch09lvl1sec98/the-library-client-xml-rpc-interface)

siguiente ▶

(/book/business/9781789532470/9/ch09lvl1sec100/using-the-odoorpc-library)



Usando la biblioteca OdooRPC

Otra biblioteca cliente relevante a considerares `odoorpc` . Es una biblioteca cliente más moderna que utiliza el protocolo JSON-RPC en lugar de XML-RPC. De hecho, el cliente web oficial de Odoo usa JSON-RPC, y el XML-RPC original es compatible principalmente para la compatibilidad con versiones anteriores.



Nota

La `OdooRPC` biblioteca ahora está bajo el paraguas de la Asociación Comunitaria Odoo y está siendomantenido activamente Puede obtener más información al respecto en <https://github.com/OCA/odoorpc> (<https://github.com/OCA/odoorpc>) .

La `odoorpc` biblioteca se puede instalar desde PyPI:

Dupdo

```
$ pip3 install --user odoorpc
```

La forma en que se utiliza la API de Odoo no es muy diferente si está utilizando JSON-RPC o XML-RPC. Entonces, verá que, aunque algunos detalles difieren, la forma en que se usan estas bibliotecas de cliente diferentes no es muy diferente.

La `OdooRPC` biblioteca configura una conexión de servidor cuando `odoorpc.ODOO` se crea un nuevo objeto, y luego debemos usar el `ODOO.login()` método para crear una sesión de usuario. Al igual que en el lado del servidor, la sesión tiene un `env` atributo con el entorno de la sesión, incluido el ID de usuario `uid` ,y `context` .

Podemos usar `OdooRPC` para proporcionar una implementación alternativa a la `library_api.py` interfaz con el servidor. Debe proporcionar las mismas características, pero se implementa utilizando JSON-RPC en lugar de XML-RPC.

Cree un nuevo `library_odoorpc.py` archivo junto a él, con el siguiente código:

Dupdo

```
from odoorpc import ODOO

class LibraryAPI():

    def __init__(self, srv, port, db, user, pwd):
        self.api = ODOO(srv, port=port)
        self.api.login(db, user, pwd)
        self.uid = self.api.env.uid
        self.model = 'library.book'
        self.Model = self.api.env[self.model]

    def execute(self, method, arg_list, kwarg_dict=None):
        return self.api.execute(
            self.model,
            method, *arg_list, **kwarg_dict)
```

La `OdooRPC` biblioteca implementa `Model` y `Recordset` objetos que imitan el comportamiento de los homólogos del lado del servidor. El objetivo es que programar al cliente debe ser casi lo mismo que programar en el servidor. Los métodos utilizados por nuestro cliente harán uso de esto, a través del modelo de libro de biblioteca almacenado en el `self.Model` atributo.

El `execute()` método implementado aquí no será utilizado por nuestro cliente y se incluyó para permitir la comparación con las otras implementaciones alternativas que se discuten en este capítulo.

A continuación, nos fijamos en la aplicación de los `search_read()` , `create()` , `write()` y `unlink()` métodos cliente. En el mismo archivo, agregue estos métodos dentro de la `LibraryAPI()` clase:

Dupdo

```
def search_read(self, text=None):
    domain = [('name','ilike', text)] if text else []
    fields = ['id', 'name']
    return self.Model.search_read(domain, fields)

def create(self, title):
    vals = {'name': title}
    return self.Model.create(vals)

def write(self, title, id):
    vals = {'name': title}
    self.Model.write(id, vals)

def unlink(self, id):
    return self.Model.unlink(id)
```

Tenga en cuenta que el código se parece al código del lado del servidor de Odoo, porque usa una API que es similar a lo que podría escribir en un complemento de Odoo.

Una vez hecho esto, podemos probarlo editando el `library.py` archivo, cambiando la `from library_api import LibraryAPI` línea a `from library_odoorpc import LibraryAPI` . Ahora pruebe la `library.py` aplicación cliente y debería funcionar igual que antes.

- ◀ Sección anterior Sección (/book/business/9781789532470/9/ch09lvl1sec99/the-library-client-user-interface)

siguiente ▶ (/book/business/9781789532470/9/ch09lvl1sec101/about-the-erppeek-client)



Sobre el cliente ERPpeek

ERPpeek es un versátilherramienta que se puede utilizar como una **interfaz interactiva de línea de comandos (CLI)** y como una **biblioteca de Python** , con una API más conveniente que la proporcionada por la `xmlrpc` biblioteca. Está disponible en el índice PyPi y puede ser instalado con lo siguiente:

Dupdo

```
$ pip3 install --user erppeek
```

No solo se `ERPpeek` puede usar como una biblioteca de Python, también es una CLI que se puede usarpara realizar acciones administrativas en el servidor. Cuando el `shell` comando Odoo proporcionó una sesión interactiva local en el servidor host, la `erppeek` biblioteca proporciona una sesión interactiva remota para un cliente a través de la red.

Al abrir una línea de comando, podemos echar un vistazo a las opciones disponibles, como se muestra aquí:

Dupdo

```
$ erppeek --help
```

Veamos una sesión de muestra, como sigue:

Dupdo

```
$ erppeek --server='http://localhost:8069' -d 12-library -u adminUsage (some commands):
  models(name)           # List models matching pattern
  model(name)             # Return a Model instance
(...)
Password for 'admin':
Logged in as 'admin'
12-library >>> model('res.users').count()
3
12-library >>> rec = model('res.partner').browse(14)
12-library >>> rec.name
'Azure Interior'
```

Como puede ver, se estableció una conexión con el servidor y el contexto de ejecución proporcionó una referencia al `model()` método para obtener instancias de modelos y realizar acciones en ellos.

La `erppeek.Client` instancia utilizada para la conexión también está disponible a través de la `client` variable.

En particular, proporciona una alternativa al cliente web para administrar los módulos adicionales instalados:

- `client.modules()` enumera los módulos disponibles o instalados
- `client.install()` realiza la instalación del módulo
- `client.upgrade()` realiza actualizaciones de módulos
- `client.uninstall()` desinstala módulos

Por lo tanto, `ERPpeek` también puede proporcionar un buen servicio como herramienta de administración remota para servidores Odoo.

Se `ERPpeek` pueden encontrar más detalles sobre <https://github.com/tinyerp/erppeek> (<https://github.com/tinyerp/erppeek>) .

siguiente ➤ (/book/business/9781789532470/9/ch09lvl1sec102/summary)



Resumen

Nuestro objetivo para este capítulo fue aprender cómo funciona la API externa y de qué es capaz. Comenzamos a explorarlo usando un simple cliente Python XML-RPC, pero la API externa se puede usar desde cualquier lenguaje de programación. De hecho, la documentación oficial proporciona ejemplos de código para Java, PHP y Ruby.

Hay varias bibliotecas para manejar XML-RPC o JSON-RPC, algunas genéricas y otras específicas para usar con Odoo. Nosotros mostramos una biblioteca particular `OdooRPC`.

Con esto, terminamos los capítulos dedicados a la API de programación y la lógica empresarial. Ahora, es hora de entrar en las vistas y la interfaz de usuario. En el próximo capítulo, veremos con más detalle las vistas de back-end y la experiencia del usuario que el cliente web puede proporcionar de inmediato.



Otras lecturas

Este material de referencia adicional puede complementar los temas descritos en este capítulo:



- ▶ La documentación oficial sobre los servicios web de Odoo, incluidos ejemplos de código en lenguajes de programación distintos de Python: <https://www.odoo.com/documentation/12.0/webservices/odoo.html> (<https://www.odoo.com/documentation/12.0/webservices/odoo.html>)
- ▶ La `OdooRPC` documentación: <https://pythonhosted.org/OdooRPC/> (<https://pythonhosted.org/OdooRPC/>)
- ▶ La `ERPpeek` documentación: <https://erppeek.readthedocs.io/en/latest/> (<https://erppeek.readthedocs.io/en/latest/>)

◀ Sección anterior Sección (/book/business/9781789532470/9/ch09lvl1sec102/summary)

siguiente ▶ (/book/business/9781789532470/10)

