Debugovací HTTP Proxy

Jakub Arnold

Abstrakt

Tématem zápočtového programu je implementace debugovací HTTP proxy, která umožnuje transparentně zachytit/upravit nešifrovanou komunikaci protokolu HTTP a zjednodušit tak vývoj síťových aplikací používajících HTTP.

1 Kompilace a spuštění

Pro úspěšnou kompilaci je potřeba .NET framework verze 4.5 a vyšší, a to z důvodu použití funkce async/await [1], která zjednodušuje práci s asynchroním zpracováním HTTP požadavků. Také je potřeba nainstalovat knihovnu Json.NET [2], která se používá pro serializaci HTTP požadavků. Tato instalace proběhne ve Visual Studiu (příp. MonoDevelop) automaticky při první kompilaci, případně se dá pustit ručně:

1. Pravým kliknutím na projekt (ne solution)
2. Manage NuGet packages

a nebo případně:

1. Tools
2. Options
3. NuGet package manager
4. Allow NuGet to download missing packages

Požadovaná knihovna je velmi malá a slouží pouze pro parsování a serializaci v jednom konkrétním místě aplikace (třída *ProxyStore*.)

2 Rozsah implementace

Program pracuje v textovém režimu konzole, a je možné jej spustit v jednom ze dvou režimů.

Interaktivní
: Všechny HTTP požadavky jsou pozastaveny a uživatel může rozhodnout, zda je přeposlat dál, případně upravit před přeposláním.

Neinteraktivní
: Všechny HTTP požadavky jsou automaticky přeposlány a program pouze zaznamenává průběh komunikace.

Zachycené HTTP požadavky jsou automaticky ukládány do textových souborů v pracovním adresáři programu (ve formátu JSON), což umožnuje uživateli kdykoliv program ukončit, a při novém spuštění bude mít všechno ve stejném stavu, jako při původním spuštění. Soubory je také možné jednoduše ručně upravit (pouze pokud je program vypnutý), a změny se projeví při dalším spuštění.

Se zachycenými požadavky jsou možné následující operace

Modifikace hlaviček
: Hlavičky HTTP požadavku jsou zpracovávány jednotlivě, což umožnuje uživateli libovolnou z nich změnit/smazat, případně přidat nějaké další.

Přeposlání požadavku
: Každý uložený (a případně upravený) požadavek je možné (znovu) poslat na cílový server.

Úprava těla požadavku
: Tělo požadavku je také zpracováno zvlášť, což uživateli umožnuje jej libovolně změnit.

3 Struktura programu

Navigace programu má následující strukturu

- Main menu
  - Start/stop proxy
  - Seznam všech požadavků
    - Seznam požadavků (možno zobrazit detail)
      - Seznam hlaviček
      - Odpověď serveru (pokud je dostupná)
      - Možnost přehrát požadavek
      - Úprava hlaviček/těla požadavku
    - Smazat vše
    - Obnovit
  - Quit

Zachycené požadavky (instance třídy *IncomingHttpRequest*) jsou automaticky ukládány na disk a znovu načteny při spuštění programu. Je tedy teoreticky možné zachytit probíhající komunikaci, vypnout program, další den se k němu vrátit a ty samé požadavky znovu ručně přehrát.

Zde je však omezení, že původní požadavek už klientovi zpátky nepříjde, protože po ukončení programu se uzavře spojení.

4 Implementace a použité knihovny

Program závisí pouze na knihovnách standardně distribuovaných s platformou .NET, konkrétně jmenný prostor *System.Net*, který obsahuje nizkoúrovňovou implementaci samotného HTTP serveru (konkrétně *System.Net.HttpListener*), a knihovně Json.NET [2].

Samotný program je pak rozdělen do dvou assembly, kde jedna tvoří samotnou logiku HTTP proxy (*Boxxy.Core*), a druhá je uživatelské prostředí v příkazové řádce (*Boxxy.CLI*). Zpracování HTTP požadavků probíhá v odděleném vlákně, a uživatel tedy může program používat, zatímco příchozí požadavky jsou zpracovány na pozadí.

Třída *Boxxy.Core.ProxyStore* poté vše udržuje v paměti, a zároveň při jakékoliv změně synchronizuje uložené požadavky na disk. Uživatel se tedy o ukládání nemusí vůbec starat, všechno probíhá automaticky. Samotné požadavky jsou zapouzdřeny do třídy *Boxxy.Core.IncomingHttpRequest*, která ukládá všchny informace o požadavku, včetně informace zda byl odeslán na cílový server (a tedy zda je dostupná odpověď), a zároveň zda byla konkrétní instance načtena z disku. Tato informace má velký vliv na to, zda je možné původnímu odesilateli přeposlat zpátky odpověď ze serveru, protože pro deserializované objekty už není dostupný původní socket, přes který se klient připojil.

Samotné hlavičky a tělo požadavku jsou také načteny kompletně do instancí *Boxxy.Core.IncomingHttpRequest*, a není tedy možné požadavky streamovat z klienta přímo na server. Následkem je sice potenciálně pomalejší průběh komunikace, což ale programu určenému pro debugování nevadí.

4.1 Implementace uživatelského rozhranní

Veškerá komunikace uživatele s programem probíhá pomocí jednoduchého konzolového uživatelského rozhranní, které je strukturálně rozděleno na několik *obrazovek*, resp. menu mezi kterými se uživatel může pohybovat.

Základní stavební prvek je interface *IScreen*, které všechny třídy reprezentující jednotlivá menu implementují.

Víceúrovňová navigace v menu je poté řešena zanořeným voláním patřičných *IScreen.Run()*, kde v rámci průběhu jedné iterace obrazovky je spuštěna jiná, která kompletně převezme kontrolu, až dokud se uživatel nerozhodne vrátit na původní obrazovku.

Všechny třídy implementující *IScreen* fungují na podobném principu (převzato z principu fungování update/render smyček u grafických programů/her), a to že

drží nějaký vnitřní stav který ovlivňuje vykreslení obrazovky, a při jakékoliv interakci s uživatelem se celé menu smaže a překreslí (za pomoci *System.Console.Clear*.)

Tímto způsobem je možné jednoduše docílit rozhranní, které působí velmi interaktivně, a to s použitím *Console.ReadKey()*, který umožní rovnou reagovat na stisknutou klávesu bez nutnosti zmáčknout Enter (na rozdíl od *Console.Read()*, který používá nějaký vnitřní input buffer a vrací hodnotu až po stisku Enteru.) Tímto je možné uživatelské rozhranní překreslit okamžitě když uživatel stiskne klávesu, viz. např. úvodní obrazovka při spuštění proxy.

Reference

[1] Asynchronous Programming with Async and Await, https://msdn.microsoft.com/en-us/library/hh191443.aspx

[2] Json.NET, http://www.newtonsoft.com/json