InternetDomainName

Introduction

InternetDomainName is a useful tool for parsing and manipulating domain names. It can be used as a validator, a component extractor, and as a value type for passing around domain names in a type-safe way.

However, there are some aspects of InternetDomainName behavior which may be surprising, and which can lead to bugs in calling code. This document addresses these concerns.

Details

Public suffixes and private domains

An InternetDomainName object is guaranteed to be syntactically valid according to relevant RFC specifications, but it is **not** guaranteed to correspond to an actual addressable domain on the Internet. It is impossible to do that without doing a net lookup of the domain and trying to contact it, and that is unacceptable overhead for most common cases.

Still, it is often very useful to determine whether a given domain name **might** represent an actual domain on the Internet. For this purpose, we use data from the Public Suffix List (PSL), a list maintained by the Mozilla Foundation. There are methods on InternetDomainName to determine the relationship of a given domain to the PSL. To put it in its most basic terms, if domain.hasPublicSuffix() returns true, then the domain **might** correspond to a real Internet address; otherwise, it almost certainly does not.

At this point we need to back up and define some terms. There are four terms of interest:

- Top-Level Domain (TLD): A single-label domain with no children, such as com or au.
- Registry suffix: A domain, such as com or co.uk, controlled by a domain name registry (e.g. Verisign for com), under which people can register subdomains through a domain name registrar (e.g. Namecheap). Such domain name registrations are legally protected by internet governing bodies like ICANN.
- Public suffix: This category is a superset of the registry suffixes, additionally including suffixes like blogspot.com that are not registry-controlled but allow the public to register subdomains. There are several common cases where it is more appropriate to categorize domains by public suffix rather than registry suffix. For example, one should never set cookies on a public suffix.
- Effective Top-Level Domain: A deprecated synonym for “public suffix”.

It’s worth reading the linked articles carefully before proceeding.

A major source of confusion is that people say “TLD” when they mean either “registry suffix” or “public suffix”. All three of these are independent concepts. So, for example,

- `com` is all three: a TLD, a registry suffix, and a public suffix
- `au` is a TLD, but not a registry suffix or public suffix
- `com.au` is a registry suffix and public suffix, but not a TLD
- `blogspot.com` is a public suffix, but neither a registry suffix nor a TLD
- `squerf` is none of the three

This confusion is especially dangerous because TLD and registry suffix have crisp, formal definitions, while public suffix does not. In the end, a public suffix is something that a credible source has asked the PSL maintainers to add to the list. Credible sources include ICANN and country-domain managers, but also include private companies offering services that share the characteristics that (fuzzily) define a public suffix – independent subdomains and supercookie suppression. So, for example, many Google-owned domains (e.g. `blogspot.com`) are included in the PSL.

Getting back to `InternetDomainName`, as long as we limit ourselves to using `hasPublicSuffix()` to validate that the domain is a plausible Internet domain, all is well. The danger arises from the methods that identify or extract the “top private domain”. From a technical point of view, the top private domain is simply the rightmost superdomain preceding the public suffix. So for example, `www.foo.co.uk` has a public suffix of `co.uk`, and a top private domain of `foo.co.uk`.

As noted in the documentation on `isUnderPublicSuffix()`, `isTopPrivateDomain()`, and `topPrivateDomain()`, the only thing these methods are (mostly) reliable for is determining where one can set cookies. However, what many people are actually trying to do is find the “real” domain, or the “owner” domain, from a subdomain. For example, in `mail.google.com` they would like to identify `google.com` as the owner domain. So they write

```
InternetDomainName owner =
    InternetDomainName.from("mail.google.com").topPrivateDomain();
```

. . . and sure enough, `owner` ends up with the domain `google.com`. Indeed, this idiom (and ones like it) work a great deal of the time. It seems intuitive that “the domain under the public suffix” should be semantically equivalent to “the owner domain”.

But it’s not, and therein lies the problem. Consider `blogspot.com`, which appears in the PSL. While it has the characteristics of a public suffix – people can register domains under it (for their blogs), and cookies should not be allowed to be set on it (to prevent cross-blog cookie shenanigans), it is itself an addressable domain on the Internet (which happens to redirect to `blogger.com` at time of writing, but that could easily change).

So if one uses the idiom above on `foo.blogspot.com`, `owner` will be that same domain, foo.blogspot.com. This is the right answer for cookie-setting any many common applications, but it is obviously surprising to many people.

For those rare applications where the goal is actually to determine the domain name that was purchased from a registry, the correct abstraction is not public suffix but registry suffix. If we alter the above code to use the parallel methods for registry suffixes:

```
InternetDomainName owner =
    InternetDomainName.from("foo.blogspot.com").topDomainUnderRegistrySuffix();
```

…then we end up with owner set to blogspot.com.

This handy table summarizes the differences between the various types of suffix:

InternetDomainName	publicSuffix()	topPrivateDomain()	registrySuffix()	topDomainUnderRegistrySuffix()
google.com	com	google.com	com	google.com
co.uk	co.uk	N/A	co.uk	N/A
www.google.co.uk	co.uk	google.co.uk	co.uk	google.co.uk
foo.blogspot.com	blogspot.com	foo.blogspot.com	com	blogspot.com
google.invalid	N/A	N/A	N/A	N/A

The big lessons here are:

- TLDs, registry suffixes, and public suffixes are not the same thing.
- Public suffixes are defined by humans, for strictly limited purposes (mostly domain validation and supercookie prevention), and change unpredictably.
- There is no defined mapping between the relationship of a given domain to a public suffix, the ability of that domain to respond to web requests, and the "ownership" of that domain.
- The registry suffix can help you determine the "ownership" of a domain, in the sense of ICANN-style domain registrations; but this information is less suitable than the public suffix for most applications.
- You can use InternetDomainName to determine whether a given string represents a plausibly addressable domain on the Internet, and to determine what portion of a domain is likely to allow cookies to be set.
- You cannot use InternetDomainName to determine if a domain exists on the Internet as an addressable host.

Remember that if you do not heed this advice, your code will appear to work on a huge variety of inputs… but the failure cases are all bugs just waiting to happen, and the set of failure cases will change as PSL updates are incorporated into the code underlying InternetDomainName.